Formal Reasoning

Herman Geuvers
Partly based on the reader of fall of 2002 by Henk Barendregt and Bas Spitters,
and with thanks to Wim Gielen for his reader "Discrete Wiskunde."
Revised and expanded, fall of 2008, by Engelbert Hubbers and Freek Wiedijk.
Translated fall of 2016 by Kelley van Evert.

December 12, 2016

Contents

References

[1] W. Gielen. Discrete wiskunde voor informatici. Katholieke Universiteit Nijmegen - Subfaculteit Wiskunde 2002.

[2] M. Huth and M. Ryan. *Logic in Computer Science.* Cambridge University Press, 2004. ISBN 0-521-54310-X.

[3] J.F.A.K. van Benthem, H.P. van Ditmarsch, J.S. Lodder, J. Ketting, and W.P.M. Meyer-Viol. *Logica voor informatici.* Pearson Addison-Wesley, Nederland, 2003. ISBN 90-430-0722-6 (dit boek wordt ook gebruikt in het college Beweren en Bewijzen).

[4] R.J. Wilson. *Introduction to Graph Theory.* Pearson Education Limited, fifth edition, 2010. ISBN 978-0-273-72889.

1 Propositional logic

In this chapter we discuss propositional logic, often also called propositional calculus.

1.1 Formal languages and natural languages

Natural languages, such as Dutch, English, German, etc., are not always as exact as one would hope. Take a look at the following examples:

- Socrates is a human being. Human beings are mortal. So, Socrates is mortal.
- I am someone. Someone painted the Mona Lisa. So, I painted the Mona Lisa.

The first statement is correct, but the second is not. Even though they share the same form.

And what about the sentence,

This sentence is not true.

Is it true, or not?

To avoid these kinds of problems, we use formal languages. Formal languages are basically laboratory-sized versions, or models, of natural languages. But we'll see that these artificial languages, even though they're relatively simple, can be used to express statements and argumentations in a very exact and unambiguous manner. And this is very important for many applications, such as describing the semantics of a program. Such specifications should obviously not lead to misunderstandings.

To start, we'll show how we make the transition from the English language to a formal language. In reasoning, we often combine small statements to form bigger ones, as in for instance: '*If* it rains *and* I'm outside, *then* I get wet.' In this example, the small statements are 'it rains', 'I'm outside', and 'I get wet.'

1.2 Dictionary

We can also formalize this situation with a small dictionary.

R	it rains
S	the sun shines
RB	there is a rainbow
W	I get wet
D	I stay dry
Out	I'm outside
In	I'm inside

So then, the sentence we introduced above becomes 'if R and Out, then W.' Similarly, we could form statements as: 'if RB, then S', 'S and RB', or 'if R and In, then D.'

1.3 Connectives

We can also translate the connectives, which we'll do like this:

Formal language	English
$f \wedge g$	f and g
$f \vee g$	f or g, or both
$f \to g$	if f, then g
$f \leftrightarrow g$	f if and only if g[1]
$\neg f$	not f

Now, the sentences we formed above become 'RB→S', 'S∧RB', and '(R∧In)→D.'

English	Semi-formal	Formal
If it rains *and* I'm outside, *then* I get wet.	If R and Out, then W.	$(R \wedge \text{Out}) \to W$
If there is a rainbow, *then* the sun shines.	If RB, then S.	$RB \to S$
I'm inside *or* outside, or both.	In or Out, or both.	$\text{In} \vee \text{Out}$

Exercise 1.A Form sentences in our formal language that correspond to the following English sentences:
(i) It's neither raining, nor is the sun shining.
(ii) The sun shines unless it rains
(iii) Either the sun shines, or it rains. (But not both simultaneously.)
(iv) There is only a rainbow if the sun is shining and it's raining.
(v) If I'm outside, I get wet, but only if it rains.

So far, we've modelled our symbols (like ∨ and ∧) to be very much like the English connectives. One of these English connectives is the word 'or,' which combines smaller statements into larger, new statements. However, the meaning of the English word 'or' can be a bit ambiguous. Take the following proposition: '$1 + 1 = 2$ or $2 + 3 = 5$.' Is it true, or not? It turns out that people sometimes differ in opinion on this, and when you think about it a bit, there are two distinct meanings that 'or' can have in the English language. Students of Information Science and Artificial Intelligence obviously don't like these ambiguities, so we'll simply choose the meaning of ∨ to be one of the two usual meanings of 'or': we'll agree that 'A or B' is also true in the case that both A and B are true in of themselves. Definition 1.5 will formalize this agreement.

Exercise 1.B Can you also express $f \leftrightarrow g$ using the other connectives? If so, show how.

Exercise 1.C Translate the following formal sentences into English:
(i) $R \leftrightarrow S$
(ii) $\text{RB} \to (R \wedge S)$
(iii) $\text{Out} \to \neg\,\text{In}$
(iv) $\text{Out} \vee \text{In}$

Definition 1.1 The language of propositional logic is defined to be as follows. Let A be an infinite collection of *atomic propositions* (also sometimes called *propositional variables* or *letters*):

$$A := \{a, b, c, d, a_1, a_2, a_3, \ldots\}$$

[1] Mathematicians will often simply write "iff" instead of the lengthier "if and only if."

Let V be the set of *connectives*:

$$V := \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$$

Let H be the set of *parentheses*:

$$H := \{(,)\}$$

Then, let the *alphabet* be the set $\Sigma := A \cup V \cup H$, where '∪' stands for the union of two sets.

Now, we can form the *words* of our language:

1. Any atomic proposition is a word.
2. If f and g are words, then so too are $(f \wedge g), (f \vee g), (f \rightarrow g), (f \leftrightarrow g)$, and $\neg f$.
3. All words are made in this way. (No additional words exist.)

We call the words of this language *propositions*.

Convention 1.2 Usually, we omit the outermost parentheses of a formula, thus for example writing $a \wedge b$ instead of $(a \wedge b)$. Of course we cannot always do the same with the inner parentheses, take for example the logically inequivalent $(R \wedge S) \rightarrow RB$ and $R \wedge (S \rightarrow RB)$. What we can do, is *agree* upon a notation in which we're allowed to omit *some* of the parentheses. We do this by defining a *priority* for each connective:

- $\neg$ binds stronger than $\wedge$
- $\wedge$ binds stronger than $\vee$
- $\vee$ binds stronger than $\rightarrow$
- $\rightarrow$ binds stronger than $\leftrightarrow$

This mean that we must read the formula $\mathrm{In} \vee \mathrm{RB} \rightarrow \mathrm{Out} \leftrightarrow \neg\, \mathrm{S} \wedge \mathrm{R}$ as the formula $((\mathrm{In} \vee \mathrm{RB}) \rightarrow \mathrm{Out}) \leftrightarrow (\neg\, \mathrm{S} \wedge \mathrm{R})$.

Only using these priorities is not enough though: it only describes where the implicit parentheses are in the case of *different* connectives. When statements are built up by repeated use of the *same* connective, it is not clear yet where these parentheses should be read.

For example, should we parse $\mathrm{Out} \rightarrow \mathrm{R} \rightarrow \mathrm{W}$ as expressing $(\mathrm{Out} \rightarrow \mathrm{R}) \rightarrow \mathrm{W}$ or as expressing $\mathrm{Out} \rightarrow (\mathrm{R} \rightarrow \mathrm{W})$? This is formalized by defining the *associativity* of the operators.

Convention 1.3 The connectives $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$ are *right associative*. This means that, if $v \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, then we must read

$$A \;\; v \;\; B \;\; v \;\; C$$

as expressing

$$A \;\; v \;\; (B \;\; v \;\; C)$$

Note, though, that this is a choice that is sometimes made differently, outside of this course. So sometimes, in other readers and books on logic, an other choice might be made.

Remark 1.4 (This remark may only have meaning to you after reading Chapter 3, "Languages and automata"; so after reading that chapter, you might want to reread this remark.) The formal definition of our language, with the help of a *context-free grammar*, is as follows: Let $\Sigma = A \cup V \cup H$, that is to say $\Sigma = \{a, b, c, \ldots, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \neg, (,)\}$. Then the language is defined by the grammar

$$S \rightarrow a \mid b \mid c \mid \ldots \mid \neg S \mid (S \wedge S) \mid (S \vee S) \mid (S \rightarrow S) \mid (S \leftrightarrow S)$$

Obviously, in stead of the dots all other symbols of the alphabet A should be listed.

1.4 Meaning and truth tables

The sentence 'if a and b, then a' is true, whatever you substitute for a and b. So, we'd like to be able to say: the sentence '$a \wedge b \rightarrow a$' is true[2]. But we can't, because we haven't formally defined what that means yet. As of yet, '$a \wedge b \rightarrow a$' is only one of the words of our formal language. Which is why we'll now turn to defining the meaning of a word (or statement) of language, specifically speaking, when such a statement of logic would be *true*.

For the atomic propositions, we can think of any number of simple statements, such as '2=3,' or 'Jolly Jumper is a horse,' or 'it rains.' In classical logic, which is our focus in this course, we simply assume that these atomic propositions way be true, or false. We don't further concern ourselves with the specific truth or falsity of particular statements like 'on January the 1st of 2050 it'll rain in Nijmegen.'

The truth of atomic propositions will be defined solely by their interpretation in a *model*. For example, '2=3' is not true in the model of natural numbers. And 'Jolly Jumper is a horse' is true in the model that is a comic of Lucky Luke. And the sentence 'it rains' was not true in Nijmegen, on the 17th of September of 2002.

Now let's take a look at the composite sentence '$a \wedge b$.' We'd want this sentence to be true in a model, exactly in the case that both a and b are true in that model. If we then just enumerate the possible truth values of a and of b, we can define the truth of $a \wedge b$ in terms of these.

In Computing Science, we often simply write 1 for *true*, and 0 for *false*. Logical operations, then, are the elementary operations on bits.

So, we get the following truth table:

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

And what of the 'if..., then...' construction to combine statements into larger ones? Here, too, natural language is somewhat ambiguous at times. For example, what if A is false, but B is true. Is the sentence 'if A, then B' true? Examples are:

'if $1 + 1 = 3$, then $2 + 2 = 6$'
'if I jump off the Erasmus building, I'll turn into a bird'
'if I understand this stuff, my name is Alpje'

[2]Note: $a \wedge b \rightarrow a$ should be read as $(a \wedge b) \rightarrow a$

We'll put an end to all this vagueness by simply *agreeing* upon the truth tables for the connectives (and writing '$A \to B$' instead of 'if A, then B'), in this next definition.

Definition 1.5 The *truth tables* for the logical connectives are defined to be:

x	$\neg x$
0	1
1	0

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

x	y	$x \to y$
0	0	1
0	1	1
1	0	0
1	1	1

x	y	$x \leftrightarrow y$
0	0	1
0	1	0
1	0	0
1	1	1

Using our truth tables, we can determine the truth value of complex propositions from the truth values of the individual atomic propositions. We do this by writing out larger truth tables for these complex propositions.

Example 1.6 This is the truth table for the formula $a \vee b \to a$:

a	b	$a \vee b$	a	$a \vee b \to a$
0	0	0	0	1
0	1	1	0	0
1	0	1	1	1
1	1	1	1	1

Exercise 1.D Find the truth tables for:

(i) $a \vee \neg a$
(ii) $(a \to b) \to a$
(iii) $a \to (b \to a)$
(iv) $a \wedge b \to a$
(v) $a \wedge (b \to a)$
(vi) $\neg a \to \neg b$

1.5 Models and truth

In the introduction, we spoke of *models* and truth in models. In the truth tables, we've seen that the '*truth*' of a proposition is fully determined by the values that we assign to the atomic propositions. A model of propositional logic is therefore simply some assignment of values ($\{0, 1\}$) to the atomic propositions.

Definition 1.7 A *model* of propositional logic is an *assignment*, or *valuation*, of the atomic propositions: a function $v : A \to \{0, 1\}$.

To determine the truth value of a proposition f, we don't really need to know the value of all atomic propositions, but only of those of which f is comprised. Therefore, we'll actually just equate a model to a *finite assignment* of values.

Example 1.8 In the model v that has $v(a) = 0$ and $v(b) = 1$, the proposition $a \vee b \to a$ has the value 0.

Convention 1.9 If v is a model, we'll also simply write $v(f)$ for the value that f is determined to have under that model. We say that f is *true in a model* v in the case that $v(f) = 1$.

So $a \vee b \to a$ is not true, or false, in v when $v(a) = 0$ and $v(b) = 1$. However, $a \to (b \to a)$ is true is such a model.

Definition 1.10 If a proposition f is true in every conceivable model (which is to say that in its truth table, there are only 1's in its column), then we call that proposition *logically true*, *logically valid* or just *valid*. The notation for this is: $\models f$. A logically true statement is also called a *tautology*.

If a proposition f is *not* logically true, that can be denoted by writing $\not\models f$.

Exercise 1.E Which of the following propositions are logically true?

(i) $a \vee \neg a$	(v) $a \to (b \to a)$
(ii) $a \to (a \to a)$	(vi) $a \wedge b \to a$
(iii) $a \to a$	(vii) $a \vee (b \to a)$
(iv) $(a \to b) \to a$	(viii) $a \vee b \to a$

Exercise 1.F Let f and g be *arbitrary* propositions. Find out whether the following statements hold. Explain your answers.

(i) If $\models f$ and $\models g$, then $\models f \wedge g$.
(ii) If not $\models f$, then $\models \neg f$.
(iii) If $\models f$ or $\models g$, then $\models f \vee g$.
(iv) If (if $\models f$, then $\models g$), then $\models f \to g$.
(v) If $\models \neg f$, then not $\models f$.
(vi) If $\models f \vee g$, then $\models f$ or $\models g$.
(vii) If $\models f \to g$, then (if $\models f$, then $\models g$).
(viii) If $\models f \leftrightarrow g$, then ($\models f$ if and only if $\models g$).
(ix) If ($\models f$ if and only if $\models g$), then $\models f \leftrightarrow g$.

1.6 Logical equivalence

Definition 1.11 Two propositions f and g are said to be *logically equivalent* in the case that f is true in a model if and only if g is true in that same model.
To formulate this more precisely: f and g are logically equivalent if for every model v it holds that $v(f) = 1$ if and only if $v(g) = 1$. This boils down to saying that f and g have the same truth tables.
To denote that f and g are logically equivalent, we write $f \equiv g$.

Often, a proposition can be replaced by a simpler, equivalent proposition. For example, $a \wedge a$ is logically equivalent to a. So, $a \wedge a \equiv a$.

Exercise 1.G For each of the following couples of propositions, show that they are logically equivalent to each other.

(i) $(a \wedge b) \wedge c$ and $a \wedge (b \wedge c)$ (ii) $(a \vee b) \vee c$ and $a \vee (b \vee c)$

In this case, apparently, the placement of parentheses doesn't really matter. For this reason, in some cases we simply omit such superfluous parentheses. In convention 1.3 we agreed that all binary connectives would be right associative. But here, we see that this choice, at least for $\wedge$ and $\vee$, is arbitrary: had we agreed that $\wedge$ and $\vee$ associate to the left, then that would have had no consequence for the truth value of the composite propositions.[3] In exercise 1.D we've seen that for the connective $\rightarrow$, it actually does matter in which direction it associates!

Remark 1.12 The propositions $a \wedge b$ and $b \wedge a$ are mathematically speaking logically equivalent. In English though, the sentence 'they married and had a baby' often means something entirely different compared to 'they had a baby and married.'

Here are a number of logically equivalences that demonstrate the distributivity of the operators $\neg$, $\wedge$, and $\vee$ over parentheses. The first two are called the *De Morgan* laws.

1. $\neg(f \wedge g) \equiv \neg f \vee \neg g$.
2. $\neg(f \vee g) \equiv \neg f \wedge \neg g$.
3. $f \wedge (g \vee h) \equiv f \wedge g \vee f \wedge h$.[4]
4. $f \vee g \wedge h \equiv (f \vee g) \wedge (f \vee h)$.

Exercise 1.H Let f and g be propositions. Is the following statement true? $f \equiv g$ if and only if $\models f \leftrightarrow g$.

1.7 Logical consequence

In English, the statement that 'the sun is shining' follows logically from the statement that 'it is raining and the sun is shining.' Now, we want to define this same logical consequence for our formal language: that a is a logical consequence of $a \wedge b$.

Definition 1.13 A proposition g is a *logical consequence*, also sometimes called a *logical entailment*, of the proposition f, if g is true in every model for which f is true. Said differently: a proposition g is a logical consequence of f if, in every place in the truth table of f in which there is a 1, the truth table of g also has a 1. Notation: $f \models g$.

If g is *not* a logical consequence of f, that can be denoted by $f \not\models g$.

Exercise 1.I Are the following statements true?

(i) $a \wedge b \models a$ (iii) $a \models a \vee b$
(ii) $a \vee b \models a$ (iv) $a \wedge \neg a \models b$

Theorem 1.14 *Let f and g be propositions. Then the following holds:*

[3] The reason that we chose for the right associativity of $\wedge$ and $\vee$ is to be consistent with the proof system Coq that will be used in the course "Beweren en Bewijzen".

[4] Note: by the conventions we specified for parentheses, this should be read as $(f \wedge g) \vee (f \wedge h)$.

$$\models f \to g \textit{ if and only if } f \models g.$$

You should now be able to find a proof for this proposition. [*Hint:* Take a look at exercises 1.F and 1.I.]

Remark 1.15 Note that the symbols $\models$ and $\equiv$ were not included in the definition of the propositions of our formal language. These symbols are not a part of the language, but are merely used to speak mathematically *about* the language. Specifically, this means that constructions as '$(f \equiv g) \to (\models g)$' and '$\neg \models f$' are simply meaningless, and we should try to avoid writing down such invalid constructions.

Remark 1.16 For more on logic, you can have a look at books like [3] and [2].

1.8 Important concepts

2 Predicate logic

"A woman is happy if she loves someone,
and a man is happy if he is loved by someone."

Instead of discussing the truth of this sentence, we are going to find out how to write it in a formal way. Hopefully, this will then help us analyze the truth of such sentences.

To jump right in, we'll give a formal account of the sentence:

Dictionary

W	set of (all) women
M	set of (all) men
$L(x, y)$	x loves y
$H(x)$	x is happy

Formal translation

$$\begin{gathered} \forall w \in W \left[\big(\exists x{\in}(M \cup W)\ L(w, x)\big) \to H(w)\right] \\ \wedge \\ \forall m \in M \left[\big(\exists x{\in}(M \cup W)\ L(x, m)\big) \to H(m)\right] \end{gathered} \tag{1}$$

The '$\forall$' symbol stands for "for all", and the '$\exists$' symbol stands for "there exists". The '$\in$' symbol denotes the membership of an element in a set, and '$\cup$', which we've encountered before, stands for the union of two sets. If we translate the formal sentence back to English, in a very literal way, we get:

> For every woman it holds that, if there is a person that she loves, she is happy, and for every man it holds that, if there is person that loves him, he is happy.

You should now check whether this is indeed the same as the sentence that we started with, and whether you can see how this is indeed represented in the formal translation above. Note our use of the square parentheses [_] (also called brackets.) This is only for readability though, so that you can easily see which parenthesis belongs to which.
Because it enables us to translate a sentence, we'll also call our dictionary an *interpretation*.

2.1 Predicates, relations and constants

Previously, we saw how propositional logic allowed us to translate the English sentence

If Sharon is happy, Koos is not.

as:

Dictionary

SH	Sharon is happy
KH	Koos is happy

Translation

$$SH \to \neg KH$$

But suppose we add more people to our statements, like Joris, and Maud. We'll easily end up with an inconveniently lengthy dictionary soon, because we have to add an atomic proposition for every new person (JH, MH, ...).

This is why we now take a better look at the form of the statement that

Sharon is happy

and find out that it is of the shape of a *predicate* $H(\ )$ applied to a *subject* s (Sharon). Let's write that as $H(s)$. The immediate benefit is that we can now also write the very similar

$$H(k),\ H(j), \text{ and } H(m)$$

for the subjects k, j, and m. Instead of subjects, we'll often speak of *constants*.

Also for our list of subjects, we'll use our dictionary to formally denote which subject belongs to which name. Moreover, we'll indicate to which *domain* each of our subjects belongs.

Dictionary

s	Sharon ∈ "women"
k	Koos ∈ "men"
j	Joris ∈ "men"
m	Maud ∈ "women"

Maybe the real benefit of our new system is not yet entirely clear. (Now that we have one predicate H added to four subjects s, k, j, m, making a total of five symbols, which is more than the four we started out with: SH, KH, JH, MH...) But suppose now that our subjects would gain a number of other qualities, such as:

Dictionary

$T(x)$	x is tall
$B(x)$	x is beautiful
$N(x)$	x is nice
$I(x)$	x is intelligent

Moreover, besides our predicates (qualities), we also have (binary) relations, like the one we used above:

$$L(x, y): \ x \text{ loves } y.$$

Now we can take the following sentence

Sharon is an intelligent beautiful woman;
and there is a nice tall guy who loves this character.

and formalize it with the formula

$$\begin{aligned} I(s) \quad \wedge \quad & B(s) \\ \wedge \quad & \exists x \in M \left[N(x) \wedge T(x) \wedge \exists y \in W \ [L(x,y) \wedge B(y) \wedge I(y)] \right]. \end{aligned}$$

...or did we mean to say,

$$\begin{aligned} I(s) \quad \wedge \quad & B(s) \\ \wedge \quad & \exists x \in M \ [N(x) \wedge T(x) \wedge L(x,s)]? \end{aligned}$$

Notice that we take these two statements in the sentence above, and join them together to a single formula with the '$\wedge$' symbol. And that, although the formula doesn't state that Sharon is a woman, that it doesn't need to, because we already defined that $s \in$ "women" in the dictionary. (So we don't need to add something like for example "$s \in W$" in the formula.)

The difference between the first and the second formal translation lies in what what referred to as "this character." (This might refer to "Sharon," or it might refer to "intelligent and beautiful woman.") It is not logic that decides such questions, but logic does make it explicit that this choice has to be made. Said differently: it makes explicit the ambiguities that often occur in the English (or any other natural) language.

Exercise 2.A Give two possible translations for the following sentence.

Sharon loves Maud; a nice man loves this intelligent character.

Now let's translate the other way around, and decode a formula.

Exercise 2.B Translate the following sentences to English.

(i) $\exists x \in M \left[T(x) \wedge \exists w \in W \; [B(w) \wedge I(w) \wedge L(x,w)]\right]$

(ii) $\exists x \in M \left[T(x) \wedge \exists w \in W \; [B(w) \wedge \neg I(w) \wedge L(x,w)] \wedge \exists w' \in W[I(w') \wedge L(w',x)]\right]$

2.2 The language of predicate logic

We'll now define formally what the language of predicate logic looks like.

Definition 2.1 The language of predicate logic is built up from the following ingredients:

1. *Variables*, usually written $x, y, z, x_0, x_1, \ldots$, or sometimes $v, w, \ldots$
2. *Individual constants*, also called 'names', usually $a, b, c, \ldots$,
3. *Domains*, such as $M, V, E, \ldots$,
4. *Relation symbols*, each with a fixed "arity", that we sometimes explicitly annotate them with, as in P^1, R^2, T^3, ... So this means that P^1 takes a single argument, R^2 takes two arguments, etc.
5. The *atoms*, or *atomic formulas*, are $P^1(t)$, $R^2(t_1, t_2)$, $T^3(t_1, t_2, t_3)$ etc., in which t, t_1, t_2, t_3, etc. are either variables or individual constants.
6. The *formulas* are built up as follows:
   - Every atomic formula is a formula.
   - If f and g are formulas, then so are $(f \wedge g), (f \vee g), (f \to g), (f \leftrightarrow g)$, and $\neg f$.
   - If f is a formula, and x is a variable, and D is a domain, then $(\forall x \in D \; f)$ and $(\exists x \in D \; f)$ are also formulas, made with the *quantifiers* '$\forall$' and '$\exists$'.
   - All formulas are made in this way. (No others exist.)

Convention 2.2 Just as in propositional logic, we usually omit the outermost parentheses. And to be able to omit excessive parentheses within formulas, we expand upon our previous convention 1.2 for propositional logic, adding the following:

- $\forall$ and $\exists$ bind stronger than all other connectives.

Somewhat opposing what we said earlier, in the case of $\forall$ and $\exists$, we add brackets for readability. So instead of $(\forall x \in D\ f)$, as in the definition, we write $\forall x \in D\ [f]$. And if we have a consecutive series of the same quantifiers over the same domain, we may group these quantifications: $\forall x \in D\ [\forall y \in D\ [f]]$ may be abbreviated to $\forall x, y \in D\ [f]$. Note that you are not allowed to combine existential and universal quantifications in such an abbreviation.

If you take a look back at our first formula (1) this chapter, you can see that it actually quite inconsistently uses parentheses and brackets. If we write it out as per the official definition, it would read:

$$\Big(\Big(\forall w \in W\ \big((\exists x{\in}(M \cup W)\ L(v,x)) \to H(v)\big)\Big) \land \Big(\forall m \in M\ \big((\exists x{\in}(M \cup W)\ L(x,m)) \to H(m)\big)\Big)\Big)$$

You can see why we prefer to somewhat liberally use parentheses and brackets for readability. Let us take a look at the meaning of all the parentheses in this official version of the formula:

$$\underbrace{\left(\underbrace{\left(\forall w \in W \underbrace{\left(\underbrace{\left(\exists x{\in} \underbrace{(M \cup W)}_{1}\ L\underbrace{(v,x)}_{2}\right)}_{3} \to H\underbrace{(v)}_{2}\right)}_{4}\right)}_{5} \land \underbrace{\left(\forall m \in M \underbrace{\left(\underbrace{\left(\exists x{\in} \underbrace{(M\cup W)}_{1}\ L\underbrace{(x,m)}_{2}\right)}_{3} \to H\underbrace{(m)}_{2}\right)}_{4}\right)}_{5}\right)}_{6}$$

1. These parentheses are used for the readability of the domain, which is the union of the sets M and W in this case. (The parentheses are not required by definition 2.1.)
2. These parentheses are needed because $L(v,x)$, $H(v)$, $L(x,m)$, and $H(m)$ are atomic formulas, so the parentheses are required.
3. These parentheses are needed because the $\exists$ quantifier needs them.
4. These parentheses are needed because the $\to$ requires them.
5. These parentheses are needed because the $\forall$ quantifier needs them.
6. These parentheses are needed because the conjunction with $\land$ requires them.

If we use square brackets as noted above, consistently writing $\forall x \in D\ [f]$ instead of $(\forall x \in D\ f)$, and furthermore omit all unnecessary parentheses according to our convention, we end up with the formula:

$$\forall w \in W\ \Big[\exists x{\in}(M \cup W)\ [L(v,x)] \to H(v)\Big] \land \forall m \in M\ \Big[\exists x{\in}(M \cup W)\ [L(x,m)] \to H(m)\Big].$$

Remark 2.3 Grammar of predicate logic.

Just as with propositional logic we can define the language of predicate logic as a formal language, with a grammar. This is done as follows. (You should reread this after having met grammars in Chapter 3, Languages and automata.)

Individual	:=	Variable | Name
Variable	:=	$x, y, z, x_1, y_1, z_1, \ldots$
Name	:=	$a, b, c, d, e, \ldots$
Domain	:=	$D, E, \ldots$
Atom	:=	P^1(Individual) | R^2(Individual,Individual)
		| T^3(Individual,Individual,Individual) | …
Formula	:=	Atom
		| ¬ Formula
		| (Formula → Formula) | (Formula ∧ Formula)
		| (Formula ∨ Formula) | (Formula ↔ Formula)
		| (∀ Variable ∈ Domain Formula)
		| (∃ Variable ∈ Domain Formula)

Exercise 2.C Formalize the following sentence.

> *Sharon is beautiful; there is a guy who feels good about himself whom she loves.*

Here, we'll treat feeling good about oneself as being in love with oneself.

Exercise 2.D Formalize the following sentences:

(i) *For every x and y we have: x loves y only if x feels good about him- or herself.*

(ii) *For every x and y we have: x loves y if he or she feels good about him- or herself.*

(iii) *For every x and y we have: x loves y exactly in the case that y feels good about him- or herself.*

(iv) *There is somebody who loves everyone.*

2.3 Truth and deduction

Take a look at the following two formulas

$$\begin{aligned} F_1 &= \forall x{\in}D \exists y{\in}D\ [K(x,y)] \\ F_2 &= \exists x{\in}D \forall y{\in}D\ [K(x,y)] \end{aligned}$$

(Note how we cut back on parentheses.) When is a formula *true*? Truth is relative and depends upon an *interpretation* and a model.

Definition 2.4 A *model* is that piece of the 'real' world in which formulas gain meaning by means of an *interpretation*.

Because we haven't yet explained what an interpretation is, this definition isn't really conclusive. But before we introduce the concept of an interpretation, let us first explain with a few examples what we mean with 'a piece of the real world.' Central here is the choice of the domains that we choose, or how we restrict ourselves to a certain part of them, and which predicates and relations are known within those domains.

1. Model M_1

Domain(s)	all students in the lecture hall
Predicate(s)	is female
	is more than 20 years old
Relation(s)	has a student number lower than
	is not older than
	is sitting next to

2. Model M_2

Domain(s)	people
	animals
Predicate(s)	is female
	is human
	is canine
Relation(s)	owns
	is older than

In the case of the students in the lecture hall, we can determine which are female and not. And if Sharon and Maud would happen to be present, it could be determined whether they're sitting next to each other. In the other case, there are biologists who can determine for any animal whether it is a dog or not, and furthermore it is usually easy to establish whether someone is the owner of a certain animal. In which way this is done, should actually still be formally agreed upon, if we wish to have an exact model.

The important aspect is that, if we wish to use a certain part of the world as a domain for predicate logic, we should have a conclusive and consistent way of determining the truth of statements about predicates over subjects and relations between subjects of the domain.

Now we can define the notion of an interpretation.

Definition 2.5 An *interpretation* is given by a dictionary, which states:

1. Which sets are referred to by the domain symbols,
2. Which subjects are referred to by the names (and to which domain sets these belong),
3. Which predicates and relations are referred to by the predicate and relation symbols.

In particular, then, the interpretation establishes a clear connection between the symbols in formulas and the model that we're looking at. In the literature, the interpretation is therefore often done via an interpretation function. See for example [3].

Definition 2.6 *A formula f is called true in a model under a given interpretation if the translation given by the interpretation is actually true in the model.*

Example 2.7 In the running example of Sharon, Koos, Joris, and Maud, the interpretation has been given in the dictionary we introduced: we know exactly what is meant with the formulas $L(s, k)$ and $\exists x \in M\ L(x, s)$. The model is the factual situation involving these people, in which we can indeed determine whether $L(s, k)$ happens to be true (whether Sharon loves Koos), and whether $\exists x \in M\ L(x, s)$ happens to be true (whether there is some guy who loves Sharon).

Whether the formulas F_1 and F_2 are true within M_1, can thus only be determined if we also define an interpretation which gives meaning to the symbols. Here are three different possible interpretations:

1. Interpretation I_1

D	all students in the lecture hall
$K(x, y)$	x has a student number that is lower than that of y

2. Interpretation I_2

D	all students in the lecture hall
$K(x, y)$	x is not older than y

3. Interpretation I_3

D	all students in the lecture hall
$K(x, y)$	x is sitting next to y

We can assess the truth of the formulas F_1 resp. F_2, in these models and under the given interpretations, by checking whether the propositions hold that these formulas express, within the models, and under the interpretations.

Exercise 2.E (i) Verify that F_2 does not hold in M_1 under the interpretation I_1. But does F_1 hold?

(ii) Verify that F_1 holds in M_1 under the interpretation I_2. Does F_2 hold as well?

(iii) Check whether F_1 in M_1 is true under the interpretation of I_3, by looking around in class. And check whether F_2 is true or not, as well.

Because model M_1 only mentions a single domain, it would seem as if every interpretation would have to speak of the same domain. But this is not the case, as we can see in the following interpretations for model M_2:

1. Interpretation I_4

D	people
$K(x, y)$	x isn't older than y

2. Interpretation I_5

D	people as well as animals
$K(x, y)$	x owns y

Note that in the case of I_5, that if x is an animal and y is a person, the statement $K(x, y)$ is automatically false, because animals don't own people.

3. Interpretation I_6

D	animals
$K(x, y)$	y is older than x

Let's now take a look at the formulas G_1 and G_2, and two models in which we'll interpret them. (Note their difference to F_1 and F_2.)

$$G_1 \;=\; \forall x{\in}D \exists y{\in}D\ [K(x, y)]$$
$$G_2 \;=\; \forall x{\in}D \exists y{\in}D\ [K(y, x)]$$

Example 2.8 We define the models M_3

Domain(s)	Natural numbers, $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$
Relation(s)	smaller than ($<$)

and M_4

Domain(s)	Rational numbers (fractions), $\mathbb{Q}$, (for example $-\frac{1}{2}, 3(=\frac{3}{1}), 0, \ldots$)
Relation(s)	smaller than ($<$)

And two self-evident accompanying interpretations: Interpretation I_7:

D	$\mathbb{N}$
$K(x, y)$	$x < y$

en interpretation I_8:

D	$\mathbb{Q}$
$K(x, y)$	$x < y$

In model M_3 under interpretation I_7 the formula G_1 is true. Indeed, for every number $x \in \mathbb{N}$ we can easily find a number $y \in \mathbb{N}$, for example $y = x + 1$, such that $x < y$. In model M_4 under interpretation I_8 the formula G_1 is true as well. Again, we can take $y = x + 1$ and $x < y$ holds. Whenever we have such a method in which we can state precisely how to obtain a y for any x, we speak of having an *algorithm* or *recipe*.

Convention 2.9 If a formula f is true in a model M under the interpretation I, we denote this by

$$(M, I) \models f$$

Writing it this way, we've seen that:

$$(M_3, I_7) \models G_1$$
$$(M_4, I_8) \models G_1$$

If we hadn't given the models M_3 and M_4 a name, we could also have simply written this as follows:

$$((\mathbb{N}, <), I_7) \models G_1$$
$$((\mathbb{Q}, <), I_8) \models G_1$$

Because the (important part of the) model is often already expressed in the given interpretation, we will often omit an exact definition of the model.

Exercise 2.F Verify that G_2 is indeed true in model M_4 under the interpretation I_8, but not in model M_3 under the interpretation I_7. Stated differently: verify that $((\mathbb{Q}, <), I_8) \models G_2$, and $((\mathbb{N}, <), I_7) \not\models G_2$.

Exercise 2.G Define the interpretation I_9 as:

D	$\mathbb{N}$
$K(x, y)$	$x = 2 \cdot y$

Are the formulas G_1 and/or G_2 true under this interpretation?

Exercise 2.H Define the interpretation I_{10} as:

D	$\mathbb{Q}$
$K(x, y)$	$x = 2 \cdot y$

Are the formulas G_1 and/or G_2 true under this interpretation?

Exercise 2.I We take as model the countries of Europe, and the following interpretation I_{11}:

E	the set of countries of Europe
n	The Netherlands
g	Germany
i	Ireland
$B(x, y)$	x borders y
$T(x, y, z)$	x, y, and z share a tripoint (where the borders of all three countries meet)

(i) Formalize the sentence "The Netherlands and Germany share a tripoint."
(ii) Which of the following formulas are true in this model and under this interpretation?
  (1) $G_3 := \forall x \in E \exists y \in E[B(x, y)]$
  (2) $G_4 := \forall x, y \in E[(\exists z \in E\ T(x, y, z)) \to B(x, y)]$
  (3) $G_5 := \forall x \in E[B(i, x) \to \exists y \in E[T(i, x, y)]]$.

Exercise 2.J Find a model M_5 and an interpretation I_{12} such that this formula holds:

$$(M_5, I_{12}) \models \forall x \in D \exists y \in E[R(x, y) \wedge \neg R(y, x) \wedge \neg R(y, y)]$$

In convention 2.9 we've seen that we write $(M, I) \models f$ in the case that the formula f of predicate logic (with or without equality, which will be discussed in Section 2.4) is true in a model M under the translation given by an interpretation (or, dictionary) I. Now, we'll expand a bit upon this definition.

Definition 2.10 A formula f of predicate logic f is said to be *logically true*, or *logically valid*, which we then denote $\models f$, when for *any* model and *any* interpretation, the translation holds in that model.

We often omit the *logically* part, simply writing that f is *true* or f is *valid* instead of writing that f is *logically true* or *logically valid*.

Example 2.11 Consider the following statements:
(i) $\models \forall x{\in}D\ [P(x) \to P(x)]$.
(ii) $\models (\exists x{\in}D \forall y{\in}D\ [P(x, y)]) \to (\forall y{\in}D \exists x{\in}D\ [P(x, y)])$.

(iii) $\not\models (\forall y{\in}D \exists x{\in}D\ [P(x,y)]) \to (\exists x{\in}D \forall y{\in}D\ [P(x,y)])$.

The formula within statement (iii) is not true, which can be seen by taking the interpretation $D := \mathbb{N}$ and $P(x,y) := x{>}y$. Under this interpretation, $\forall y{\in}D \exists x{\in}D\ [P(x,y)]$ is true, because for every $y \in \mathbb{N}$ we can indeed find a larger $x \in \mathbb{N}$, but $\exists x{\in}D \forall y{\in}D\ [P(x,y)]$ is not true, because there is no biggest number $x \in \mathbb{N}$. So the implication is false.

Definition 2.12 Suppose f and g are two formulas of predicate logic. We say that g follows from f, denoted as $f \models g$, when $\models f \to g$. Which means that in every situation in which f is true, g is true as well.

Statement (ii) above tells us that $\forall y{\in}D \exists x{\in}D\ [P(x,y)]$ follows from $\exists x{\in}D \forall y{\in}D\ [P(x,y)]$.

2.4 The language of predicate logic with equality

One might want to formalize the sentence:

Sharon is intelligent; there is a man who pays attention to nobody else.

To be able to formalize this, we require an *equality relation*. Then, we can write:

$$I(s) \wedge \exists x{\in}M\ [A(x,s) \wedge \forall w{\in}W\ [A(x,w) \to w = s]] \tag{2}$$

For this to be regarded a correct formula of predicate logic, we have to add the equality sign to the formal definition of the language of predicate logic.

Definition 2.13 The language of *predicate logic with equality* is defined by adding to the standard predicate logic the binary relation "=". The interpretation of this relation is always taken to be "is equal to".

We then also have to add the following rule to the definition 2.1 which states which formulas exist:

- If x and y are variables, and a and b are constants, then $(x = y)$, $(x = a)$, $(a = x)$, and $(a = b)$ are formulas as well.

Remark 2.14 The language of predicate logic with equality is very well suited to make statements about the number of objects having certain properties, such as there being exactly one such object, or at least two, or at most three, different, etc.

Exercise 2.K Consider the interpretation I_{13}:

H	domain of all human beings
$F(x)$	x is female
$P(x,y)$	x is parent of y
$M(x,y)$	x is married to y

Formalize the following sentences into formulas of predicate logic with equality:

(i) *Everyone has exactly one mother.*

(ii) *Everybody has exactly two grandma's.*

(iii) *Every married man has exactly one spouse.*

Exercise 2.L Use the interpretation I_{13} of exercise 2.K to formalize the following properties.

(i) $C(x, y)$: x and y have had a child together.

(ii) $B(x, y)$: x is a brother of y (take care: refer also to the next item).

(iii) $S(x, y)$: x is a step-sister to y.

Translate the following formulas back to English.

(iv) $\exists x{\in}H \forall y{\in}H\ P(x, y)$. And is this true?

(v) $\forall z_1{\in}H \forall z_2{\in}H[(\exists x{\in}H \exists y_1{\in}H \exists y_2{\in}H\ P(x, y_1) \wedge P(y_1, z_1) \wedge P(x, y_2) \wedge P(y_2, z_2)) \to \neg(\exists w{\in}H(P(z_1, w) \wedge P(z_2, w)))]$. And is this true?

Exercise 2.M Given the interpretation I_{14}:

D	$\mathbb{N}$
$A(x, y, z)$	$x + y = z$
$M(x, y, z)$	$x \cdot y = z$

Formalize the following:

(i) $x < y$.

(ii) $x \mid y$ (x divides y).

(iii) x is a prime number.

2.5 Important concepts

3 Languages and automata

Generating and describing languages is an important application of computers. In principle, a computer can only deal well with languages that are given an exact "formal" definition, for example, programming languages. Though one can of course learn a computer to recognize a natural language, as well, provided you give precise enough rules for it. But for now, we will deal with formally defined languages. We take a *language* to be a set of *words*. And a *word* is simply a sequence of symbols taken from a specified *alphabet*.

With these informal definitions we can already ask ourselves a number of interesting questions about languages and their words, like:

- Does L contain the word w?
- Are the languages L and L' the same?

Many problems in computer science that seem on first sight unrelated to languages, can be reformulated, or, translated, into corresponding questions about (formal) languages.

And from the topic of languages, it is just a small step to the topic of *automata*. Automata are yet again formal mathematical objects, and the key question that we often ask about them, is which language they accept (or, define). This question relates the two, and is the reason why we treat them together in this chapter.

3.1 Alphabet, word, language

Definition 3.2 1. An *alphabet* is a finite set Σ of *symbols*.[5]

2. A *word* over Σ is a finite number of symbols, strung together to a sequence.

3. λ is the word that is made up of 0 symbols, that is, no symbols at all, which is called the *empty word*.

4. Σ^* is the set of all words over Σ.

5. A *language* L over Σ is a subset of Σ^*.

Example 3.3 (i) $\Sigma = \{a, b\}$ is an alphabet.
(ii) $abba$ is in Σ^* (notation: $abba \in \Sigma^*$).
(iii) $abracadabra \notin \Sigma^*$.
(iv) $abracadabra \in \Sigma_0^*$, with $\Sigma_0 = \{a, b, c, d, r\}$.
(v) The *empty word* λ is in Σ^*, whatever Σ may be.

We can describe languages in several ways. A couple of important ways, which we will treat in this course, are *regular expressions* (see Section 3.2), *grammars* (see Section 3.3), and *finite automata* (see Section 3.5). But remember that of course, languages are also simply the sets of words they contain.

Definition 3.4 We will introduce some useful notation.

[5] Although the alphabet may of course contain symbols such as ',' and '*', we will often call all symbols 'letters' for simplicity.

1. We write a^n for $a \dots a$ where we have n times a in a sequence. More precisely put: $a^0 = \lambda$, and $a^{n+1} = a^n a$.

2. The *length* of a word w is denoted $|w|$.

3. If w is a word, then w^R is the *reversed* version of the same word, so for example $(abaabb)^R = bbaaba$.

Example 3.5 (i) $L_1 := \{w \in \{a,b\}^* \mid w \text{ contains an even number of } a\text{'s}\}$.
(ii) $L_2 := \{a^n b^n \mid n \in \mathbb{N}\}$.
(iii) $L_3 := \{wcv \in \{a,b,c\}^* \mid w$ does not contain any b, v does not contain any a, and $|w| = |v|\}$.
(iv) $L_4 := \{w \in \{a,b,c\}^* \mid w = w^R\}$.
Try, for every two of the above languages, to think of a word that is in the one, but not in the other.

If we have two languages L and L', we can define new languages with the usual set operations. (Though the complement might be new to you.)

Definition 3.6 Let L and L' be languages over the alphabet Σ.

1. $\overline{L}$ is the *complement* of L: the language comprised of all words w that are *not* in L. (So: the $w \in \Sigma^*$ such that $w \notin L$.)

2. $L \cap L'$ is the *intersection* of L and L': the language comprised of all words w that are both in L as well as in L'.

3. $L \cup L'$ is the *union* of L and L': the language comprised of all words w that are either in L or in L' (or in both).

Exercise 3.A In example 3.5 we have seen some examples of language descriptions. Now try describing these languages yourself using formal set notation:
(i) Describe $L_1 \cap L_2$.
(ii) Describe $L_2 \cap L_4$.
(iii) Describe $L_3 \cap L_4$.

Besides these general operations on sets there are some operations that only work for languages, which we will now define:

Definition 3.7 Let L and L' be two languages of the alphabet Σ.

1. LL' is the *concatenation* of L and L': the language that contains all words of the shape wv, where $w \in L$ and $v \in L'$.

2. L^R is the language that contains all *reversed* words of L, that is, all w^R for $w \in L$.

3. L^* is the language comprised of all finite concatenations of words taken from L; so that it contains all words of the shape $w_1 w_2 \dots w_k$, where $k \geq 0$ and $w_1, w_2, \dots, w_k \in L$.

The language L^* is called the *Kleene closure* of L. The language L^* contains all concatenations of 0 or more words of L, so it is always the case that $\lambda \in L^*$. Furthermore, we always have $L \subset L^*$ for any language L. Try to find out yourself what the language L^* is for $L = \emptyset$ and for $L = \{\lambda\}$.

Exercise 3.B Use the definitions of example 3.5.

(i) Prove that $L_1 = L_1^*$

(ii) Does $L_2L_2 = L_2$ hold? Give a proof, or else a counterexample.

(iii) Does $\overline{L_1} = \overline{L_1}^*$ hold? Give a proof, or else a counterexample.

(iv) For which languages of example 3.5 do we have $L = L^R$? (You need only answer, a proof is not necessary.)

3.2 Regular languages

A very popular way of describing languages is by the means of *regular expressions*. A language that can be described by such an expression, is called a *regular language*. In computer science, regular languages are seen as (relatively) simple languages: if L is such a language, it is not hard to build a little program that checks whether a given word w is contained in L. Such a program is then called a *parser*, and answering the question whether "$w \in L$" is called *parsing*. Parsers for regular languages are easy to make, and there are many programs that will even generate such parsers for you (when given as an input, a regular expression defining that language). The parsers generated by these "parser generators" are especially *efficient*: they decide very quickly whether w is or is not in L. This course will not further deal with the concept of parsing, but we will delve further into the notions of a regular language and a regular expression.

Definition 3.8 Let Σ be an alphabet. The *regular expressions* over Σ are defined as follows:

1. $\emptyset$ is a regular expression,
2. λ is a regular expression,
3. x is a regular expression, given any $x \in \Sigma$,
4. if r_1 and r_2 are regular expressions, then so too are $(r_1 \cup r_2)$ and $(r_1\, r_2)$,
5. if r is a regular expression, then r^* is a regular expression too.

Example 3.9 Some examples of regular expressions are: aba, $ab^*(a \cup \lambda)$, $(a \cup b)^*$, $(a \cup \emptyset)b^*$, and $(ab^* \cup b^*a)^*$.

Now let's formalize the relation between regular expressions and (regular) languages.

Definition 3.10 For every regular expression r, we define the *language of* r, denoted $\mathcal{L}(r)$, as follows:

1. $\mathcal{L}(\emptyset) := \emptyset$,
2. $\mathcal{L}(\lambda) := \{\lambda\}$,
3. $\mathcal{L}(x) := \{x\}$ for every $x \in \Sigma$,
4. $\mathcal{L}(r_1 \cup r_2) := \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$,
5. $\mathcal{L}(r_1\, r_2) := \mathcal{L}(r_1)\mathcal{L}(r_2)$,
6. $\mathcal{L}(r^*) := \mathcal{L}(r)^*$.

To exemplify this definition, take a look at the languages of the regular expressions from example 3.9 above:

- $\mathcal{L}(aba) = \{aba\}$,
- $\mathcal{L}(ab^*(a \cup \lambda)) = \{a\}\{b\}^*\{a, \lambda\}$, so that is the language of words that start with an a, then an arbitrary amount of b's, and ending either with an a, or just ending at that,
- $\mathcal{L}((a \cup b)^*) = \{a, b\}^*$, or, all possible words over the alphabet $\{a, b\}$,
- $\mathcal{L}((a \cup \emptyset)b^*) = \{a\}\{b\}^*$, which is the same language as the one of the regular expression of ab^*,
- $\mathcal{L}((ab^* \cup b^*a)^*) = (\{a\}\{b\}^* \cup \{b\}^*\{a\})^*$. This language is somewhat less easily described in natural language.

Sometimes regular expressions are also written using the + operator: the expression a^+ then stands for "1 or more times a." Technically speaking, this operator isn't needed, because instead of a^+, one can also simply write aa^*. (Check this.)

Exercise 3.C (i) Demonstrate that the operator ?, for which $a^?$ stands for either 0 or 1 times a, doesn't have to be added to the regular expressions, as it can be defined using the existing operators.
(ii) What is $\mathcal{L}(\emptyset ab^*)$?
(iii) We define the language $L_5 := \{w \in \{a, b\}^* \mid w \text{ contains at least one } a\}$. Give a regular expression that describes this language.
(iv) Give a regular expression that describes the language L_1 from example 3.5.
(v) Show that $\mathcal{L}(ab(ab)^*) = \mathcal{L}(a(ba)^*b)$.

We see that the regular expression $\emptyset$ isn't very useful. for any expression r, we can simply replace $r \cup \emptyset$ with r, so that the only reason to use $\emptyset$ is when we want to describe the empty language $\mathcal{L}(\emptyset)$. Apart from this use, you won't see the expression $\emptyset$ any more.[6]

Definition 3.11 Let Σ be an alphabet. We call a language L over Σ *regular* if some regular expression exists that describes it. More precisely put: a language L over Σ is regular if and only if there is a regular expression r for which $L = \mathcal{L}(r)$.

The language L_1 from example 3.5 is regular, as we have seen in exercise 3.C. However, the languages L_2, L_3, and L_4 from example 3.5 are not. We won't be able to prove this with the material of this course, though.

Exercise 3.D Show that the following languages are regular.
(i) $L_6 := \{w \in \{a, b\}^* \mid \text{every } a \text{ in } w \text{ is directly followed by a } b\}$,
(ii) $L_7 :=$ the language of all well-formed integer expressions. These expressions are made up of the symbols $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, but they never start with a 0, except for the word 0 itself, and are possibly preceded by a + or − sign.
(iii) $L_8 :=$ the language of all well-formed arithmetical expressions without parentheses. These contain all natural numbers, possibly interspersed with the operators +, −, and ×, as in for example $7 + 3 \times 29 - 78$. [*Hint:* First define a regular expression for natural numbers; such a number (almost) never begins with a 0.]

[6]Because $\mathcal{L}(\emptyset) = \emptyset$, we can succinctly use the same symbol to denote both the expression, as its language.

If a language L is regular, then the language L^R is also regular, because if L is described by some regular expression e (so $L = \mathcal{L}(e)$), then L^R is described by the regular expression e^R, because $L^R = \mathcal{L}(e^R)$. (Though, strictly speaking, we are not allowed to write L^R at all, as we have only defined the operation $\ldots^R$ on words and languages, and not on regular expression. Try to figure out what a definition of *reversed* regular expressions would look like.)

Regular languages have more nice properties: if L and L' are regular, then so too are $L \cup L'$ and $L \cap L'$, and also $\overline{L}$. This is easily seen in the case of $L \cup L'$, but the other two are more complicated. Try to figure out why they are regular too.

Convention 3.12 Sometimes, people identify a regular language with one of its describing regular expressions, and speak, say, of "the language $b^*(aab)^*$," although what is actually meant is the language $\mathcal{L}(b^*(aab)^*)$. In this course, we try not to do this, but we try to make explicit the difference between the regular expression and the corresponding language.

Exercise 3.E Which of the following regular expressions describe the same language? For any two expressions, either show that they describe the same language, or else give a word that exemplifies that this is not the case.

(i) $b^*(aab)^*$.
(ii) $b^*(baa)^*b$.
(iii) $bb^*(aab)^*$.

3.3 Context-free grammars

Let us now turn to another often used way of defining languages. Instead of attempting to *describe* a language directly, we give a method of giving a set of rules that describe how the words of the language are *generated* (or, *produced*). Such a set of rules is then called a "grammar."

Example 3.13 $\Sigma = \{a, b\}$. The language $L_9 \subseteq \Sigma^*$ is defined by the *productions* starting with S using these *production rules*:

$$\begin{array}{rcl} S & \to & aAb \\ A & \to & aAb \\ A & \to & \lambda \end{array}$$

The method of generating words is then as follows. We start with S, called the *start symbol.* Both S and A are *nonterminals*. We follow an arrow of S, of which there is only one in this example (though there could have been more in general). If we still have nonterminals in our new word, we follow one of its arrows, and so on, until there are no nonterminals left, and we have successfully produced a word. Some example productions are:

$$\begin{array}{l} S \to aAb \to aaAbb \to aabb; \\ S \to aAb \to aaAbb \to aaaAbbb \to aaabbb. \end{array}$$

This way to represent a language is called a grammar. The grammar above may also be written, more succinctly, as:

$$\begin{array}{rcl} S & \to & aAb \\ A & \to & aAb \mid \lambda \end{array}$$

By which we mean that, from A, two production rules are possible, namely $A \to aAb$ and $A \to \lambda$.

This grammar generates the language L_9, where $L_9 = \{ab, aabb, aaabbb, a^4b^4, \ldots, a^nb^n, \ldots\}$. Or, written more clearly:

$$L_9 = \{a^nb^n \mid n \in \mathbb{N} \text{ and } n > 0\}$$

Definition 3.14 A *context-free grammar* G is a triple $\langle \Sigma, V, R \rangle$ consisting of:

1. An alphabet Σ
2. A set of *nonterminals* V, containing at least the special symbol S: the *start symbol*.
3. A set of *production rules* R of the form

$$X \to w$$

   where X is a nonterminal and w a word made up of letters from the alphabet as well as nonterminals. (Put succinctly: $w \in (\Sigma \cup V)^*$.)

Convention 3.15 We will denote nonterminals by capital letters (S, A, B, etc), reserving lowercase for the alphabet letters (a, b, c, etc), which are also called terminals.

Example 3.16 (i) The language L_9 from example 3.13 is produced by a context-free grammar.

(ii) The language L_{10} is generated by the context-free grammar $\langle \Sigma, V, R \rangle$ having $\Sigma = \{a\}$, $V = \{S\}$, and $R = \{S \to aaS, S \to a\}$. This grammar generates all words containing an odd number of a's.

(iii) The language L_{11} is generated by the context-free grammar $\langle \Sigma, V, R \rangle$ with $\Sigma = \{a, b\}$, $V = \{S, A, B\}$, and $R = \{S \to AB, A \to Aa, A \to \lambda, B \to Bb, B \to \lambda\}$. The language L_{11} consists of all words that start with a sequence of zero or more a's and is followed by a sequence of zero or more b's.

Definition 3.17 Languages generated by context-free grammars are called *context-free languages*. We denote the language generated by G as $\mathcal{L}(G)$.

Remark 3.18 Context-free languages are systematically studied in the course 'Talen en Automaten' (Languages and Automata) in the computer science curriculum.

Example 3.19 The language of well-formed arithmetical expressions, including parentheses, is context-free. (And *not* regular!) A possible grammar for this language is:

$$\begin{array}{lcl}
S & \to & L\,S\,O\,S\,R \mid G \\
L & \to & (\\
R & \to &) \\
O & \to & + \mid \times \mid - \\
G & \to & D\,C \\
D & \to & 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
C & \to & 0C \mid 1C \mid 2C \mid 3C \mid 4C \mid 5C \mid 6C \mid 7C \mid 8C \mid 9C \mid \lambda
\end{array}$$

Can you find productions for the expressions $(33 + (20 * 5))$ and $((33 + 20) * 5)$?

Exercise 3.F The best way to show that a language is context-free is by giving a context-free grammar.

(i) Show that the language of *balanced parentheses expressions* is context-free. By this, we mean the expressions over $\{(,)\}$ where every opening parenthesis is closed with a parenthesis as well, so for example $(()(()))$ and $(())()$ are balanced, but $(()()))$ is not.
(ii) Show that the language L_1 from example 3.5 is context-free.
(iii) Show that the language L_2 from example 3.5 is context-free. (Check example 3.13.)
(iv) Show that the language L_3 from example 3.5 is context-free.
(v) Show that the language L_4 from example 3.5 is context-free.

Exercise 3.G Consider the grammar G_1:

$$\begin{aligned} S &\rightarrow AS \mid Sb \mid \lambda \\ A &\rightarrow aA \mid \lambda \end{aligned}$$

(i) Write G_1 as a triple $\langle \Sigma, V, R \rangle$.
(ii) Give a production demonstrating that $aabb \in \mathcal{L}(G_1)$.
(iii) Can you give a production demonstrating that $bbaa \in \mathcal{L}(G_1)$ within three minutes?

Exercise 3.H Consider the following grammar for the language L_{12}.

$$\begin{aligned} S &\rightarrow aSb \mid A \mid \lambda \\ A &\rightarrow aAbb \mid abb \end{aligned}$$

The nonterminals are S and A, and $\Sigma = \{a, b\}$.
(i) Give productions of *abb* and *aabb*.
(ii) Which words does L_{12} contain?

With some exercise, it is not so hard to see which words are contained in L_{12}. However, it is not as easy to actually *prove* that it contains no other words. Usually, this is possible, depending on the complexity of the grammar, but for our current purposes, this is too hard. A simpler question is the following:

Show that *aab* is not contained in L_{12}.

How does one go about showing this? To show that a word *is* produced by a grammar, one only has to find a generating production, in which one then usually succeeds. But how to show that a grammar is *not* able of producing a word?

In computing science, the notion of an *invariant* was introduced to deal with this kind of proof.

Definition 3.21 An invariant of G is a property P that holds for all words that are generated by G.

To prove that P indeed holds for all $w \in \mathcal{L}(G)$ one needs to demonstrate that:
(i) P holds for S; and
(ii) that P is *invariant* under the production rules, meaning that, if P holds for some word v and v' can be produced from v, then P also holds for v'.

Remark 3.22 Note that in the second item, one must prove the invariance for *all* words v of Σ^*, not only the words that are contained in the grammar's language!

If, then, a word w does not satisfy P, one can conclude that w cannot be generated by the grammar. To summarize, to prove that some word w is not in a grammar's language, using invariants, you do the following:

- Determine some "good" property P (called the invariant).
- Show that P holds for S.
- Show that P is invariant under the production rules.
- Show that w does not satisfy P.

Now let's get back to our problem.

Example 3.23 We want to show that $aab \notin L_{12}$. What would be a good invariant? We take

$$P(w) := \text{ the number of } b\text{'s in } w \geq \text{ the number of } a\text{'s in } w$$

This is indeed an invariant, because:

- $P(S)$ holds.
- If $P(v)$ holds and $v \longrightarrow v'$, then $P(v')$ holds as well. (Check this for every production rule: either both an a and a b are added, or an a and two b's, or neither an a nor a b; in all three cases, the number of b's stays greater or equal to the number of a's.)

  Note that, although definition 3.21 talks of all words that are 'produced', it is only necessary to check all single step productions. (Try to convince yourself to find out why this is the case.)

So now we have proven that $P(w)$ holds for all $w \in L_{12}$. But because $P(aab)$ is obviously not true, we can then conclude that $aab \notin L_{12}$.

Exercise 3.I We take another look at the grammar G_1 from exercise 3.G. It was then already noted that $bbaa \notin \mathcal{L}(G_1)$.

(i) Is

$$P(w) := w \text{ does not contain } ba \text{ as sub-word}$$

an invariant for G_1 that proves that $bbaa \notin \mathcal{L}(G_1)$?

(ii) Is

$$P(w) := w \text{ does neither contain } ba \text{ nor } bA \text{ as sub-word}$$

an invariant for G_1 that proves that $bbaa \notin \mathcal{L}(G_1)$?

(iii) Is

$$P(w) := w \text{ does neither contain } ba\text{, nor } bA\text{, nor } bS \text{ as sub-word}$$

an invariant for G_1 that proves that $bbaa \notin \mathcal{L}(G_1)$?

Exercise 3.J Use invariants to prove that:

(i) $bba \notin L_{12}$.

(ii) *aabbb* is not produced by the grammar of L_3 that you constructed in exercise 3.F.
(iii) *aabbb* is not produced by the grammar for L_4 that you constructed in exercise 3.F.

Remark 3.24 Context-free grammars are called *context-free*, because the items on the left hand side of production rules are only allowed to be single nonterminals. So for example, the rule $Sa \to Sab$ is not allowed. And therefore, one never needs to take into account the *context* of the nonterminal (the symbols that may surround it in an intermediate step of production).

3.4 Right linear grammars

A well-known restricted form of context-free grammars are the *right linear grammars*.

Definition 3.25 A *right linear grammar* is a context-free grammar in which the production rules are always of the form

$$\begin{array}{lcl} X & \to & wY \\ X & \to & w \end{array}$$

where X and Y are nonterminals and $w \in \Sigma^*$.

That is, in a right linear grammar, nonterminals are only allowed at the end of a rewrite, on the right hand side of a production rule.

Example 3.26 (i) In example 3.16, only the grammar L_{10} is right linear.
(ii) Sometimes, for a context-free grammar that is not right linear, one can find an equivalent right linear grammar. For example, the following grammar (over $\Sigma = \{a, b\}$) is right linear and generates L_{11} from example 3.16:

$$\begin{array}{lcl} S & \to & aS \mid B \\ B & \to & bB \mid \lambda \end{array}$$

There is a mathematical theorem that states that the class of languages that can be produced by right linear grammars is exactly the class of regular languages.

Theorem 3.27 *A language L is regular if and only if there is a right linear grammar that describes it.*

Corollary 3.28 *A regular language is always context-free.*

We will not prove this theorem. To prove it, you have to show how to create a right linear grammar for every regular expression such that their languages are the same, and the other way around, creating a regular expression for each right linear grammar, such that their languages are the same. We will illustrate this by giving an example.

Example 3.29 Consider the regular expression

$$ab^*(ab \cup \lambda)(a \cup bb)^*$$

A right linear grammar producing the same language as that of the expression, is:

$$\begin{array}{lcl} S & \to & aA \\ A & \to & bA \mid B \\ B & \to & abC \mid C \\ C & \to & aC \mid bbC \mid \lambda \end{array}$$

Sometimes, right linear grammars are depicted with so-called *syntax diagrams*. We will just show an example of what such a diagram can look like. Figure 3.1 displays the syntax diagram corresponding to the grammar from example 3.29.

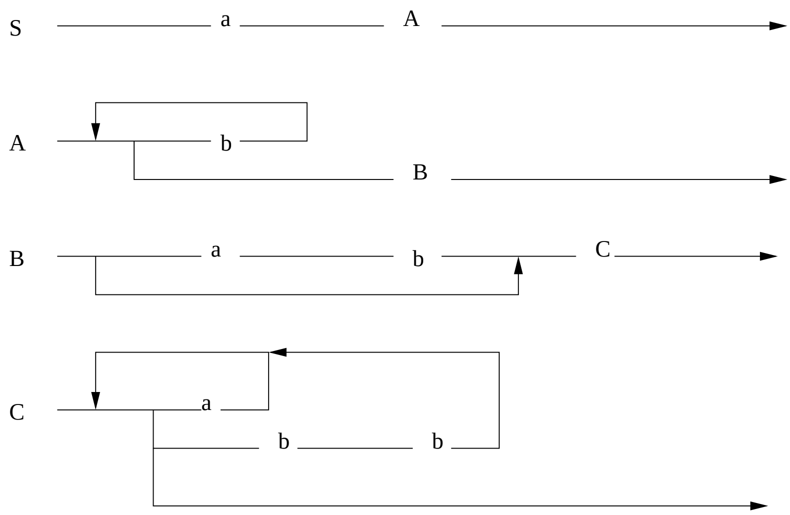


Figure 3.1: Syntax diagram for example 3.29

A direct result of theorem 3.27 is that these two languages, over the alphabet $\{a, b\}$, are regular (see example 3.16): $L_{10} = \{a^n \mid n \text{ is odd}\}$ and $L_{11} = \{wv \mid w \text{ contains only } a\text{'s}, v \text{ contains only } b\text{'s}\}$. Try to construct regular expressions for these languages.

Exercise 3.K (i) Consider the following grammar over the alphabet $\{a, b, c\}$:

$$\begin{array}{rcl} S & \to & A \mid B \\ A & \to & abS \mid \lambda \\ B & \to & bcS \mid \lambda \end{array}$$

Check whether you can produce these words with the grammar: *abab*, *bcabbc*, *abba*. If you can, provide a production. If not, provide an invariant which you could use to prove that the word can not be produced.

(ii) Describe the regular language L_{13} that this grammar generates, with a regular expression.

(iii) Construct a right linear grammar for the language L_{14} consisting of all words of the shape $ab \ldots aba$ (that is, words with alternating a's and b's, starting and ending with an a; make sure to also include the word a).

Exercise 3.L Give right linear grammars for the languages of exercise 3.D:

(i) $L_6 := \{w \in \{a, b\}^* \mid \text{every } a \text{ in } w \text{ is directly followed by a } b\}$,

(ii) $L_7 :=$ the language of well-formed integer expressions. These consist of the symbols $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, but never start with a 0, except for the word 0 itself, and may be preceded by a + or a − sign.

(iii) $L_8 :=$ the language of well-formed arithmetical expressions without parentheses. These consist of natural numbers, interspersed with the operators $+$, $-$, and $\times$, as in for example $7 + 3 \times 29 - 78$.

Exercise 3.M Here we give a grammar for a small part of the English language.

$$
\begin{array}{rcl}
S = \langle\text{sentence}\rangle & \to & \langle\text{subjectpart}\rangle\langle\text{verbpart}\rangle\textbf{.} \\
\langle\text{sentence}\rangle & \to & \langle\text{subjectpart}\rangle\langle\text{verbpart}\rangle\langle\text{objectpart}\rangle\textbf{.} \\
\langle\text{subjectpart}\rangle & \to & \langle\text{name}\rangle \mid \langle\text{article}\rangle\langle\text{noun}\rangle \\
\langle\text{name}\rangle & \to & \textbf{John} \mid \textbf{Jill} \\
\langle\text{noun}\rangle & \to & \textbf{bicycle} \mid \textbf{mango} \\
\langle\text{article}\rangle & \to & \textbf{a} \mid \textbf{the} \\
\langle\text{verbpart}\rangle & \to & \langle\text{verb}\rangle \mid \langle\text{adverb}\rangle\langle\text{verb}\rangle \\
\langle\text{verb}\rangle & \to & \textbf{eats} \mid \textbf{rides} \\
\langle\text{adverb}\rangle & \to & \textbf{slowly} \mid \textbf{frequently} \\
\langle\text{adjectives}\rangle & \to & \langle\text{adjective}\rangle\langle\text{adjectives}\rangle \mid \lambda \\
\langle\text{adjective}\rangle & \to & \textbf{big} \mid \textbf{juicy} \mid \textbf{yellow} \\
\langle\text{objectpart}\rangle & \to & \langle\text{adjectives}\rangle\langle\text{name}\rangle \\
\langle\text{objectpart}\rangle & \to & \langle\text{article}\rangle\langle\text{adjectives}\rangle\langle\text{noun}\rangle
\end{array}
$$

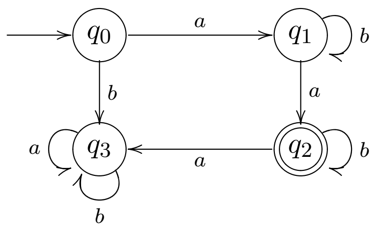
(i) Is this grammar right linear?
(ii) Show how you produce the following sentence: *Jill frequently eats a big juicy yellow mango.*
(iii) Make some more sentences.

3.5 Automata

At the start of this section, we stated that languages can also be formalized using automata. To explain this connection, we first define what an automaton is.

In computer science, there is the study of machines. For example by looking at computers themselves, but also at a higher, conceptual level, computer scientists study idealized, abstract machines. An advantage of studying such an idealized and abstract *machine model* is that it is easier to study the important properties they have. A prolific class of these abstract machines are the *finite automata*. These finite automata have many more applications than just modelling simple calculations (as machines usually do). Finite automata, and simple extensions, can model processes as well, for example. Let us first take a look at what a finite automaton is, and how one 'calculates' with it. Here is an example:

M_0 :



(3)

An automaton is a so-called directed graph, in which the lines are *arrows* and have *labels*. (See Section 4.1 for a formal account of graphs.) The nodes of the graph are referred to as the *states* of the automaton, here: q_0, q_1, q_2, and q_3. The states are commonly drawn as circles with their name written inside. There are two distinguished types of states that have a special role:

1. The *initial state*, distinguished by an incoming arrow that doesn't depart from any other state. Every automaton has *exactly one* initial state, often named q_0 as in the example above.

2. The *final states*, distinguished by drawing their circle with a double border. Every automaton has *at least one* final state. The example above has exactly one, namely q_2. (It is allowed to have the initial state simultaneously be a final state.)

We can regard an automaton as a (very) simple computer that can compute things for us. This computation happens in the following way: you give it a word (in the example, this would be a word over the alphabet $\{a, b\}$), then the automaton computes according to the word, following the appropriate arrow for each subsequent letter of the given word, and then after a number of steps *halts*, either in a final state, or in a non-final state. In the first case, we say the automaton *accepts* the word, in the second, it *rejects* the word.

Example 3.30 The automaton M_0 accepts the word *aa*: it starts in state q_0, and then reads the first letter a. This brings it to the state q_1, and it then reads the second letter, another a. This then brings it to state q_2. There is no input any more, and the automaton has halted. Because it halted in a final state, *aa* is accepted by the automaton. We depict this *computation* with the notation:

$$q_0 \stackrel{a}{\rightarrow} q_1 \stackrel{a}{\rightarrow} q_2$$

So, the notation shows how each subsequent letter is consumed, but does not depict whether the word has been accepted or not, which means you have to add this conclusion in writing. In this case, the computation ends in the final state q_2, and thus *aa* is accepted. Check that the words *abbba* and *ababb* are accepted as well, though *ababab* and *bab* are not.

Before continuing, we give a formal definition of the notion of a finite automaton.

Definition 3.31 A *finite automaton* is a quintuple $M := \langle \Sigma, Q, q_0, F, \delta \rangle$:

1. A finite set Σ, the input alphabet, or set of *atomic operations*,
2. A finite set Q of *states*,
3. A distinguished state $q_0 \in Q$ called the *initial state*,
4. A non-empty set of distinguished states $F \subseteq Q$, the *finite states*,
5. A *transition function* δ, that maps every tuple of a state q and an action a to a new state q'. (These are the labelled arrows in (3).)

A finite automaton is also called a *DFA*, as an abbreviation for "Deterministic Finite Automaton."

Example 3.32 This means that the automaton M_0 from (3) is actually the quintuple $\langle \Sigma, Q, q_0, F, \delta \rangle$, where $\Sigma = \{a, b\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $F = \{q_2\}$, and the transition function δ is defined by:

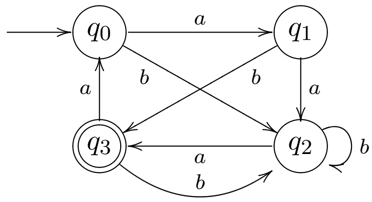
$$\begin{array}{rclcrcl} \delta(q_0, a) & = & q_1 & \quad & \delta(q_0, b) & = & q_3 \\ \delta(q_1, a) & = & q_2 & & \delta(q_1, b) & = & q_1 \\ \delta(q_2, a) & = & q_3 & & \delta(q_2, b) & = & q_2 \\ \delta(q_3, a) & = & q_3 & & \delta(q_3, b) & = & q_3 \end{array}$$

Specifically, note how the deterministic behaviour is captured by the fact that for every state and letter, exactly one transition is defined.

Instead of writing such a mathematical definition out in full, we usually simply draw a diagram as in the example on page 34.

Exercise 3.N Consider the automaton M_1.

M_1 :



Here, $Q = \{q_0, q_1, q_2, q_3\}$, $F = \{q_3\}$, and $\Sigma = \{a, b\}$.

(i) Check whether these words are accepted or not: *abaab*, *aaaba*, *bab*, λ, and *aabbab*.

(ii) Are these statements true? Give a proof or counterexample.

(1) If w is accepted, then so is $wabba$.

(2) If w is accepted, then wab is not accepted.

(3) If w is not accepted, then waa will not be accepted either.

(4) If w is not accepted, then neither is wbb.

3.6 Languages and automata

We can think of Σ as the set of "atomic operations" that lead us from one state to another, but also as the set of symbols of an alphabet, as we have done already above. If we think of Σ as representing the alphabet, the automaton can be seen as a *language recognizer*. In this way, for each automaton, there is a corresponding language, namely the language recognized by the automaton.

Definition 3.33 For an automaton $M := \langle \Sigma, Q, q_0, F, \delta \rangle$, we define the *language of* M, to be $L(M)$:

$$L(M) := \{w \in \Sigma^* \mid w \text{ is accepted by } M\}.$$

So: $w \in L(M)$ if and only if the automaton M halts in a final state after consuming all of w.

Let us take a look at our initial automaton M_0, from page 34. It accepts the following words:

- aa is accepted,
- awa is accepted, with w an arbitrarily long sequence of b's,
- $awav$ is accepted, with w and v both arbitrarily long sequences of b's.

If we go "past" state q_2, we can never get back to a final state, and so the above description lists all words that the automaton accepts. Summarized:

$$L(M_0) = \{ab^n ab^m \mid n, m \geq 0\}$$

Because we learned regular languages and expressions previously, we see that this language indeed is described by a regular expression, namely:

$$L(M_0) = \mathcal{L}(ab^* ab^*).$$

Trying to find a similar corresponding regular expression for M_1, turns out to be a bit harder:

- ab is accepted,
- ba is accepted,
- aaa is accepted,
- aab^ka is accepted, with $k \geq 0$,
- bb^ka is accepted, with $k \geq 0$,
- $aab^ka(ba)^l$ is accepted, with $k \geq 0$, $l \geq 0$,
- $bb^ka(ba)^l$ is accepted, with $k \geq 0$, $l \geq 0$,

…and this is not all, because if we go "past" state q_2, we can in fact loop back to q_2 again. How can we then systematically analyze the language of an automaton? A method for doing so is constructing a corresponding *grammar*, that *generates* the same language as the automaton recognizes. This is done in this way:

1. For every state q_i, introduce a nonterminal X_i, and distinguish the starting nonterminal S for the initial state q_0.
2. For every transition $q_i \overset{a}{\to} q_j$ in the automaton, add the production rule $X_i \to aX_j$.
3. For every final state $q_i \in F$, add the production rule $X_i \to \lambda$.

Constructing the grammar G_2 (with $\Sigma = \{a, b\}$) for the automaton M_1, we then get:

$$\begin{array}{rcl} S & \to & bB \mid aA \\ A & \to & aB \mid bC \\ B & \to & bB \mid aC \\ C & \to & bB \mid aS \mid \lambda \end{array}$$

Note that this grammar is *right linear*, and so the language $L(M_1)$ is indeed *regular*.

This new description of the language $L(M_1)$, using a grammar, is interesting, if only because it is a new description that we haven't seen before. But what happens if we try to 'optimize' this grammar by substituting symbols? First, inline the rule $A \to aB \mid bC$ into the rule $S \to bB \mid aA$ to get $S \to bB \mid aaB \mid abC$. Then, do the same for the C rule. This gives us a new grammar yet again, G_3, still generating the same language, and still right linear as well:

$$\begin{array}{rcl} S & \to & bB \mid aaB \mid abbB \mid abaS \mid ab \\ B & \to & bB \mid abB \mid aaS \mid a \end{array}$$

Exercise 3.O Conclude from the grammar G_3 that
(i) $(aba)^kab \in \mathcal{L}(G_3)$ for all $k \geq 0$,
(ii) $aab^ka \in \mathcal{L}(G_3)$ for all $k \geq 0$,
(iii) if $w \in \mathcal{L}(G_3)$, then also $abaw \in \mathcal{L}(G_3)$,
(iv) if $w \in \mathcal{L}(G_3)$, then also $aaaaw \in \mathcal{L}(G_3)$,

It is general knowledge that any finite automaton can be translated into a right linear grammar, as we have seen and done for M_1.

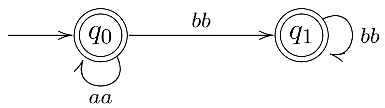
Theorem 3.34 *For every finite automaton M, a right linear grammar G can be constructed such that $\mathcal{L}(G) = L(M)$.*
(The language that G generates is the same as the language that M accepts.) A direct result is that the language $L(M)$ of a finite automaton M is always regular.

Exercise 3.P Construct a right linear grammar for the finite automaton M_0, similarly as one was constructed for M_1 above. After that, optimize the grammar by removing useless production rules and/or substituting nonterminals for their production rules.

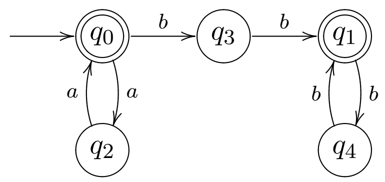
We can also translate the other way around: constructing a finite automaton for a given right linear grammar. Take a look at the right linear grammar G_4:

$$\begin{aligned} S &\rightarrow aaS \mid bbB \mid \lambda \\ B &\rightarrow bbB \mid \lambda \end{aligned}$$

First, we introduce a state for every nonterminal, and make transitions labelled with letter sequences instead of just letters. Each nonterminal that leads to λ becomes a final state, and S becomes the initial state.

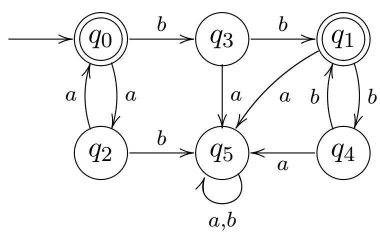


Then, we expand the letter sequences into single letter transitions by adding intermediate states, and get:



And now we are almost done. The remaining problems lies in the fact that, in a full automaton, *every state and letter* must have an outgoing transition leading to a new state, which is not the case as of yet. So solve this, we add a so-called “sink” that catches all additional useless transitions, and that any computation cannot escape from into a final state any more. This gives us our final automaton, where q_5 is the newly added sink:

M_2 :



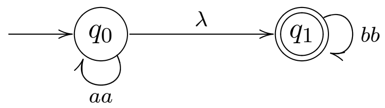
Remark 3.35 Here, we conveniently drew a single arrow, from q_5 to itself, with two labels, that actually stands for two arrows with the respective labels, because else the drawing would become a bit of a mess. Instead of adding all these arrows, we could have agreed upon

omitting them, as well as the "sink." We don't do so, however, because a word such as *bba* should clearly be *not* accepted by M_2, which is definitely the case in the final automaton, but without the added arrows and sink as in the earlier version, it would halt in the final state q_1 (with remaining input), which is a bit unclear ...

This procedure of constructing automata for right linear grammars works well in general, except in the case of production rules of the shape $S \to B$, where the right hand side doesn't contain any letters in front of the nonterminal. Then, the word in front of the nonterminal is λ, as in the case of grammar G_5:

$$\begin{array}{lcl} S & \to & aaS \mid B \\ B & \to & bbB \mid \lambda \end{array}$$

Performing the first step of constructing a corresponding automaton, we get:



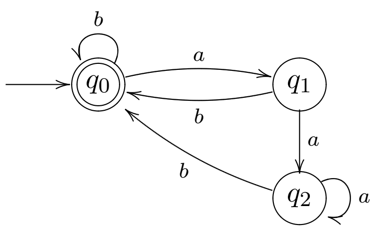
And end up with a transition labelled λ, the empty word, which evidently leads to a problem in the next step. The general solution is to first create an *equivalent* right linear grammar without any productions of this shape. This is always possible, but we won't show how to do this in general. For the grammar G_5, this initial step would lead to G_4, so that the automaton M_2 indeed accepts the language generated by G_5. (And G_4 and G_5 generate the same language: check this yourself!)

Theorem 3.36 *For every right linear grammar G, a finite automaton M can be constructed such that $L(M) = \mathcal{L}(G)$.*

Exercise 3.Q Construct a finite automaton that recognizes the language of the following grammar:

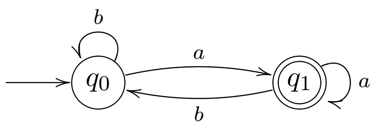
$$\begin{array}{lcl} S & \to & abS \mid aA \mid bB \\ A & \to & aA \mid \lambda \\ B & \to & bB \mid \lambda \end{array}$$

Exercise 3.R Consider the automaton M_3:

M_3 :


Construct a right linear grammar that generates $L(M_3)$.

Exercise 3.S Consider the finite automaton M_4:

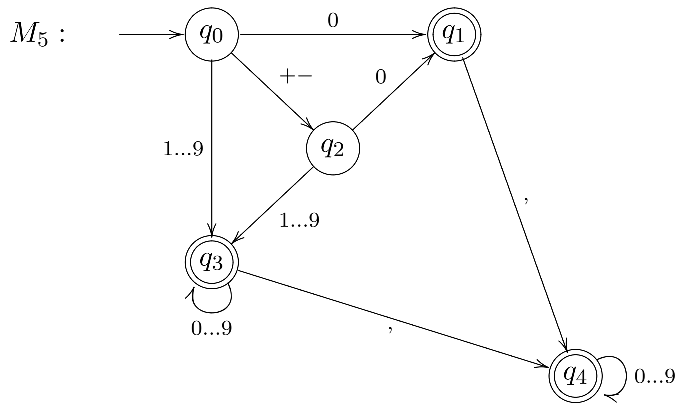
M_4 :


(i) Construct a right linear grammar that generates $L(M_4)$.
(ii) Provide a (simple) description of $L(M_4)$.
(iii) If all states are made into final states, which language does M_4 recognize?
(iv) If we switch the final states with non-final states (every final state becomes a non-final state and the other way around), which language would M_4 recognize?

Exercise 3.T Define $L_{15} := \{(ab)^k(aba)^l \mid k, l \geq 0\}$ over the alphabet $\Sigma = \{a, b\}$. Construct a finite automaton that recognizes this language.

Exercise 3.U Define $L_{16} := \{(ab)^k x(ab)^l \mid x \in \{a, b\}, k, l \geq 0\}$ over the alphabet $\Sigma = \{a, b\}$. Construct a finite automaton that recognizes this language.

Exercise 3.V Consider the automaton M_5 that accepts (as input) numbers in 'normal' decimal form. Such a number may optionally be preceded by a + or − sign, but not with a 0, except of course for the number 0 itself and numbers like 0, 42. In the diagram, the arrow with $0 \ldots 9$ represents 10 individual arrows with 0, 1, …, 9 as labels, respectively, etc.



(i) Is the automaton M_5 well-formed? Are all 'correct' numbers accepted and all 'incorrect' ones rejected?
(ii) Revise M_5 until it is well-formed (according to your idea of what decimal notation is).
(iii) Give a right linear grammar that generates the same language as M_5.

3.7 Important concepts

4 Discrete mathematics

This chapter deals with a number of small subjects that we have collected under the title of *discrete mathematics*. Notable about all these subjects, and the reason we have grouped them under this title, is that they all deal with natural numbers: the number of vertices in a graph, the number of steps in a recursive computation, the number of ways to traverse a grid from one point to another, etc. We never have to deal with problems of continuity, with an infinite number of points between two objects.

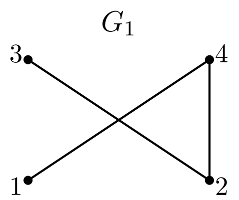
For more information, take a look at the introductory texts [1] or [4].

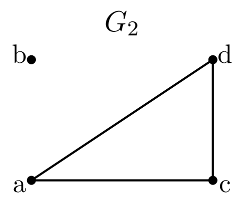
4.1 Graphs

This first section will give a short introduction to graph theory. Graphs are often encountered when studying things like languages, networks, data structures, electrical circuits, transport problems, flow diagrams, and so on.

Intuitively, a graph consists of a set V of vertices and a set E of edges between vertices.

Example 4.1 Two examples are:





All uncertainties you might have with this informal description, such as the questions: 'When are two graphs the same?', 'Must the vertices lie on a flat surface?', or 'May the edges intersect each other?', are resolved with a formal definition:

Definition 4.2 A *graph* is a tuple $\langle V, E\rangle$, of which V is a set of names, and E a set of 2-element subsets of V. The elements of V are called *vertices* (singular: *vertex*), or sometimes *nodes*, and the elements of E are called *edges*. We denote the edges not as usual sets, $\{v, w\}$, but instead as (v, w). So keep in mind that (v, w) and (w, v) denote the same edge.

Example 4.3 The graph G_1 from example 4.1 is then $\langle V, E\rangle$, with $V = \{1, 2, 3, 4\}$ and $E = \{(1, 4), (2, 3), (2, 4)\}$. And $G_2 = \langle V, E\rangle$, with $V = \{a, b, c, d\}$ and $E = \{(a, c), (a, d), (c, d)\}$.

Remark 4.4 Notable in our definition 4.2, is that V is allowed to be the empty set. Also, the edges don't have a *direction*: they are *lines* instead of *arrows*. Technically then, we are dealing with *undirected* graphs, where the automata of chapter 3 were so-called directed graphs, with arrow edges.

Another remark is that it is technically impossible for an edge to connect a vertex to itself, because such an edge, connecting say v to itself, would be $(v, v) = \{v, v\} = \{v\}$, but that is a 1-element subset of V, and thus excluded by definition. Also impossible are two edges connecting the same two vertices. Of course, all these impossibilities could be resolved by changing the definition, but as it happens, this definition is elegantly simple, and already leads to enough mathematical expressivity and contemplation.

Definition 4.5 Let $G = \langle V, E \rangle$ be a graph, and $v, w \in V$.

- A *neighbor* of v is any vertex x for which $(v, x) \in E$.
- The *degree* (or: *valency*) of v is the number of neighbors v has.
- A *path* from v to w is a sequence of *distinct* edges $(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)$, with $n > 0$, $x_0 = v$, and $x_n = w$. We will also denote such a path as: $x_0 \to x_1 \to x_2 \to \cdots \to x_n$.
- With the statement 'G is *connected*' we mean to say that every pair of vertices has a path connecting the two.
- A *component* is an as large as possible connected part of G.
- A *cycle*, also sometimes called a *circuit*, is a path from a vertex to itself.
- With the statement 'G is *planar*', we mean to say that G can be drawn upon the plane, i.e. a flat surface, such that no edges intersect each other. (You are allowed to bend the edges if that is necessary.)
- With the statement 'G is a *tree*' we mean that G is connected and doesn't contain any cycles. (Which means, you can indeed picture it as a tree.)

To give an example of how real-world problems can be stated in terms of graph theory: the question whether a graph is planar is relevant to whether we can burn an electrical circuit onto a single layer chip.

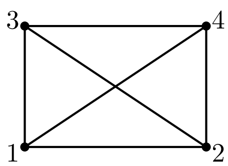
Exercise 4.A Prove that in a tree, between any two points v and w, there is exactly one path that connects the two.

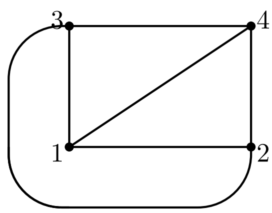
Exercise 4.B A *bridge* in a graph G is an edge e for which: if you remove e from G, then the number of components of G increases. Prove that in a tree, every edge is a bridge.

Example 4.6 Let's take a look at some 'real-world' graphs.

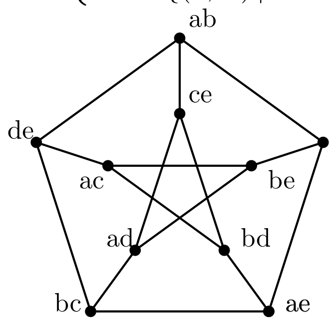
(i) The '*country graph*' is $\langle V, E \rangle$, in which V is the set of all countries of the world, and E is the relation 'borders' (on the ground). The neighbors of The Netherlands are Germany and Belgium, and thus the degree of The Netherlands is 2. A path from The Netherlands to Spain would be, for example: The Netherlands $\to$ Germany $\to$ France $\to$ Spain. The country graph is not connected. $\{\text{England}, \text{Scotland}, \text{Wales}\}$ is a component. The Netherlands $\to$ Germany $\to$ Belgium $\to$ The Netherlands is a cycle. The country graph is planar.

(ii) $K_4 = \langle \{1, 2, 3, 4\}\ ,\ \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\} \rangle$ (the *complete graph* with four points). Generally: K_n is the graph $\langle \{1, \ldots, n\}\ ,\ \{(v, w) \mid 1 \leq v < w \leq n\} \rangle$. The graph K_4 is planar. To understand this, note how instead of intuitively drawing it as the left picture, we can also draw it as in the right picture:





(iii) The *Petersen graph* is $\langle V, E\rangle$ with $\begin{cases} V = \{\text{ab, ac, ad, ae, bc, bd, be, cd, ce, de}\} \\ E = \{(v, w) \mid v \text{ and } w \text{ have no letters in common}\} \end{cases}$



It can be shown that the Petersen graph is not planar.

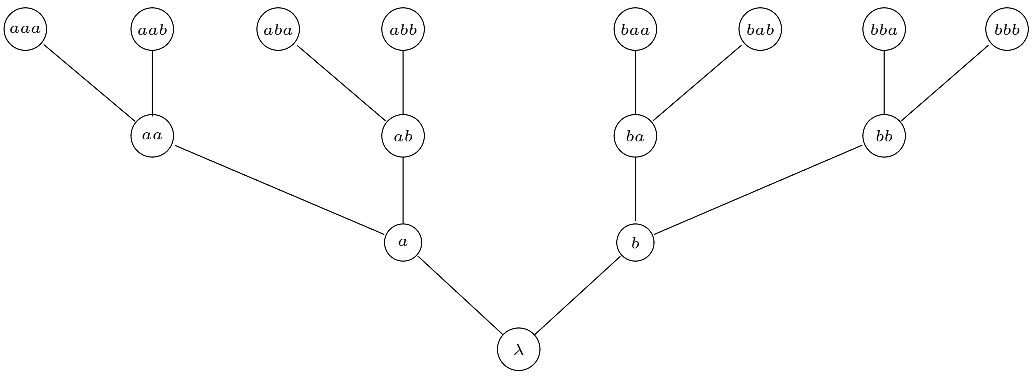
(iv) If a language is given by an inductive definition, we can make a corresponding graph. Take for example the language produced as follows:

axiom	λ
rule	$x \rightarrow xa$ $y \rightarrow yb$

We can then create the corresponding graph by:

- first, writing down all words that are included in the language, merely by an axiom (these are then the initial vertices in the graph),
- then, step-wise adding words (vertices) to the graph on the grounds of the languages' production rules, whereby we connect any new word to the word it was built up from (with its producing rule) with an edge

For the above language, we then get this (unfinished, infinite) graph:

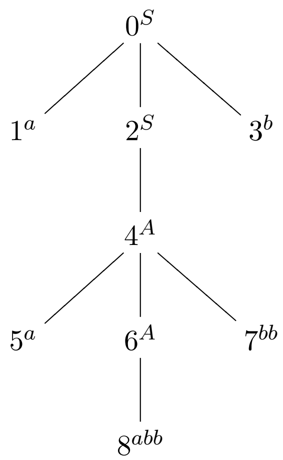


The graph is indeed a tree.

(v) For a word that is produced by a context-free grammar, you can create a *parse tree*, that demonstrates how the word is produced from the grammar. Taking the context-free grammar from exercise 3.H:

$$\begin{aligned} S &\rightarrow aSb \mid A \mid \lambda \\ A &\rightarrow aAbb \mid abb \end{aligned}$$

The word *aaabbbbb* is produced by this grammar, in the following way: $S \to aSb \to aAb \to aaAbbb \to aaabbbbb$. Below is the parse tree corresponding to this production.



In the parse tree, a leaf corresponds to a final sequence of letters, and a non-leaf vertex corresponds to a nonterminal that produces the word parts depicted by the vertices drawn below it. So, following the production rules $S \to aSb$, we draw a vertex with label S, and three subvertices: a leaf with label s, a non-leaf with label S, and a leaf with label b. The next production step, $A \to aAbb$, continues the graph generation, until we have ended up with the above. Reading the leaves, beginning at the top left, counter-clockwise around the graph, we get the produced word: *aaabbbbb*.

Remark 4.7 Information scientists almost always draw trees 'upside down': with the 'trunk' at the top and the branches pointing down, whereas mathematicians often draw trees with the right side up (see previous example). In a parse tree, often only the labels are drawn, and not the numbering as included in the example above.

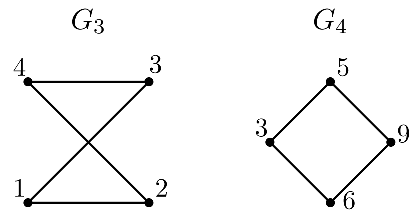
Exercise 4.C Prove that for all $n \geq 1$, it holds that K_n has exactly $\frac{1}{2}n(n-1)$ edges.

4.2 Isomorphic graphs

Definition 4.8 A function f from a set A to a set B is called a *bijection* if every element in B has exactly one original. Formally: for every $b \in B$, there exists some $a \in A$ such that $f(a) = b$ and for every $a' \in A$ it holds that if $f(a') = b$, then $a = a'$.

Definition 4.9 We say that two graphs $\langle V, E\rangle$ and $\langle V', E'\rangle$ are *isomorphic* if there is a bijection $\varphi : V \to V'$ such that for all $v, w \in V$, we have $(v, w) \in E$ if and only if $(\varphi(v), \varphi(w)) \in E'$. Put differently: two graphs are called isomorphic if, disregarding the labels of the vertices, they are the same. A 'property preserving' bijection φ like the above is called an *isomorphism*.

Example 4.10 Consider the following two graphs.

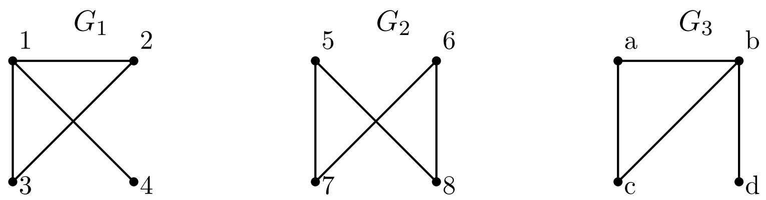


The graphs G_3 and G_4 are isomorphic, because we have an isomorphism φ:

$$\begin{array}{rcl} 1 & \mapsto & 6 \\ 2 & \mapsto & 9 \\ 3 & \mapsto & 3 \\ 4 & \mapsto & 5 \end{array}$$

Isomorphic graphs are 'the same' if we are only interested in the graph-theoretic properties. For example: If G and G' are isomorphic and G is connected, then so is G'.

Exercise 4.D Check which of the graphs below are isomorphic to each other:

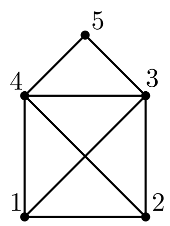


If two graphs are isomorphic, give an isomorphism between the two. If they are not, then explain why such an isomorphism cannot exist.

4.3 Euler and Hamilton

Definition 4.11 An *Euler path*, in a graph $\langle V, E\rangle$, is a path in which every edge from E is included exactly once. An *Euler circuit*, or *Euler cycle* is an Euler path that is a cycle, as well.

Example 4.12 Consider the graph:



The path $1 \to 4 \to 5 \to 3 \to 1 \to 2 \to 4 \to 3 \to 2$ is an Euler path. Because vertex 1 has degree 3, this graph has no Euler circuit. This can be seen by examining the times that such a cycle would traverse vertex 1: if it does so only once, then one of its three edges cannot have been traversed, and if the cycle would traverse vertex 1 twice (or more), then at least one of its edges must have been used multiple times. So apparently, the graph doesn't admit an Euler cycle.

If we write out this argument in general, we have proved the following simple proposition:

Theorem 4.13 (Euler) *In a connected graph with at least two vertices:*

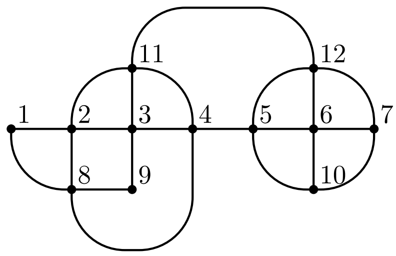
1. An Euler circuit exists if and only if every vertex has an even degree.

2. An Euler path exists if and only if there are at most two vertices of odd degree.

Euler circuits are of importance to for example newspaper deliverers, or neighborhood police officers, that want to efficiently pass through their neighborhood, ideally visiting each street exactly once, and ending up where they started. A whole different problem is of importance to the 'traveling salesman', who wants to visit each of his customers (or cities) exactly once, returning home afterwards. He would be interested in a so-called 'Hamilton circuit':

Definition 4.14 A *Hamilton path*, in a graph $\langle V, E\rangle$, is a path in which each vertex of V is traversed exactly once. A *Hamilton circuit*, *Hamilton cycle* is a Hamilton path, that is a cycle, as well (with the exception that the initial and final vertices are allowed to be the same).

Exercise 4.E Given here is a city map G of a village, on which streets are indicated by edges. There are pubs located on every vertex. The pubs are indicated by vertices, numbered 1 through 12:



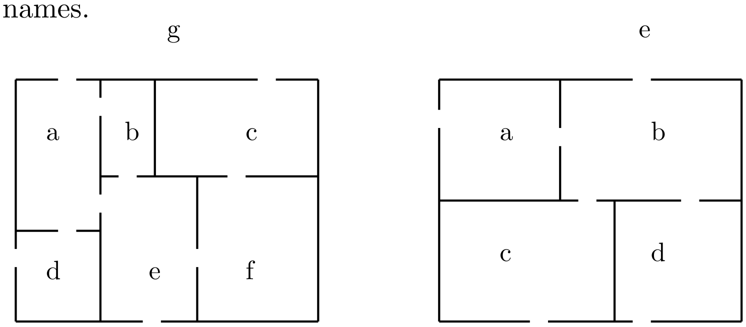
Formulate the following questions in terms of Hamilton- and Euler- circuits and paths, and answer them as well:

(i) Is it possible to make a walk in such a way that every street is traversed only once? If so, give an example, and if not, explain why.

(ii) Is it possible to make a walk, passing every street exactly once, and starting and ending in pub 3? If so, give an example, and if not, explain why.

(iii) Can a pub-crawl be organized such that every pub is visited exactly once? If so, give an example, and if not, explain why.

Exercise 4.F Below, two floor plans of houses are given, in which the rooms and the garden have been given names.

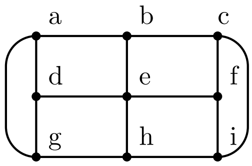


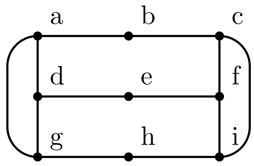
(i) For both floor plans, draw a corresponding graph, where the rooms, including the garden, become vertices, and the doors connecting rooms become edges connecting vertices.

(ii) For both houses, check whether it is possible to make a stroll through the house in such a way that every door is used exactly once, and you end up in the room where you started out. Explain your answer, and if you argue that such a stroll is possible, give an explicit example.

(iii) For both houses, check whether a stroll exists that passes through each room, as well as the garden, exactly once, returning to the original room afterwards. Explain your answer and give concrete examples if these strolls indeed exist.

Exercise 4.G Which of the following graphs has a Hamilton circuit? Provide such a circuit or explain why a Hamilton circuit cannot exist.





Exercise 4.H Let $Q_3 = \langle V, E \rangle$ be the three dimensional hypercube graph, where V are the eight corners of a cube and E are the twelve edges connecting each vertex with three other vertices.

(i) Is Q_3 planar?

(ii) Does Q_3 have a Hamilton circuit?

(iii) Does Q_3 have an Euler circuit?

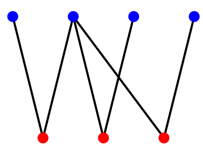
Don't forget to explain your answers!

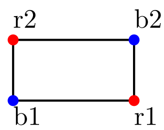
Exercise 4.I Show that the Petersen graph does contain a Hamilton path, but doesn't contain a Hamilton cycle.

4.4 Colorings

Definition 4.15 A graph $\langle V, E \rangle$ is called *bipartite* if V can be written as $V_1 \cup V_2$ where V_1 and V_2 are disjoint, in such a way that every edge leaving from a vertex of V_1, leads to a vertex of V_2, and the other way around. Put differently: you can color the vertices red and blue in such a way that no edge connects two vertices of the same color.

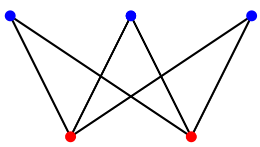
Example 4.16 Two examples of bipartite graphs:



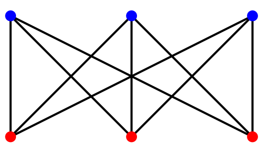


Example 4.17 The *complete bipartite graph* with m red and n blue vertices, where every red vertex is connected to every blue vertex and the other way around, is denoted by $K_{m,n}$.

$K_{2,3}$



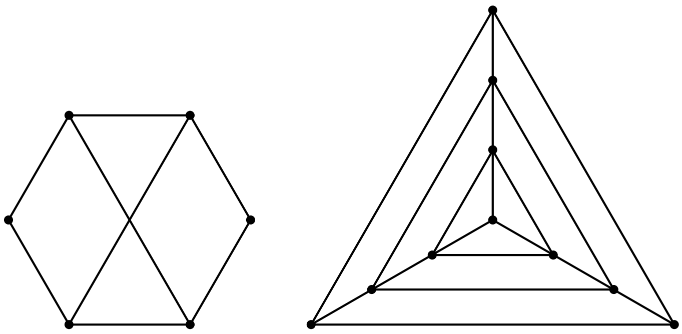
$K_{3,3}$



Definition 4.18 A *vertex coloring* of a graph $\langle V, E\rangle$ is a function $f : V \to \{1, \ldots, n\}$, such that for every edge (v, w), we have $f(v) \neq f(w)$. So: each vertex is assigned one of n colors, and neighboring vertices never share the same color.
The *chromatic number* of a graph is the least number n for which such a coloring is possible.

Example 4.19 Bipartite graphs are then graphs whose chromatic number is either 1 or 2.

Exercise 4.J Find the chromatic number for the following two graphs. Explain your answer.



Exercise 4.K Show that if a bipartite graph has a Hamilton path, the number of red vertices and blue vertices differs at most one.

Exercise 4.L At a certain university quite a lot of language courses are being offered: Arabic, Bulgarian, Chinese, Dutch, Egyptian, French and German. When creating the schedule for these courses the schedule maker has to take these requirements into account:

- All languages are being taught each day.
- Each lesson takes 105 minutes.
- The slots for the lessons are 08.45–10.30, 10.45–12.30, 13.45–15.30, 15.45–17.30 and 18.45–20.30.
- The building with five lecture rooms can only be rented as a whole, so the more courses are being taught in parallel, the cheaper it will be for the university.
- Some students have registered for more than one course and hence these courses should not be given in parallel. In the table below the places marked with $*$ indicate that there is at least one student that has registered for both the language in this row as the language in this column.

	A	B	C	D	E	F	G
A		*	*	*			*
B	*		*	*	*		*
C	*	*		*		*	
D	*	*	*			*	
E		*					
F			*	*			*
G	*	*				*	

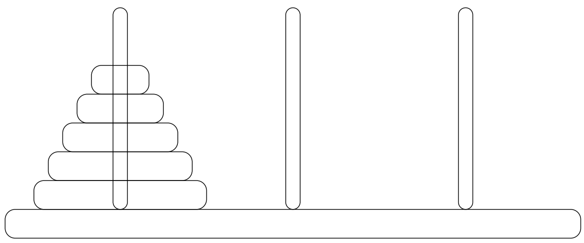
Give a schedule for the daily lessons that complies with the given requirements. Use graph theory to prove that your schedule is optimal.

An interesting theorem by Appel and Haken, the proof of which was only found in 1975 and which is unfortunately too complicated to be included in this course, is the following:

Theorem 4.20 (Four color theorem.) *The chromatic number of any planar graph is at most 4. Put differently: every map (of countries or cities, for example) can be colored using at most four colors, such that no adjacent territories share the same color.*

4.5 Towers of Hanoi

The puzzle of the 'Towers of Hanoi' consists of three rods, on which a number of discs of different sizes are slided. The aim of the puzzle is to reconfigure the discs so that they all reside on one of the other rods, by subsequently moving a single disc from the top of one rod to another, but where it is not allowed to place a larger disc above a smaller disc.



In the picture above, we have five discs. Can you solve the puzzle?

After some puzzling, you might succeed. But often, the strategy by which you did, is forgotten afterwards; moreover, we would like to solve the problem in general, for say 7 discs, or 8 discs, as well.

The important concept here is *generalizing* the problem: can we solve the puzzle for any number of discs? Note that the problem is very similar if we would only have four discs, and that the four disc problem, in turn, is very similar to the three disc problem.

Suppose we have already figured out how to solve the problem for four discs. Then, disregarding a fifth disc below all others on the initial rod, we can first transfer the four top discs to the second rod. Now, the fifth, large, disc, is cleared free on the first rod, and thus we move it to the third rod. Finally, we transfer the four discs unto it on the third rod, which we were able to do by assumption, resulting in all five discs now residing in order on the third rod, and we have solved the problem for five discs.

So, as you see, we can subsequently solve the problem for an extra disc, if we have already solved it for some number of discs.

Remark 4.21 This means that *if* we can solve the problem for a single disc, we can solve it for any number of discs, including the case of five discs, but also, say, ten or fifteen. And of course, the single disc problem is trivial: you can just move the disc freely.

Definition 4.22 The method we just demonstrated of solving a mathematical problem, is the method of *recursion*. Using recursion, one tackles a problem by dividing it into simpler problems, which are basically the same, ensuring that the solution of the larger problem then follows from those of the smaller problems. And of course, you should not forget to manually solve the smallest problem as well.

Now consider this question: given our strategy to solving the puzzle of Hanoi as described above, how many individual steps are needed? Call this number a_n, if we are dealing with the version with n discs. Of course, $a_1 = 1$, and $a_2 = 3$ is easy to see as well. But what about a_5? It can be hard to see this at once. But again, recursion comes to the aid: in order to solve Hanoi with five discs, we would first solve it for four discs, then move the largest one to the third rod, and then use the four disc solution once more. In a formula, that would translate to: $a_5 = a_4 + 1 + a_4 = 2a_4 + 1$. Of course, this *recursive formula* is not specific for a_5, it holds in general: $a_{n+1} = 2a_n + 1$, for the numbers of discs $n \geq 2$ of course. So, we can compute $a_3 = 2 \cdot a_2 + 1 = 2 \cdot 3 + 1 = 7$, and $a_4 = 2a_3 + 1 = 15$, and $a_5 = 2a_4 + 1 = 31$, etc... This gives us a neat sequence: $1, 3, 7, 15, 31, 63, 127, \ldots$

Exercise 4.M The sequence a_n is recursively defined by:

$$\begin{aligned} a_0 &= 3 \\ a_{n+1} &= a_n^2 - 2a_n \text{ for } n \geq 0 \end{aligned}$$

Use this definition to compute the value of a_5.

Exercise 4.N The sequence b_n is recursively defined by:

$$\begin{aligned} b_0 &= 4 \\ b_{n+1} &= b_n^2 - 2b_n \text{ for } n \geq 0 \end{aligned}$$

Use this definition to compute the value of b_5.

Exercise 4.O Consider the sequence c_n for $n \geq 0$, given by the values:

$$1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349, \ldots$$

Give a recursive definition for c_n.

4.6 Programming by recursion

Recursion is not only useful for doing mathematics, it is also an invaluable programming technique.

Suppose you have a little library in your programming language, that has an addition operation, although it doesn't yet support multiplication. How would you define multiplication? Easy, with recursion:

$$\begin{aligned} n * 1 &:= n \\ n * (m + 1) &:= n * m + n \end{aligned}$$

And once we have multiplication, we can also define exponentiation:

$$\begin{aligned} n^1 &:= n \\ n^{(m+1)} &:= n^m * n \end{aligned}$$

Both cases display how recursion allows you to define the operation (or, solve the problem) in terms of smaller cases, which are combined to form the new case.

```
mult := proc(n,m)
  if m=1
    then n
    else mult(n,m-1)+n
  end if
end proc;
```

```
pow:=proc(n,m)
  if m=1
    then n
    else mult(pow(n,m-1),n)
  end if
end proc;
```

Figure 4.2: Recursive `Maple` procedures

Example 4.23 Although we talk of programming, the formulas above might not resemble actual computer code enough to bring home the point. Therefore, we added the following two pieces of `Maple` code in Figure 4.2 for illustration purposes—and of course most other languages would allow a similar definition. `Maple`, by the way, is a so-called computer algebra system, allowing for all kinds of other interesting calculations. If you are interested, you can use `Maple` on `lilo.science.ru.nl`.

Exercise 4.P The `Maple` programs in Example 4.23 are not very robust. If we provide for m a value which is not a natural number, `Maple` will go into an infinite loop. Modify the programs in such a way that if both m and n are integers, `Maple` gives the correct result.

Remark 4.24 In exercises and exams, we will sometimes ask for you to give a program using recursion. If we do so, so-called *pseudocode* will suffice: you need not actually know or adhere to a specific programming language. All is OK as long as it is clear how the program would work.

4.7 Binary trees

We come by another kind of recursive programming when studying what are called binary trees. An example of such a tree can be seen in figure 4.3.

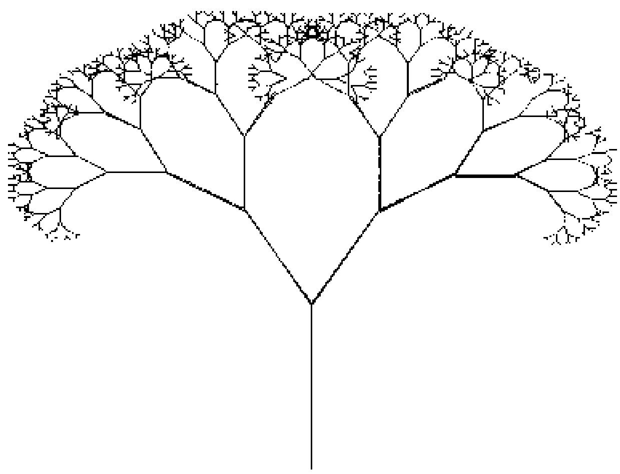


Figure 4.3: A binary tree

Although this tree might seem daunting, it was drawn with only a simple recursive procedure. The main insight here, is that any larger binary tree is composed of a stem, out of which two smaller binary trees grow: one going left, and one going right, both drawn slightly smaller, and of course at an angle.

So, a recursive recipe (or recursive function, or recursive procedure, or recursive program) is easily provided:

1. draw a stem
2. draw the left (sub)tree
3. draw the right (sub)tree

More precisely:

$$f(n,x,y,\alpha,\ell) = \begin{cases} \text{do nothing} & \text{if } n = 0 \\ \text{1. } \mathit{draw\,stem}(\text{position } \langle x,y\rangle, \text{angle } \alpha, \text{length } \ell) & \\ \text{2. } f(n-1, \langle x_1,y_1\rangle, \alpha-30, \ell/2) & \text{if } n > 0 \\ \text{3. } f(n-1, \langle x_2,y_2\rangle, \alpha+30, \ell/2) & \end{cases}$$

As input to the program are given: the height of the tree n, the starting position from which the tree should grow $\langle x, y\rangle$, the angle at which it should be drawn α, and the length its stem should have ℓ. The coordinates $\langle x_1, y_1\rangle$ then denote the endpoint of its stem, and can be calculated from α and ℓ.

The height of the tree, n, is the most important input to this recursive procedure, for our purposes of explaining recursion. So in the list below we will omit the program's other variables.

The task of drawing a tree of height n is reduced to the task of drawing two trees of height $n - 1$, and hence this program indeed uses recursion. This can be clearly seen when taking a look at the steps that the computer will consecutively take to execute this program:

```
[1] draw tree: f(3)
[1.1] draw stem
[1.2] draw left tree: f(2)
[1.2.1] draw stem
[1.2.2] draw left tree: f(1)
[1.2.2.1] draw stem
[1.2.2.2] draw left tree: f(0)
[1.2.2.2.1] do nothing
[1.2.2.3] draw right tree: f(0)
[1.2.2.3.1] do nothing
[1.2.3] draw right tree: f(1)
[1.2.3.1] draw stem
[1.2.3.2] draw left tree: f(0)
[1.2.3.2.1] do nothing
[1.2.3.3] draw right tree: f(0)
[1.2.3.3.1] do nothing
```

```
[1.3] draw right tree: f(2)
[1.3.1] draw stem
[1.3.2] draw left tree: f(1)
[1.3.2.1] draw stem
[1.3.2.2] draw left tree: f(0)
[1.3.2.2.1] do nothing
[1.3.2.3] draw right tree: f(0)
[1.3.2.3.1] do nothing
[1.3.3] draw right tree: f(1)
[1.3.3.1] draw stem
[1.3.3.2] draw left tree: f(0)
[1.3.3.2.1] do nothing
[1.3.3.3] draw right tree: f(0)
[1.3.3.3.1] do nothing
```

Note how a *recursive definition* is often very simple, though the *execution* of a recursive function can be quite complicated: quite a lot of book-keeping can be required to keep track of which sub-problem is being solved at any given point in time. Typical work for computers!

4.8 Induction

Let's return back to the sequence we encountered earlier, the sequence of the number of steps needed to solve the puzzle of Hanoi with n discs. This sequence was given by the recursive formula $a_{n+1} = 2a_n + 1$, where a_n stands for the number of steps needed. Of course computing a_n for a high n, for example a_{38}, can now be a bit tedious, because we have to first compute all previous a_n's. It would be more convenient if we had a direct formula to compute a_{38}.

Taking another look at the sequence: $1, 3, 7, 15, 31, 63, 127, \dots$, one might notice that it looks very much like the sequence of powers of two: $2, 4, 8, 16, 32, 64, 128, \dots$ The numbers of the first sequence seem to always be 1 less than those of the powers of two. But how do we know this for sure, and know that it isn't just a coincidence of the first part of the sequences? Suppose for now that $a_{37} = 2^{37} - 1$ would be indeed the case. Then we also have:

$$a_{38} = 2 \cdot a_{37} + 1 = 2 \cdot \left(2^{37} - 1\right) + 1 = 2^{38} - 2 + 1 = 2^{38} - 1.$$

So, if it holds for the 37th element, then also for the 38th, and so on. Of course there is nothing specific about it being exactly the 37th going on above, so we have the general truth that if $a_n = 2^n - 1$, then also $a_{n+1} = 2^{n+1} - 1$. Furthermore, we already know that $a_1 = 2^1 - 1 = 1$. So then indeed, it holds for a_2, and then also for a_3, and for a_4, and so on, and in particular also for a_{38}. We have now proven that indeed for all n, we have $a_n = a^n - 1$, and this is what we call a *direct formula* for a_n, because it doesn't depend on any previous values.

The method of proof demonstrated above is what we call *induction*. Induction, as a proof technique, is very similar to recursion, a way to define recursive formulas.

Definition 4.25 Induction can be used to prove that a certain *predicate* $P(n)$ holds for all natural numbers $(0, 1, 2, 3, \dots)$. Such a *proof by induction* is given by:

1. A proof of $P(0)$. (The *base case*.)

2. A proof that: if $P(k)$ holds for some $k \in \mathbb{N}$, where k is greater than or equal to the base case, then also $P(k+1)$. (The *induction step.*)

Giving these two is then enough to show that $P(n)$ holds for all n. When proving $P(k+1)$, (in 2.), the assumption that $P(k)$ holds already is called the *induction hypothesis* (IH).

To see how, for example, $P(37)$ then follows from such a proof by induction, note first that $P(0)$ holds, and thus (by 2.) also $P(1)$, and thus (again by 2.) also $P(2)$, etc, until we arrive at the truth of $P(37)$.

In our example above, the predicate $P(n)$ was the statement that $a_n = 2^n - 1$.

Remark 4.26 A recurring question is whether one should start with $P(0)$ or with $P(1)$. The answer to this question depends on the situation. If you want to prove something about *all* natural numbers, then you should of course start with $P(0)$. But if you only want to prove something about all natural numbers greater than, say, 7, then of course you may start with $P(8)$. It may sometimes even be necessary to prove the first number of cases separately, because the regularity only arises after that. Your induction step proof might then be for, say, $n \geq 5$.

Remark 4.27 In a way, induction can be seen as the opposite of recursion. The emphasis of induction lies on consecutively proving larger cases, while recursion is used to break a problem down into smaller cases: the opposite of the first.

Example 4.28 How much does $1 + 2 + 3 + \cdots + 99$ add up to? You could calculate this manually, or use a calculator, but in both cases you would be sure to be busy for a while. Unless you are smart of course, and use recursion and induction. Suppose you define $s(n) := 1 + 2 + \cdots + n$, for $n \geq 1$. Then a recursive program for computing $s(n)$ would be $s(n+1) = s(n) + n + 1$.

After looking at the first number of results $s(1), s(2), s(3), \ldots$, you might start to suspect that generally $s(n)$ is given by $s(n) = (n^2 + n)/2$. To prove this direct formula for the value of $s(n)$, we use induction. In this example our predicate $P(n)$ is the statement that $s(n) = (n^2 + n)/2$.

Base Case Does $P(1)$ hold? Yes, because $s(1) = 1 = (1^2 + 1)/2$.

Induction Step Now suppose that we already know that $P(k)$ holds for some $k \in \mathbb{N}$ with $k \geq 1$, that is, $s(k) = (k^2 + k)/2$ (IH). Then do we have that $P(k+1)$ holds as well? Let's see what we can say about $s(k+1)$:

$$\begin{aligned} s(k+1) &= s(k) + k + 1 \text{ (by definition of } s(k+1)) \\ &= \frac{k^2 + k}{2} + k + 1 \text{ (IH)} \\ &= \frac{k^2 + k + 2k + 2}{2} \text{ (algebra)} \\ &= \frac{(k^2 + 2k + 1) + (k + 1)}{2} \text{ (algebra)} \\ &= \frac{(k+1)^2 + (k+1)}{2} \text{ (algebra)} \end{aligned}$$

So $P(k+1)$ indeed holds.

So indeed, it follows by induction that $s(n) = (n^2 + n)/2$ holds generally for $n \geq 1$.

The mathematician Gauss, by the way, was able to solve this problem in an easy way, without using induction. See exercise 4.U.

Example 4.29 In how many ways can we order the numbers 1, 2, 3, 4, 5, 6, 7, 8 and 9? It is cumbersome to write down all possibilities. But luckily, we don't need to. Let a_n denote the number of orderings of n elements. So, the question is what the value of a_9 is. At least we know where to start: a_1 is simply 1. Furthermore, $a_{n+1} = (n + 1) \cdot a_n$, because we would first choose an element to put at the front of the list, which is a choice of 1 in $n+1$, and then we order the remaining n elements, for which there are a_n possibilities.

So how, then, do we calculate a_9? Well, the definition we gave above is exactly that of the factorial: $a_n = n!$. Which means, we can use any standard scientific calculator.

Example 4.30 What is the value of the sum

$$1 + a + a^2 + a^3 + \cdots + a^n$$

given some $n \in \mathbb{N}$ and an arbitrary a? Let's first test it out on some small values:

	$n = 0$	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 10$	$n = 42$
$a = 0$	1	1	1	1	1	1	1
$a = 1$	1	2	3	4	5	11	43
$a = 3$	1	4	13	40	121	88573	164128483697268538813
$a = \frac{2}{3}$	1	$\frac{5}{3}$	$\frac{19}{9}$	$\frac{65}{27}$	$\frac{211}{81}$	$\frac{175099}{59049}$	$\frac{328256958598444055419}{109418989131512359209}$
$a = -1$	1	0	1	0	1	1	1
$a = -2$	1	-1	3	-5	11	683	2932031007403

An attentive student might remember the formula for calculating the sum of a geometric sequence:

$$1 + a + a^2 + a^3 + \cdots + a^n = \begin{cases} n + 1 & \text{if } a = 1 \\ \frac{a^{n+1}-1}{a-1} & \text{if } a \neq 1 \end{cases}$$

We will prove that the case where $a \neq 1$ holds, by induction. We start off by defining precisely what our predicate $P(n)$ is:

$$P(n) := \quad 1 + a + a^2 + a^3 + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1}$$

Base Case We want to prove the statement for all $n \in \mathbb{N}$, so we start with the base case of $P(0)$, which means that we have to prove that:

$$1 = \frac{a^{0+1} - 1}{a - 1}$$

Indeed this is the case, because $a^{0+1} - 1 = a^1 - 1 = a - 1$. So the numerator and the denominator of the fraction are equal, which implies that the fraction is indeed equal to 1. Recall that we have $a \neq 1$ by assumption, so we don't have to worry about dividing by zero.

Induction Step We may assume the induction hypothesis, $P(k)$ for some $k \in \mathbb{N}$ such that $k \geq 0$, and must now prove that $P(k+1)$ holds as well, that is:

$$1 + a + a^2 + a^3 + \cdots + a^k + a^{k+1} = \frac{a^{k+2} - 1}{a - 1}$$

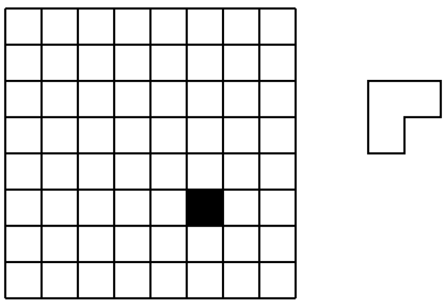
We do this by rewriting the left hand side of the equation until we see how our assumption $P(k)$ fits into it, then use the induction hypothesis, and then rewrite a bit more until we have the right hand side of the equation.

$$\begin{aligned}
& 1 + a + a^2 + a^3 + \cdots + a^k + a^{k+1} \\
= \; & (1 + a + a^2 + a^3 + \cdots + a^k) + a^{k+1} \text{ (reveal left hand side of } P(k)) \\
= \; & \frac{a^{k+1} - 1}{a - 1} + a^{k+1} \text{ (apply IH)} \\
= \; & \frac{a^{k+1} - 1}{a - 1} + a^{k+1} \cdot \frac{a - 1}{a - 1} \text{ (making the denominators equal)} \\
= \; & \frac{a^{k+1} - 1}{a - 1} + \frac{a^{k+2} - a^{k+1}}{a - 1} \text{ (multiplying of fractions)} \\
= \; & \frac{a^{k+1} - 1 + a^{k+2} - a^{k+1}}{a - 1} \text{ (adding fractions)} \\
= \; & \frac{a^{k+2} - 1}{a - 1} \text{ (simplifying)}
\end{aligned}$$

And now we have ended up with the right hand side of the equation of $P(k+1)$.

So now it follows by induction that $P(n)$ holds for all $n \in \mathbb{N}$ and $a \neq 1$.

Example 4.31 So far we have only concerned ourselves with theorems of which the proofs are based on, and rely upon skills of, algebra. This example demonstrates that other options are possible as well. Consider this 8×8 board, of which one of its squares has been removed. Can the rest of the board be tiled completely with only tiles of the particular shape pictured next to the board?



A bit of puzzling might convince you that this is indeed possible. But the question is pressing, whether this is coincidental to the particular square removed from the board, or whether it may be tiled whichever square is removed. The latter turns out to generally be the case, for which we will now give an inductive proof.

First, we define our predicate $P(n)$:

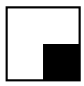
$P(n)$:= A $2^n \times 2^n$ board of which a single square has been removed, can always be tiled (with tiles of the shape pictured above), regardless of which square was removed.

Now we can make the theorem clear:

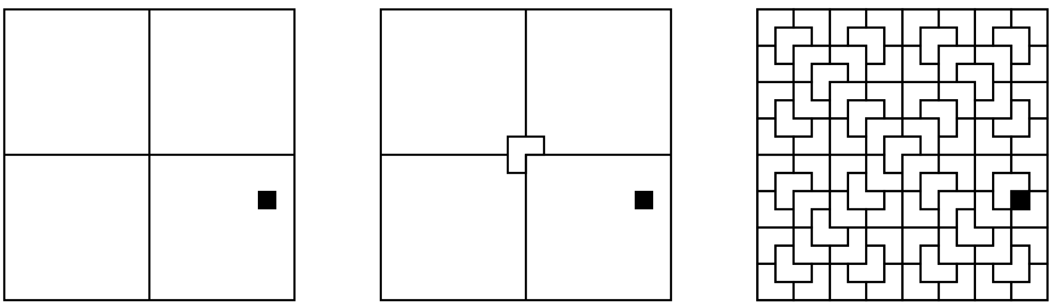
Theorem *The predicate $P(n)$ holds for all $n \geq 1$.*

Proof by induction on n.

Base Case The base case is $P(1)$, because the theorem explicitly tells us that $n \geq 1$. Which means, we must prove that a 2×2 board can be tiled, if any single square has been removed. This is of course easily seen to be true. Whichever square is removed doesn't matter, because we can simply rotate the tile. The picture below illustrates the base case tiling.



Induction Step Let $k \in \mathbb{N}$ and $k \geq 1$. We may now assume that $P(k)$ holds (our induction hypothesis), which means that any $2^k \times 2^k$ board with one square removed can be tiled with the special tile. And we now have to prove that $P(k+1)$ holds, which means that any $2^{k+1} \times 2^{k+1}$ board with one square removed can be tiled. Now let us consider an arbitrary $2^{k+1} \times 2^{k+1}$ board, of which one square has been removed. Note that such a board can always be subdivided into four $2^k \times 2^k$ board, of which one of them has a square removed. By rotating our initial board, we can state without loss of generality, that the square will have been removed in the lower right sub-board. The leftmost illustration depicts this situation.



Now, crucially, we place the first tile in the center of our board, as depicted in the middle illustration. Note that now each sub-board can be regarded as having 'removed' a single square, except for the lower right board—but it already had a square removed. Now, we can simply apply our induction hypothesis to all four sub-boards, thereby immediately yielding a tiling of the full board as well. And so we have proven that $P(k+1)$ holds.

So with induction it follows that $P(n)$ holds for all $n \geq 1$.

Note that in this proof, we have not only proven that $P(n)$ holds generally, but we have actually specified a definite method for tiling. The rightmost illustration depicts the result of this tiling method for a 16×16 board.

Exercise 4.Q The inventor of the chessboard was told by the king of Persia, that he would be rewarded any object of choice. The inventor chose the following: 1 grain of rice on the first field of the chessboard, 2 grains of rice on the second, 4 on the third, and so on, doubling the number of grains for each successive field. The king thought the inventor to be very humble. Now the question is: how many grains of rice did the inventor's choice amount to? Let's formulate it formally: he asked for

$$1 + 2 + 2^2 + 2^3 + 2^4 + \cdots + 2^{63}$$

grains of rice. Can you find a direct formula for the result of this sum? And can you then prove it by induction? [*Hint:* First explicitly count the result for some initial parts of the sequence $1, 2, 2^2, 2^3, 2^4, \ldots,$ and find out if you can recognize some emerging pattern in 1, $1+2$, $1+2+2^2$,.... What is your guess?]

Exercise 4.R Consider the sequence a_n defined by:

$$\begin{cases} a_0 & = & 0 \\ a_{n+1} & = & a_n + 2n + 1 \quad \text{for } n \geq 0 \end{cases}$$

Prove that for all $n \in \mathbb{N}$, $a_n = n^2$.

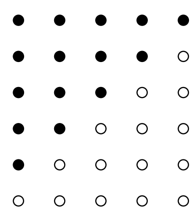
Exercise 4.S In exercise 4.C, we proved that for all $n \geq 1$ the complete graph K_n has exactly $\frac{1}{2}n(n-1)$ edges. Now, try to prove this by induction on n.

Exercise 4.T Prove by induction, that for all $n \in \mathbb{N}$, $2^n \geq n$.

Exercise 4.U So we have seen the formula for the sum of the following sequence:

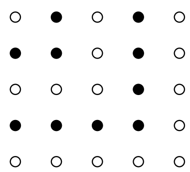
$$1 + 2 + 3 + 4 + 5 + \cdots + n = \frac{(n+1)n}{2}.$$

What is the connection with this picture?



Exercise 4.V How many distinct sequences of length n can we make, filled with the numbers 1 through 5? This too, can be solved easily with recursion. Let a_n denote the number of distinct sequences of length n. The simplest sequence, of length 1, can of course be made in 5 ways, so $a_1 = 5$. A sequence of length $n+1$ can be made by first taking a sequence of length n, and then appending a new element after it. So, $a_{n+1} := 5 \cdot a_n$. Now, recall the definition of raising to a power. Prove by induction, that for all $n \geq 1$, $a_n = 5^n$.

Exercise 4.W Prove, by induction, that $1+3+5+7+\cdots+(2n-1)=n^2$ for $n \geq 1$. What is the connection with the picture below?



4.9 Pascal's Triangle

Before we discuss Pascal's Triangle, we repeat some combinatorics by an example.

Example 4.32 We will be taking a look at the Dutch Lotto[7]. In this game of chance, there is a 'vase', filled with 45 sequentially numbered balls (1 up to 45). A notary then pulls six balls from this vase, at random (and without putting them back afterwards). The numbers on these six balls are then compared to the numbers that the players have chosen beforehand. The more numbers coincide, the higher the rewards are. But how many distinct outcomes are there to such a draw? Well, for the first ball to be drawn, there are of course 45 possibilities. And for the second, 44, and for the third, 43, etc. So that would lead us to the conclusion that there are

$$45 \cdot 44 \cdot 43 \cdot 42 \cdot 41 \cdot 40 = \frac{45 \cdot 44 \cdot 43 \cdot 42 \cdot 41 \cdot 40 \cdot 39 \cdot 38 \cdot \cdots \cdot 1}{39 \cdot 38 \cdot \cdots \cdot 1} = \frac{45!}{39!} = \frac{45!}{(45-6)!}$$

ways to draw six consecutive balls. But note that the ordering of these balls, after they have been drawn, doesn't make a difference. So for example, the draw $(5, 10, 2, 29, 6)$ is equal, with respect to this game, to the draw $(10, 2, 5, 29, 6)$, but these two have both been counted once in the calculation above. So we have to compensate for all draws that we counted 'double'. Each ordering of six of these balls leads to the same outcome. We know that the number of orderings of six of these balls is 6!, as we have seen earlier, so to counter this problem, we must divide the result of our calculation above by 6!, giving:

$$\frac{45!}{6!\,(45-6)!} = 8145060$$

This is then the actual number of possible draws.

Definition 4.33 Let n and k be natural numbers, with $k \leq n$. We then define the *binomial* $\binom{n}{k}$ as the number of ways in which k objects can be drawn from a set of n elements (as above).

We now continue by looking at the grid coordinates (n, k) of a grid that is skewed in such a way that we have $(0, 0)$ at the top:

$$\begin{array}{ccccccccccccccc}
 & & & & & & & (0,0) & & & & & & & \\
 & & & & & & (1,0) & & (1,1) & & & & & & \\
 & & & & & (2,0) & & (2,1) & & (2,2) & & & & & \\
 & & & & (3,0) & & (3,1) & & (3,2) & & (3,3) & & & & \\
 & & & (4,0) & & (4,1) & & (4,2) & & (4,3) & & (4,4) & & & \\
 & & (5,0) & & (5,1) & & (5,2) & & (5,3) & & (5,4) & & (5,5) & & \\
 & (6,0) & & (6,1) & & (6,2) & & (6,3) & & (6,4) & & (6,5) & & (6,6) & \\
\dots & & \dots & & \dots & & \dots & & \dots & & \dots & & \dots & & \dots
\end{array}$$

[7] Although we will disregard 'Superzaterdag' and the 'Jackpot' for convenience.

We will now assign values to these coordinates in different ways, and see how these different assignments relate.

Definition 4.34 Assign ones to the left and right borders of the triangle, and then fill in the rest of the triangle by consecutively adding up the two values directly above a new one. Put more precisely: all coordinates $(n, 0)$ and (n, n) get the value 1, and then each (n, k) is given by adding up the values of $(n-1, k)$ and $(n-1, k-1)$. *The first version of Pascal's Triangle* ($P\triangle_1$) is thus as follows[8]:

$$\begin{array}{ccccccccccccccc}
 & & & & & & & 1 & & & & & & & \\
 & & & & & & 1 & & 1 & & & & & & \\
 & & & & & 1 & & 2 & & 1 & & & & & \\
 & & & & 1 & & 3 & & 3 & & 1 & & & & \\
 & & & 1 & & 4 & & 6 & & 4 & & 1 & & & \\
 & & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & & \\
 & 1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1 & \\
1 & & 7 & & 21 & & 35 & & 35 & & 21 & & 7 & & 1
\end{array}$$

Definition 4.35 Assign to each coordinate (n, k), the value $\binom{n}{k}$. This then gives us *the second version of Pascal's Triangle* ($P\triangle_2$).

Exercise 4.X Prove that all numbers on the outer edge of $P\triangle_2$ are 1.

We claim, for $k < n$, that $\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k}$. If we draw a subset A containing $k + 1$ elements from the set $1, 2, 3, 4, \ldots n + 1$, then there are two possibilities: either $n + 1 \in A$, or $n + 1 \notin A$. In the first case, $A \setminus \{n + 1\}$ is a subset containing k elements, drawn from $\{1, 2, 3, 4, \ldots, n\}$. In the second case, A is a subset containing $k + 1$ elements, drawn from $\{1, 2, 3, 4, \ldots, n\}$. So indeed $\binom{n+1}{k+1} = \binom{n}{k+1} + \binom{n}{k}$. Or, put differently: $P\triangle_2$ also follows the principle that the value of each coordinate can be found by adding the values of the two coordinates that lie above it. The result of which is that the two triangles $P\triangle_1$ and $P\triangle_2$ are exactly the same.

Definition 4.36 Assign to each coordinate (n, k), the number that denotes the amount of possible roads leading from $(0, 0)$ to (n, k), where each subsequent step should be one step down, either diagonally to the left, or to the right. This gives us *the third version of Pascal's Triangle* ($P\triangle_3$).

One can then observe that:

- The border of $P\triangle_3$ is filled with ones.
- The triangle $P\triangle_3$ also follows the 'principle of adding' described above for the other two versions of Pascal's Triangle.

Result: $P\triangle_3$ is yet again exactly the same triangle as $P\triangle_1$ and $P\triangle_2$.

[8] Of course this triangle carries on infinitely, we just abbreviated it to the first eight rows.

Definition 4.37 Assign to each coordinate (n, k), the coefficient that x^k takes on in the polynomial $(1 + x)^n$. Example: the coordinate $(6, 2)$ gets the value 15, because $(1 + x)^6 = 1 + 6x + 15x^2 + 20x^3 + 15x^4 + 6x^5 + x^6$ and thus we see that the coefficient of x^2 is 15. This gives us *the fourth version of Pascal's Triangle* (P$\triangle_4$).

Exercise 4.Y Show that P$\triangle_4$, too, is equal to P$\triangle_1$.

We have seen that the four ways of presenting Pascal's Triangle all lead to the same triangle of numbers. Because we have seen that all versions coincide, we may speak of *the* Triangle of Pascal, without having to refer to any one of its specific instantiations.

Exercise 4.Z Demonstrate how the triangle of Pascal can be used to figure out how many distinct ways there are, to pick four objects out of a collection of six. What is the notation for the corresponding binomial?

Definition 4.38 The following equality is known under the name of *Newton's Binomial Theorem*.

$$(1 + x)^n = \binom{n}{0} + \binom{n}{1}x + \binom{n}{2}x^2 + \cdots + \binom{n}{n}x^n$$

Relating this theorem to what we have seen above, it actually simply is the statement: that P$\triangle_2$ is equal to P$\triangle_4$. The more general version of the theorem,

$$(y + z)^n = \binom{n}{0}y^n + \binom{n}{1}y^{n-1}z + \binom{n}{2}y^{n-2}z^2 + \cdots + \binom{n}{n}z^n$$

follows simply from the fact that $(y + z)^n = y^n(1 + (\frac{z}{y})^n)$ when $y \neq 0$. The case in which $y = 0$ is trivial.

4.10 Important concepts

5 Modal logic

Although propositional logic and predicate logic are the most well-known, there are many other 'logics'. Different logics vary in their ability to represent nuances of statements. They differ in which operators are used to write down formulas, and sometimes they differ in the way interpretations are given to those operators as well.

Some examples are: Floyd-Hoare logic (in which you can describe accurately, the behavior of programs), intuitionistic logic (which expresses exactly those things that can be computed by computers), and paraconsistent logic (in which it is possible to retract the truth statements as new information becomes available).

A special class of logics are the so-called *modal logics*, which are used to describe *to which extent* we should believe certain statements to be true. Modal logics are the subject matter of this chapter.

Within the class of modal logics, there is the important sub-class of *temporal logics*. These logics are able to express *at which points in time* statements are true. At the end of this chapter, we present a temporal logic in more detail. For further reading on modal logic, we recommend the books [2] and [3].

5.1 Necessity and possibility

Traditionally, modal logics express the notion of *necessity*. Take the following sentence:

It is raining.

This sentence could be true or not true, but it is not *necessarily* true. It does not always rain, and even when it does, the reason is not simply that it *must* be raining: if the weather was different it may not have rained. The following sentence however, is necessarily true:

If it rains, water falls from the sky.

The necessity of the truth of this sentence follows from the meaning of the word 'raining', which is exactly, that water falls from the sky.

This difference of necessity does not find expression in propositional logic. If we use the following dictionary:

R	it is raining
W	water is falling from the sky

then the formalizations of the sentences are:

$$R$$

and

$$R \rightarrow W$$

in which the necessity is not clearly expressed. To this purpose, modal logic introduces a new operator $\Box$, that should be read as meaning 'necessarily.' The sentence:

It is necessarily true that raining implies that water falls from the sky.

is then written formally as:

$$\Box(R \to W)$$

You might think the reverse is true as well, that water falling from the sky implies that it is raining, but this is not true. For instance, turning on a garden hose, or standing under a waterfall, are situations in which water does fall from the sky, without it raining. Therefore, we have:

It is not necessarily true that water falling from the sky implies that it is raining.

which, written as a formula in modal logic, is:

$$\neg\Box(W \to R)$$

Of course it is still possible for

$$W \to R$$

to hold in certain situations. For example, if it is raining, then the right-hand side of the implication is true, and hence (as we can see in the truth table for implication) the whole formula is true.

Besides necessity, modal logic also has a notation for *possibility*. This is written as: $\Diamond$. The true sentence:

It is possible that it is raining.

is then expressed in modal logic by the formula:

$$\Diamond R$$

If you think about it a bit, you will find that the above statement means the same as:

It it not necessary for it not to be raining.

Which is in turn expressed as:

$$\neg\Box\neg R$$

Exercise 5.A So we see that the formula $\Diamond R$ means the same as $\neg\Box\neg R$. Try to find a formula without the symbol '$\Box$' that means the same as $\Box R$. Then, translate both $\Box R$ and the formula you found to ordinary English.

Definition 5.1 Any statement U will either be:

- necessary
- impossible
- contingent

A statement is said to be *contingent* when it is not necessarily true, nor impossibly true. That is, its truth is somewhere between necessarily true and necessarily false.

Exercise 5.B In modal logic the statement 'U is necessarily true' is symbolically represented as $\Box U$, and 'U is impossible' as $\Box\neg U$, or alternatively, $\neg\Diamond U$. Give two different formulas in modal logic which express the statement that 'U is contingent.'

Exercise 5.C Use the following dictionary

M	I have money
B	I am buying something

to translate these English sentences to formulas of modal logic:

(i) *It is possible for me to buy something without having money.*

(ii) *It is necessarily true, if I am buying something, for me to have money.*

(iii) *It is possible that if I buy something I don't have any money.*

Which of *these* sentences seem true? Explain why.

5.2 Syntax

We will now give a context-free grammar of the formulas of modal logic. (Modal predicate logic also exists, but we will not treat it here.) This grammar is exactly the same as the one we have given for propositional logic before (see remark 1.4), except that the two modal operators are added:

$$S \;\rightarrow\; a \mid \neg S \mid (S \wedge S) \mid (S \vee S) \mid (S \rightarrow S) \mid (S \leftrightarrow S) \mid \Box S \mid \Diamond S$$

Convention 5.2 Just as with propositional logic, we allow ourselves to leave out unnecessary parentheses. In doing this, we stick to the convention that the modal operators have the same operator precedence as negation '$\neg$', which means that they bind stronger than the binary operators.

This means that the formula

$$\Diamond a \wedge b \rightarrow \Diamond(\Box a \vee \neg c)$$

should be read as

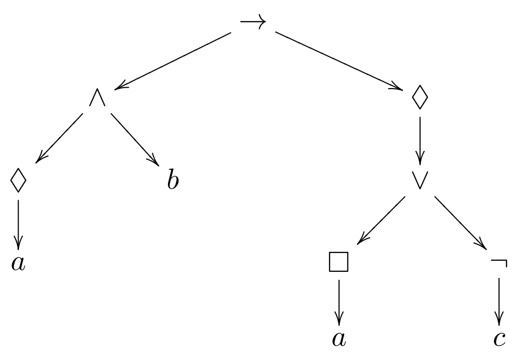
$$(((\Diamond a) \wedge b) \rightarrow (\Diamond((\Box a) \vee (\neg c))))$$

Because the grammar does not add parentheses to the unary operators, the following formula is the 'official' form of the ones above:

$$((\Diamond a \wedge b) \rightarrow \Diamond(\Box a \vee \neg c))$$

The first two should be seen as different (and possibly clearer) ways to represent this 17-symbol-long formula.

We can also draw the structure of the formula in a tree:



In the tree, the atomic propositions are the leaves, and the logical operators the nodes.

Exercise 5.D For each of the formulas below, give the 'official' form according to the grammar of modal logic, and also draw a tree according to its structure.

(i) $\Box(\Box a)$
(ii) $\Box a \to a$
(iii) $(\Box a) \to a$
(iv) $\Box(a \to a)$
(v) $\neg a \to \neg\Box a$
(vi) $\Diamond a \to \Box\Diamond a$
(vii) $\Box a \to \Diamond a$
(viii) $\Diamond a \wedge b \to \Diamond\Box a \vee \neg c$

5.3 Modalities

Up until now the operator $\Box$ has had the fixed meaning of 'necessity' and the operator $\Diamond$ that of 'possibility'. This is the traditional reading, but only one of a variety of possible interpretations of these symbols. Depending on what we interpret the symbols to mean, we get different modal logics:

logic	modality	$\Box f$	$\Diamond f$
modal logic	necessity	f is necessarily true	f is possibly true
epistemic logic	knowledge	I know that f	f doesn't contradict my knowledge
doxastic logic	belief	I believe that f	f doesn't contradict my beliefs
temporal logic	time	f is always true	f is sometimes true
deontic logic	obligation	f ought to be	f is permissible
program-logic		after any execution, f holds	after some execution, f holds

If not specified otherwise, 'modal logic' often refers to the traditional reading of 'necessity'. At the same time, it is also used to refer to the whole class of modal logics.

Exercise 5.E Use the following dictionary:

R	I am ready
H	I am going home

For each of the logics listed in the table above (except program-logic), give an English translation of the formula:

$$\neg R \to \neg\Diamond H$$

Each of the logics enumerated in the table, have yet again their own variations. For instance, just as there are a whole set of modal logics, so too, are there many different temporal logics. The meaning 'true at all times' can be changed to 'always true from this point on' or maybe 'always true after this point in time.' Or else, it could take into account that it is not yet clear what is going to happen. (This is called 'branching time'.)

5.4 Axioms

Depending on what the modal operators are meant to describe, you might want them to have different properties. For instance, whether the formula

$$\Box f \to f$$

should always be true. (Note: this should be read as '$(\Box f) \to f$' and not as '$\Box(f \to f)$', because the $\Box$ binds stronger than the $\to$.) In the traditional interpretation of modal logic this indeed always holds, because it translates to:

If f is necessarily true, then f holds.

And of course, necessary truth implies truth. But you wouldn't want it to hold in doxastic logic, because then it would be an encoding of the following English sentence:

If I believe f to be true, then f holds.

Although we all know that belief does not imply truth, because a belief may be mistaken. If we turn to the logic of knowledge, epistemic logic, we want it to be true again:

If I know that f holds, then f holds.

Because *knowing* something to be true implies in particular that it must indeed be true (in addition to the fact that it is known). Hence, we see that the truth of a particular property depends on the chosen interpretation of $\Box$.

In modal logic, there are many such principles (properties), called *axiom schemes* of modal logic. Some of the most important ones are listed below:

name	axiom scheme	property
K	$\Box(f \to g) \to (\Box f \to \Box g)$	distributive
T	$\Box f \to f$	reflexive
B	$f \to \Box\Diamond f$	symmetric
4	$\Box f \to \Box\Box f$	transitive
5	$\Diamond f \to \Box\Diamond f$	Euclidean
D	$\Box f \to \Diamond f$	serial

The first column presents the letter or number usually used to denote the axiom scheme, and the third column the property that is associated with the axiom scheme.

Exercise 5.F Construct a matrix that indicates which axiom schemes holds in which logic. Let the rows denote the logics listed on page 68, and the columns the axiom schemes. The matrix will then have a total of 36 cells. Then write a '+' or '−' in each cell, according to whether you think the axiom scheme is always true in the logic, or you think that it need not always hold.

It is ultimately the choice of axiom schemes to be included that distinct a particular modal logic. Regardless of which *interpretation* is given to the operators $\Box$ and $\Diamond$, the choice of axiom schemes determines how the symbols may be reasoned with (which, in turn, determines whether a particular interpretation is sensible.)

In this perspective, the most important axiomatic systems of modal logic are:

axiomatic system	axioms included
K	K
D	K + D
T	K + T
S4	K + T + 4
S5	K + T + B + 4 + 5

As you can see, the K-axiom is included in all of these logics. The scheme forms a basis on which all others are built, and therefore the modal logic **K** is the *weakest* of the modal logics.

5.5 Possible worlds and Kripke semantics

In section 5.1 we decided whether certain statements were true or not, by intuition. Now, we will formalize this. When we say that 'it is raining' is not necessarily true, we do so because we can imagine a world in which it is not raining. Such a world is called a *possible world*. If in all possible worlds, it is raining, then we must conclude that it is necessarily true for it to rain. This way of deciding the truth of formulas of modal logic, is called the possible world semantics. (In logic, *syntax* defines the rules by which well-formed formulas are constructed, while *semantics* describe the meaning of formulas. Thus, syntax is about shape, and semantics about meaning and truth.)

You may think that in the possible world semantics, the formula $\Box f$ describes the situation that the formula f holds in all worlds, and the formula $\Diamond f$ the situation that the formula f holds in at least one world. But this has undesired consequences. For instance, the formula

$$\Box f \to f$$

would then always be true (because if it holds in *all* worlds, it also holds in our current actual world), whereas we saw in the previous section that this formula is not supposed to hold in general (remember the modal logic of beliefs). So, we have to construct something a bit more complicated.

The necessary additional ingredient, is that we should specify which worlds are 'visible' from within any given world. This means that for every world, we will specify a set of *accessible* worlds, which are 'visible from its vantage point.' The meaning of $\Box f$ will then be that it is true in a world x, exactly in the case that f is true in all *accessible* worlds of x. Similarly, $\Diamond f$ is true in a world x if and only if there is at least one world, accessible to x, in which f is true. This is formalized in definition 5.6, later on.

To understand why this notion of accessibility is actually quite reasonable, it is useful to consider the case of temporal logic. In a possible world semantics for temporal logic, the worlds accessible to a certain world x would be that world x itself, in addition to all subsequent worlds (in time). Which means that the definition given above of $\Box f$ translates to: 'f is true from this point on.'

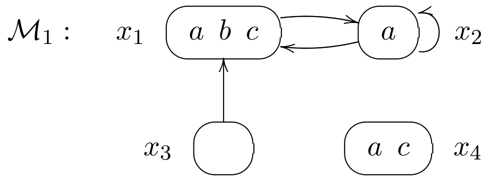
A collection of worlds, together with a relation of accessibility, is called a *Kripke model*, named after the American philosopher and logician Saul Kripke. We will now give a clean mathematical definition of these models.

Definition 5.3 A *Kripke model* $\mathcal{M} = \langle W, R, V\rangle$ consists of:

- a non-empty set W of *worlds*
- a function R such that for each world $x \in W$, the set $R(x) \subseteq W$ is the set of *accessible worlds* of x. The set $R(x)$ is also called the *successor* of world x. The function R can also be seen as a relation, which is then called the *accessibility relation*.
- a function V such that for each world $x \in W$, the set $V(x)$ is the set of *atomic propositions* that are true in world x.

We will draw Kripke models as directed graphs, where nodes denote worlds, and arrows between worlds denote accessibility. In the nodes, we write which atomic propositions hold in the corresponding world.

Example 5.4 Here is an example of a Kripke model, which we will name $\mathcal{M}_1$:



In this model there are four worlds $W = \{x_1, x_2, x_3, x_4\}$. The accessibility relation of this Kripke model is given by:

$$\begin{array}{rcl} R(x_1) & = & \{x_2\} \\ R(x_2) & = & \{x_1, x_2\} \\ R(x_3) & = & \{x_1\} \\ R(x_4) & = & \emptyset \end{array}$$

and the worlds make the following atomic propositions true:

$$\begin{array}{rcl} V(x_1) & = & \{a, b, c\} \\ V(x_2) & = & \{a\} \\ V(x_3) & = & \emptyset \\ V(x_4) & = & \{a, c\} \end{array}$$

E.g., the propositions a, b, and c are true in world x_1, but the proposition d is *not* true in world x_1.

We now formalize when formulas are to be held true in a given model $\mathcal{M}$ and world x. First, some notation:

Definition 5.5 The statement "the formula f of modal logic is true in a world x of a Kripke model $\mathcal{M}$" is denoted by

$$\mathcal{M}, x \Vdash f$$

When the model is clear from the context, this is sometimes shortened to:

$$x \Vdash f$$

And when $x \Vdash f$ does not hold, we write $x \nVdash f$, with $\Vdash$ struck out.

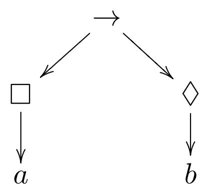
The truth of a formula is usually tied to a specific world, in which case it corresponds exactly to the truth definition of propositional logic as we have seen before. The only formulas in which more than one world is taken into consideration are those formed with the modal operators. This observation naturally leads to the following definition of the truth of a given formula f:

Definition 5.6 Consider a Kripke model $\mathcal{M} = \langle W, R, V \rangle$. Let $x \in W$, p be a propositional atom, and f and g formulas of modal logic. Then we define:

$$\begin{array}{rll} x \Vdash p & \text{iff} & p \in V(x) \\ x \Vdash \neg f & \text{iff} & x \nVdash f \\ x \Vdash f \wedge g & \text{iff} & x \Vdash f \text{ and } x \Vdash g \\ x \Vdash f \vee g & \text{iff} & x \Vdash f \text{ or } x \Vdash g \\ x \Vdash f \rightarrow g & \text{iff} & \text{if } x \Vdash f \text{ then } x \Vdash g \\ x \Vdash f \leftrightarrow g & \text{iff} & (x \Vdash f \text{ iff } x \Vdash g) \\ & & \\ x \Vdash \Box f & \text{iff} & \text{for all } y \in R(x)\text{, it holds that } y \Vdash f \\ x \Vdash \Diamond f & \text{iff} & \text{at least one } y \in R(x) \text{ exists, for which } y \Vdash f \end{array}$$

Remark 5.7 The *modal* character of course lies in the last two lines of this definition. The rest are exactly the same as for propositional logic.

Let us now check whether $\Box a \rightarrow \Diamond b$ is true in world x_1 of the Kripke model $\mathcal{M}_1$ we defined above. This formula has the structure:



Hence, we first ask ourselves whether $\Box a$ and $\Diamond b$ are true in x_1. For the first, we have to check *all* successors of x_1, which in this case is just x_2. And indeed, a is true in x_2, that is, $x_2 \Vdash a$, and so $\Box a$ is true in x_1, or, $x_1 \Vdash \Box a$. For $\Diamond b$ to be true in x_1, we must find a successor of x_1 in which b is true. But x_2 is the only successor of x_1, and in fact $x_2 \nVdash b$, so that $x_1 \nVdash \Diamond b$.

Now we turn to the truth table of implication. We know that $\Box a$ is true in x_1, and $\Diamond b$ is not true in x_1. And thus, $\Box a \rightarrow \Diamond b$ is not true in x_1. That is:

$$x_1 \nVdash \Box a \rightarrow \Diamond b$$

Check for yourself in which worlds of $\mathcal{M}_1$ the formula $\Box a$ holds. If you do this correctly, you will find that the formula in fact holds in all worlds of $\mathcal{M}_1$. We denote this by $\mathcal{M}_1 \models \Box a$

Definition 5.8 A formula f is said to be *true in a Kripke model* $\mathcal{M}$ if for all worlds x of $\mathcal{M}$ we have $\mathcal{M}, x \Vdash f$. This is denoted by:

$$\mathcal{M} \models f$$

Note the distinct meanings of $\models$ and $\Vdash$!

Remark 5.9 Note that in our example, in the world x_3, the formula a is *not* true, that is, $x_3 \nVdash a$. At the same time, we have seen the truth of $\mathcal{M}_1 \Vdash \Box a$. Hence we see how in a Kripke model, the truth of $\Box f$ does not imply that f is true in all worlds!

Exercise 5.G For each formula f listed below, check in which worlds x of our Kripke model in example 5.4 it holds, that is, for which x we have $x \Vdash f$. In addition, check whether f holds in the model, that is, check whether $\mathcal{M}_1 \models f$.

(i) a	(iii) $\Diamond a$	(v) $\Box a \rightarrow \Box\Box a$
(ii) $\Box a$	(iv) $\Box a \rightarrow a$	(vi) $a \rightarrow \Box\Diamond a$

Exercise 5.H (i) Can you find a Kripke model $\mathcal{M}_2$ for which $\mathcal{M}_2 \models a$ and at the same time $\mathcal{M}_2 \not\models \Box a$? If so, provide such a model; otherwise, explain why such a model cannot exist.

(ii) Provide a Kripke model $\mathcal{M}_3$ for which $\mathcal{M}_3 \models a$ although $\mathcal{M}_3 \not\models \Diamond a$. Or, if no such model can exist, explain why.

You might want to place restrictions on the structure of Kripke models. A number of such restrictions correspond to the axiom schemes we saw in the previous section. For instance, you might want to impose every world x to have an arrow from and to itself, that is, $x \in R(x)$. A Kripke model that obeys this restriction is called *reflexive*. And the set of reflexive Kripke models indeed corresponds with the axiom scheme T, that is, $\Box f \rightarrow f$. Or, you might want to impose that every world has at least one outgoing arrow. This is then called a *serial* Kripke model, and these models correspond to the scheme D, that is, $\Box f \rightarrow \Diamond f$. Note how our model $\mathcal{M}_1$ is neither reflexive nor serial. (Why?)

Of the presented axiom schemes, one is *always* true in any Kripke model. This is the axiom scheme K:

$$\Box(f \rightarrow g) \rightarrow (\Box f \rightarrow \Box g)$$

Exercise 5.I (i) Does a Kripke model $\mathcal{M}_4$ exist that is serial but not reflexive? If so, provide an example, and otherwise, explain why this is not possible.

(ii) Does a Kripke model $\mathcal{M}_5$ exist that is reflexive but not serial? If yes, provide an example, and otherwise, explain why this is not possible.

5.6 Temporal logic

We will now take a more detailed look at a specific temporal logic: LTL, which stands for *Linear Time Logic*. This is the logic that the SPIN model checker uses.[9] A *model checker* is a computer program that analyzes and pieces a software. In order to use a model checker, one first has to provide a finite automaton that describes the behavior of the software in question. Then, that automaton is fed into the checker, together with a formula of the temporal logic employed by the checker that expresses the property one wants to check of the software. The model checker then analyzes the automaton and verifies whether the property expressed by the formula holds or not.

An example of a property one may want to give to a model checker would be:

The automaton always eventually returns back to the state q_{42}.

(A property of this form is called a 'liveness' property, because it describes how the automaton always remains 'active' enough to be able to return back to a certain state.) A temporal logic formula that expresses this property would be:

$$\Box\Diamond q_{42}$$

This should be read as:

[9] SPIN stands for 'Simple Promela INterpreter' and Promela in turn stands for 'PRocess MEta LAnguage', which allows the description of finite automata. SPIN is one of the most well-known model checkers, and was developed by the Dutch Gerard Holzmann at Bell Laboratories in America.

From now on it will always hold that, there will be a point in time at which, we are back in state q_{42}.

You might be tempted to think the model checker's task is not very complex, and thus actually quite simple. The complication however, is that, even with automata of a relatively small number of states, the number of situations (worlds) that need to be checked grows very (very) fast. This, then, is the key task of a model checker: controlling what is called *state space explosion.*

Definition 5.10 In LTL, we don't denote the modal operators by $\Box$ and $\Diamond$, but by the capitals $\mathcal{G}$ and $\mathcal{F}$. So we have:

modal	LTL	shorthand	meaning
$\Box f$	$\mathcal{G}\,f$	$\mathcal{G}$lobally	from now on (including now) f will always hold
$\Diamond f$	$\mathcal{F}\,f$	$\mathcal{F}$uture	eventually f will hold (or it holds already)

In fact, LTL has even more operators:

LTL	shorthand	meaning
$\mathcal{X}\,f$	ne$\mathcal{X}$t	after exactly one step, f will hold
$f\,\mathcal{U}\,g$	$\mathcal{U}$ntil	f holds until g holds, and indeed eventually g will hold
$f\,\mathcal{W}\,g$	$\mathcal{W}$eak until	f holds until g holds, or else f will hold forever
$f\,\mathcal{R}\,g$	$\mathcal{R}$elease	g holds until f holds, at which point g is still the case, or else g will hold forever

As you can see, LTL has binary modal operators as well. The operator $\mathcal{X}f$ is sometimes also denoted with a small circle: $\circ f$.

Exercise 5.J Express the following sentences in LTL. You may use all operators of LTL in your answers.

(i) *There exists a moment after which the formula a will always be true.*
(ii) *The statements a and b are alternatingly true.*
(iii) *Every time a holds, b holds after a while as well.*

Some of the operators of LTL can in fact be expressed in terms of the others. For instance, we have:

$$
\begin{aligned}
f\,\mathcal{U}\,g &\quad \text{is equivalent to} \quad (f\,\mathcal{W}\,g) \wedge \mathcal{F}\,g \\
f\,\mathcal{W}\,g &\quad \text{is equivalent to} \quad (f\,\mathcal{U}\,g) \vee \mathcal{G}\,f \\
f\,\mathcal{R}\,g &\quad \text{is equivalent to} \quad \neg(\neg f\,\mathcal{U}\,\neg g)
\end{aligned}
$$

Exercise 5.K Define the operator $\mathcal{U}$ in terms of $\mathcal{R}$.

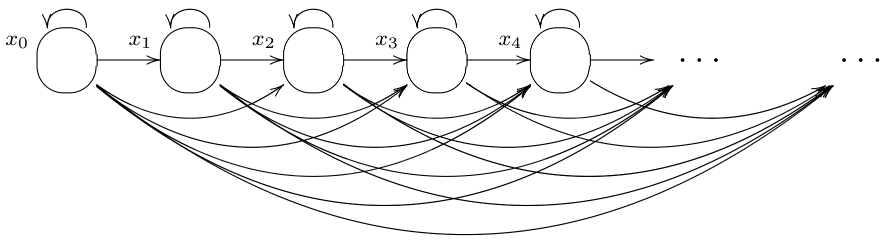
Exercise 5.L Define the operators $\mathcal{G}$ and $\mathcal{F}$ in terms of $\mathcal{U}$ and $\mathcal{R}$. You may use the propositions $\top$ and $\bot$, the are always true, respectively always false.

Exercise 5.M Show that all LTL operators (except $\mathcal{X}$) can be defined in terms of the $\mathcal{W}$ operator. Again, you may use the propositions $\top$ and $\bot$.

The worlds in Kripke models of LTL corresponds to the ticking of a clock. Another way to putting it, is that each world can be uniquely identified with a natural number. So, the set of worlds is:

$$W = \{x_i \mid i \in \mathbb{N}\}$$

Furthermore, the worlds accessible to a given world x_i are exactly all subsequent worlds x_j, for which $i \leq j$. The only difference then, between different Kripke models of LTL, is which atomic propositions hold in its worlds. Here is a picture illustrating what a Kripke model of LTL looks like:



This formulation of worlds in terms of the ticking of a clock allows us to describe the meaning of the operators of LTL of definition 5.10 in an even more mathematically precise way:

Definition 5.11 Given a Kripke model $\langle W, R, V \rangle$ with W and R as described above. Then:

$x_i \Vdash \mathcal{G}f$	for all $j \geq i$ we have $x_j \Vdash f$
$x_i \Vdash \mathcal{F}f$	there is a $j \geq i$ such that $x_j \Vdash f$
$x_i \Vdash \mathcal{X}f$	$x_{i+1} \Vdash f$
$x_i \Vdash f\,\mathcal{U}\,g$	there is a $j \geq i$ such that $x_j \Vdash g$ and for all $k \in \{i, i+1, \ldots, j-1\}$ we have $x_k \Vdash f$
$x_i \Vdash f\,\mathcal{W}\,g$	*either* there is a $j \geq i$ such that $x_j \Vdash g$ and for all $k \in \{i, i+1, \ldots, j-1\}$ we have $x_k \Vdash f$, *or* for all $l \geq i$ we have $x_l \Vdash f$
$x_i \Vdash f\,\mathcal{R}\,g$	*either* there is a $j \geq i$ such that $x_j \Vdash f$ and for all $k \in \{i, i+1, \ldots, j-1, j\}$ we have $x_k \Vdash g$, *or* for all $l \geq i$ we have $x_l \Vdash g$

With this formalization, we could prove the above-mentioned equivalences between LTL operators.

Exercise 5.N Consider the LTL Kripke model $\mathcal{M}_6 = \langle W, R, V \rangle$. So we know that $W = \{x_i \mid i \in \mathbb{N}\}$ and $x_j \in R(x_i)$ if $i \leq j$. Now define V as follows:

$$a \in V(x_i) \quad \text{iff} \quad i \text{ is a multiple of two twee}$$
$$b \in V(x_i) \quad \text{iff} \quad i \text{ is a multiple of two three}$$

Check whether the following properties hold:
(i) $x_1 \Vdash \mathcal{F}(a \wedge b)$
(ii) $x_6 \Vdash \mathcal{G}(a \vee b)$
(iii) $\mathcal{M}_6 \vDash \mathcal{GF}(a\,\mathcal{U}\,b)$

Exercise 5.O Which of the axiom schemes in the table on page 69 hold in LTL?

5.7 Important concepts

Index