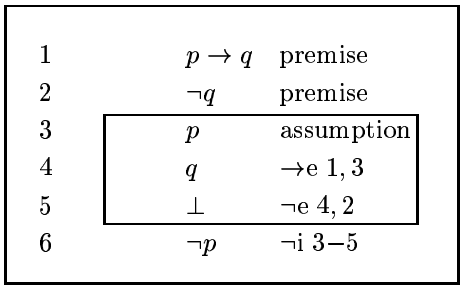
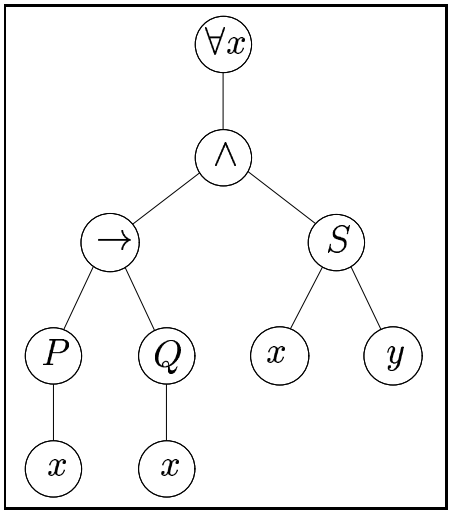
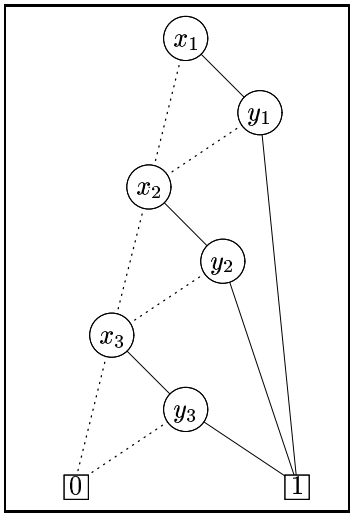
Second Edition
Logic in Computer Science:
Modelling and reasoning about systems
Solutions to designated exercises

Michael Huth and Mark Ryan



1		$p \to q$	premise
2		$\neg q$	premise
3		p	assumption
4		q	$\to$e 1, 3
5		$\bot$	$\neg$e 4, 2
6		$\neg p$	$\neg$i 3–5






```
k = 2;
t = a[1];
s = a[1];
while (k != n+1) {
   t = min(t+a[k], a[k]);
   s = min(s,t);
   k = k+1;
}
```

Please report errors and ommissions to:

MICHAEL HUTH
Department of Computing
Imperial College London, UK
mrh@doc.imperial.ac.uk

Contents

1

Propositional logic

EXERCISES 1.1 (p.78)

1(a). We chose p to stand for "The sun shines today." and q denotes "The sun shines tomorrow." The corresponding formula is then

$$p \to \neg q\,.$$

NB: If we had chosen q to denote "The sun does not shine tomorrow." then the corresponding formula would be

$$p \to q\ .$$

1(d). We choose

p : "A request occurs."

q : "The request will eventually be acknowledged."

r : "The requesting process will eventually make progress."

The formula representing the declarative sentence is then

$$p \to (q \vee \neg r)\,.$$

NB: If we had chosen r to denote "The requesting process won't ever be able to make progress," the corresponding formula would be

$$p \to (q \vee r)\ .$$

1(h). We choose

r : "Today, it will shine."

s : "Today, it will rain."

The resulting formula is $(r \vee s) \wedge \neg(r \wedge s)$.

1(i). The proposition atoms we chose are

p : "Dick met Jane yesterday."

q : "Dick and Jane had a cup of coffee together."

r : "Dick and Jane had a walk in the park."

This results in the formula

$$p \to q \vee r$$

which reads as $p \to (q \vee r)$ if we recall the binding priorities of our logical operators.

2(a). $((\neg p) \wedge q) \to r$.

2(c). $(p \to q) \to (r \to (s \vee t))$.

2(e). $(p \vee q) \to ((\neg p) \wedge r)$.

2(g). The expression $p \vee q \wedge r$ is problematic since $\wedge$ and $\vee$ have the same binding priorities, so we have to insist on additional brackets in order to resolve this conflict.

EXERCISES 1.2 (p.78)

1(a). (Bonus) We prove the validity of $(p \wedge q) \wedge r,\ s \wedge t \vdash q \wedge s$ by

1	$(p \wedge q) \wedge r$	premise
2	$s \wedge t$	premise
3	$p \wedge q$	$\wedge e_1$ 1
4	q	$\wedge e_2$ 3
5	s	$\wedge e_1$ 2
6	$q \wedge s$	$\wedge i$ 4, 5

1(c) We prove the validity of $(p \wedge q) \wedge r \vdash p \wedge (q \wedge r)$ by

1	$(p \wedge q) \wedge r$	premise
2	$p \wedge q$	$\wedge e_1$ 1
3	r	$\wedge e_2$ 1
4	p	$\wedge e_1$ 2
5	q	$\wedge e_2$ 2
6	$q \wedge r$	$\wedge i$ 5, 3
7	$p \wedge (q \wedge r)$	$\wedge i$ 4, 6

1(e). One possible proof of the validity of $q \to (p \to r), \neg r, q \vdash \neg p$ is

1	$q \to (p \to r)$	premise
2	$\neg r$	premise
3	q	premise
4	$p \to r$	$\to$e $1, 3$
5	$\neg p$	MT $4, 2$

1(f). We prove the validity of $\vdash (p \wedge q) \to p$ by

1	$p \wedge q$	assumption
2	p	$\wedge e_1$ 1
3	$p \wedge q \to p$	$\to$i $1 - 2$

1(h). We prove the validity of $p \vdash (p \to q) \to q$ by

1	p	premise
2	$p \to q$	assumption
3	q	$\to$e $2, 1$
4	$(p \to q) \to q$	$\to$i $2 - 3$

1(i). We prove the validity of $(p \to r) \wedge (q \to r) \vdash p \wedge q \to r$ by

1	$(p \to r) \wedge (q \to r)$	premise
2	$p \wedge q$	assumption
3	p	$\wedge e_1$ 2
4	$p \to r$	$\wedge e_1$ 1
5	r	$\to$e $4, 3$
6	$p \wedge q \to r$	$\to$i $2 - 5$

1(j). We prove the validity of $q \to r \vdash (p \to q) \to (p \to r)$ by

1	$q \to r$	premise
2	$p \to q$	assumption
3	p	assumption
4	q	$\to$e $2, 3$
5	r	$\to$e $1, 4$
6	$p \to r$	$\to$i $3 - 5$
7	$(p \to q) \to (p \to r)$	$\to$i $2 - 6$

1(l). We prove the validity of $p \to q, r \to s \vdash p \vee r \to q \vee s$ by

1	$p \to q$	premise
2	$r \to s$	premise
3	$p \vee r$	assumption
4	p	assumption
5	q	$\to$e $1, 4$
6	$q \vee s$	$\vee$i$_1$ 5
7	r	assumption
8	s	$\to$e $2, 7$
9	$q \vee s$	$\vee$i$_2$ 8
10	$q \vee s$	$\vee$e $3, 4 - 6, 7 - 9$
11	$p \vee r \to q \vee s$	$\to$i $3 - 10$

1(n) We prove the validity of $(p \vee (q \to p)) \wedge q \vdash p$ by

1	$(p \vee (q \to p)) \wedge q$	premise
2	q	$\wedge$e$_2$ 1
3	$p \vee (q \to p)$	$\wedge$e$_1$ 1
4	p	assumption
5	$q \to p$	assumption
6	p	$\to$e $5, 2$
7	p	$\vee$e $3, 4 - 4, 5 - 6$

Note that one could have put line 2 in between lines 5 and 6 with

the corresponding renumbering of pointers. Would the proof above still be valid if we used rules $\wedge e_2$ and $\wedge e_1$ in the other ordering?

1(o). We prove the validity of $p \to q, r \to s \vdash p \wedge r \to q \wedge s$ by

1	$p \to q$	premise
2	$r \to s$	premise
3	$p \wedge r$	assumption
4	p	$\wedge e_1$ 3
5	q	$\to e$ 1, 4
6	r	$\wedge e_2$ 3
7	s	$\to e$ 2, 6
8	$q \wedge s$	$\wedge i$ 5, 7
9	$p \wedge r \to q \wedge s$	$\to i$ 3 – 8

1(r). We prove the validity of $p \to q \wedge r \vdash (p \to q) \wedge (p \to r)$ by

1	$p \to q \wedge r$	premise
2	p	assumption
3	$q \wedge r$	$\to e$ 1, 2
4	q	$\wedge e_1$ 3
5	$p \to q$	$\to i$ 2 – 4
6	p	assumption
7	$q \wedge r$	$\to e$ 1, 6
8	r	$\wedge e_2$ 7
9	$p \to r$	$\to i$ 6 – 8
10	$(p \to q) \wedge (p \to r)$	$\wedge i$ 5, 9

The reader may wonder why two separate, although almost identical, arguments have to be given. Our proof rules force this structure (= assuming p *twice*) upon us. If we proved q and r in the same box, then we would be able to show $p \to q \wedge r$ which is our premise and not what we are after.

1(v). We prove the validity of $p \vee (p \wedge q) \vdash p$ by

1	$p \vee (p \wedge q)$	premise
2	p	assumption
3	$p \wedge q$	assumption
4	p	$\wedge e_1$ 3
5	p	$\vee e$ $1, 2-2, 3-4$

1(x). We prove the validity of $p \to (q \vee r), q \to s, r \to s \vdash p \to s$ by

1	$p \to (q \vee r)$	premise
2	$q \to s$	premise
3	$r \to s$	premise
4	p	assumption
5	$q \vee r$	$\to e$ $1, 4$
6	q	assumption
7	s	$\to e$ $2, 6$
8	r	assumption
9	s	$\to e$ $3, 8$
10	s	$\vee e$ $5, 6-7, 8-9$
11	$p \to s$	$\to i$ $4-10$

1(y). We prove the validity of $(p \wedge q) \vee (p \wedge r) \vdash p \wedge (q \vee r)$ by

1	$(p \wedge q) \vee (p \wedge r)$	premise
2	$p \wedge q$	assumption
3	p	$\wedge e_1$ 2
4	q	$\wedge e_2$ 2
5	$q \vee r$	$\vee i_1$ 4
6	$p \wedge (q \vee r)$	$\wedge$i 3, 5
7	$p \wedge r$	assumption
8	p	$\wedge e_1$ 7
9	r	$\wedge e_2$ 7
10	$q \vee r$	$\vee i_2$ 9
11	$p \wedge (q \vee r)$	$\wedge$i 8, 10
12	$p \wedge (q \vee r)$	$\vee$e $1, 2-6, 7-11$

2(a). We prove the validity of $\neg p \to \neg q \vdash q \to p$ by

1	$\neg p \to \neg q$	premise
2	q	assumption
3	$\neg\neg q$	$\neg\neg$i 2
4	$\neg\neg p$	MT 1, 3
5	p	$\neg\neg$e 4
6	$q \to p$	$\to$i $2-5$

2(b). We prove the validity of $\neg p \vee \neg q \vdash \neg(p \wedge q)$ by

1	$\neg p \vee \neg q$	premise
2	$\neg p$	assumption
3	$p \wedge q$	assumption
4	p	$\wedge e_1$ 3
5	$\bot$	$\neg$e 4, 2
6	$\neg(p \wedge q)$	$\neg$i 3 – 5
7	$\neg q$	assumption
8	$p \wedge q$	assumption
9	q	$\wedge e_2$ 8
10	$\bot$	$\neg$e 9, 7
11	$\neg(p \wedge q)$	$\neg$i 8 – 10
12	$\neg(p \wedge q)$	$\vee$e 1, 2 – 6, 7 – 11

2(c). We prove the validity of $\neg p, p \vee q \vdash q$ by

1	$\neg p$	premise
2	$p \vee q$	premise
3	p	assumption
4	$\bot$	$\neg$e 3, 1
5	q	$\bot$e 4
6	q	assumption
7	q	$\vee$e 2, 3 – 5, 6 – 6

2(d). We prove the validity of $p \vee q, \neg q \vee r \vdash p \vee r$ by

1	$p \vee q$	premise
2	$\neg q \vee r$	premise
3	$\neg q$	assumption
4	$p \vee q$	copy 1
5	p	assumption
6	$p \vee r$	$\vee$i$_1$ 5
7	q	assumption
8	$\bot$	$\neg$e 7, 3
9	$p \vee r$	$\bot$e 8
10	$p \vee r$	$\vee$e $4, 5-6, 7-9$
11	r	assumption
12	$p \vee r$	$\vee$i$_2$ 11
13	$p \vee r$	$\vee$e $2, 3-10, 11-12$

Observe how the format of the proof rule $\vee$e forces us to use the copy rule if we nest two disjunctions as premises we want to eliminate.

2(e). We prove the validity of $p \to (q \vee r), \neg q, \neg r \vdash \neg p$ by

1	$p \to (q \vee r)$	premise
2	$\neg q$	premise
3	$\neg r$	premise
4	p	assumption
5	$q \vee r$	$\to$e $1, 4$
6	q	assumption
7	$\bot$	$\neg$e $6, 2$
8	r	assumption
9	$\bot$	$\neg$e $8, 3$
10	$\bot$	$\vee$e $5, 6-7, 8-9$
11	$\neg p$	$\neg$i $4-10$

2(f). We prove the validity of $\neg p \wedge \neg q \vdash \neg(p \vee q)$ by

1	$\neg p \wedge \neg q$	premise
2	$p \vee q$	assumption
3	p	assumption
4	$\neg p$	$\wedge e_1$ 1
5	$\bot$	$\neg e$ 3, 4
6	q	assumption
7	$\neg q$	$\wedge e_2$ 1
8	$\bot$	$\neg e$ 6, 7
9	$\bot$	$\vee e$ 2, 3 – 5, 6 – 8
10	$\neg(p \vee q)$	$\neg i$ 2 – 9

2(g). We prove the validity of $p \wedge \neg p \vdash \neg(r \to q) \wedge (r \to q)$ by

1	$p \wedge \neg p$	premise
2	p	$\wedge e_1$ 1
3	$\neg p$	$\wedge e_2$ 1
4	$\bot$	$\neg e$ 2, 3
5	$\neg(r \to q) \wedge (r \to q)$	$\bot e$ 4

2(i). We prove the validity of $\neg(\neg p \vee q) \vdash p$ by

1	$\neg(\neg p \vee q)$	premise
2	$\neg p$	assumption
3	$\neg p \vee q$	$\vee i_1$ 2
4	$\bot$	$\neg e$ 3, 1
5	$\neg\neg p$	$\neg i$ 2 – 4
6	p	$\neg\neg e$ 5

Note that lines 5 and 6 could be compressed to one line with the application of RAA.

3(d). We prove the validity of $\vdash \neg p \to (p \to (p \to q))$ by

1	$\neg p$	assumption
2	p	assumption
3	p	assumption
4	$\bot$	$\neg$e $3, 1$
5	q	$\bot$e 4
6	$p \to q$	$\to$i $3 - 5$
7	$p \to (p \to q)$	$\to$i $2 - 6$
8	$\neg p \to (p \to (p \to q))$	$\to$i $1 - 7$

Note that all three assumptions/boxes are required by the format of our $\to$i rule.

3(n). We prove the validity of $p \wedge q \vdash \neg(\neg p \vee \neg q)$ by

1	$p \wedge q$	assumption
2	$\neg p \vee \neg q$	assumption
3	$\neg p$	assumption
4	p	$\wedge e_1$ 1
5	$\bot$	$\neg$e $4, 3$
6	$\neg q$	assumption
7	q	$\wedge e_2$ 1
8	$\bot$	$\neg$e $7, 6$
9	$\bot$	$\vee$e $2, 3 - 5, 6 - 8$
10	$\neg(\neg p \vee \neg q)$	$\neg$i $2 - 9$

3(q). We prove the validity of $(p \to q) \vee (q \to r)$ by

1	$q \vee \neg q$	lemma
2	q	assumption
3	p	assumption
4	q	copy 2
5	$p \to q$	$\to$i $3-4$
6	$(p \to q) \vee (q \to r)$	$\vee i_1$ 5
7	$\neg q$	assumption
8	q	assumption
9	$\bot$	$\neg$e $8, 7$
10	r	$\bot$e 9
11	$q \to r$	$\to$i $8-10$
12	$(p \to q) \vee (q \to r)$	$\vee i_2$ 11
13	$(p \to q) \vee (q \to r)$	$\vee$e $1, 2-6, 7-12$

5(a). We prove the validity of $\vdash ((p \to q) \to q) \to ((q \to p) \to p)$ by

1	$(p \to q) \to q$	assumption
2	$q \to p$	assumption
3	$\neg p$	assumption
4	p	assumption
5	$\bot$	$\neg$e $4, 3$
6	q	$\bot$e 5
7	$p \to q$	$\to$i $4-6$
8	q	$\to$e $1, 7$
9	p	$\to$e $2, 8$
10	$\bot$	$\neg$e $9, 3$
11	$\neg\neg p$	$\neg$i $3-10$
12	p	$\neg\neg$e 11
13	$(q \to p) \to p$	$\to$i $2-12$
14	$((p \to q) \to q) \to ((q \to p) \to p)$	$\to$i $1-13$

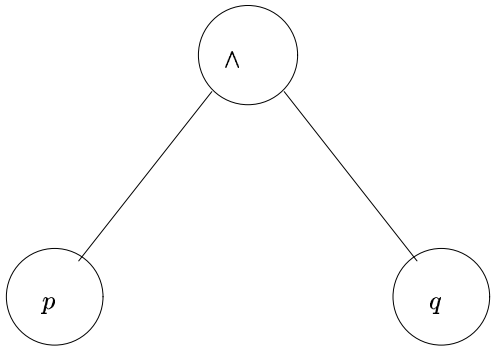
Let us motivate some of this proof's strategic choices. First, the opening of assumption boxes in lines 1 and 2 has nothing to do with strategy; it is simply dictated by the format of the formula we wish to prove. The assumption of $\neg p$ in line 3, however, is a strategic move with the desire to derive $\bot$ in order to get p in the end. Similarly, the assumption of p in line 4 is such a strategic move which tries to derive $p \to q$ which can be used to obtain $\bot$.

5(d). We prove the validity of $\vdash (p \to q) \to ((\neg p \to q) \to q)$ by

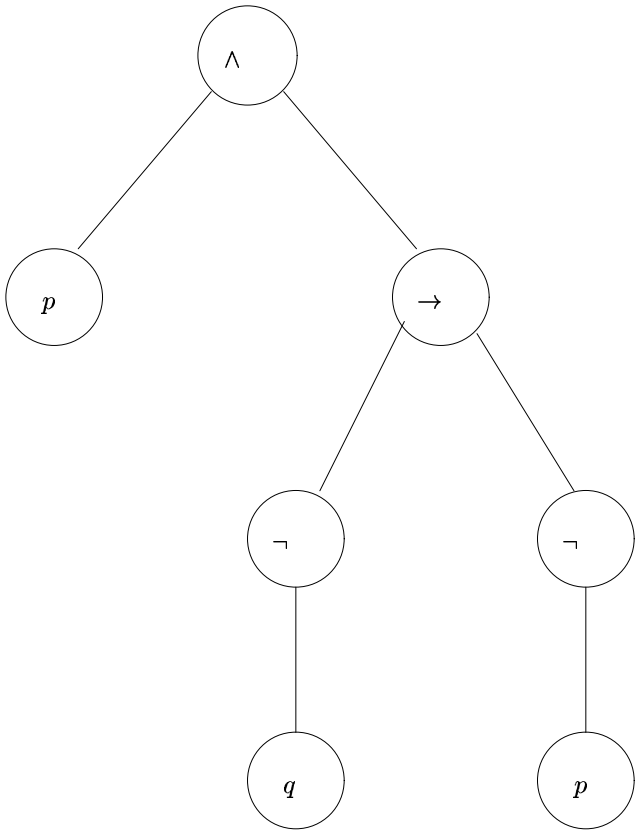
1	$p \to q$	assumption
2	$\neg p \to q$	assumption
3	$p \vee \neg p$	lemma
4	p	assumption
5	q	$\to$e 1, 4
6	$\neg p$	assumption
7	q	$\to$e 2, 6
8	q	$\vee$e $3, 4-5, 6-7$
9	$(\neg p \to q) \to q$	$\to$i $2-8$
10	$(p \to q) \to ((\neg p \to q) \to q)$	$\to$i $1-9$

EXERCISES 1.3 (p.81)

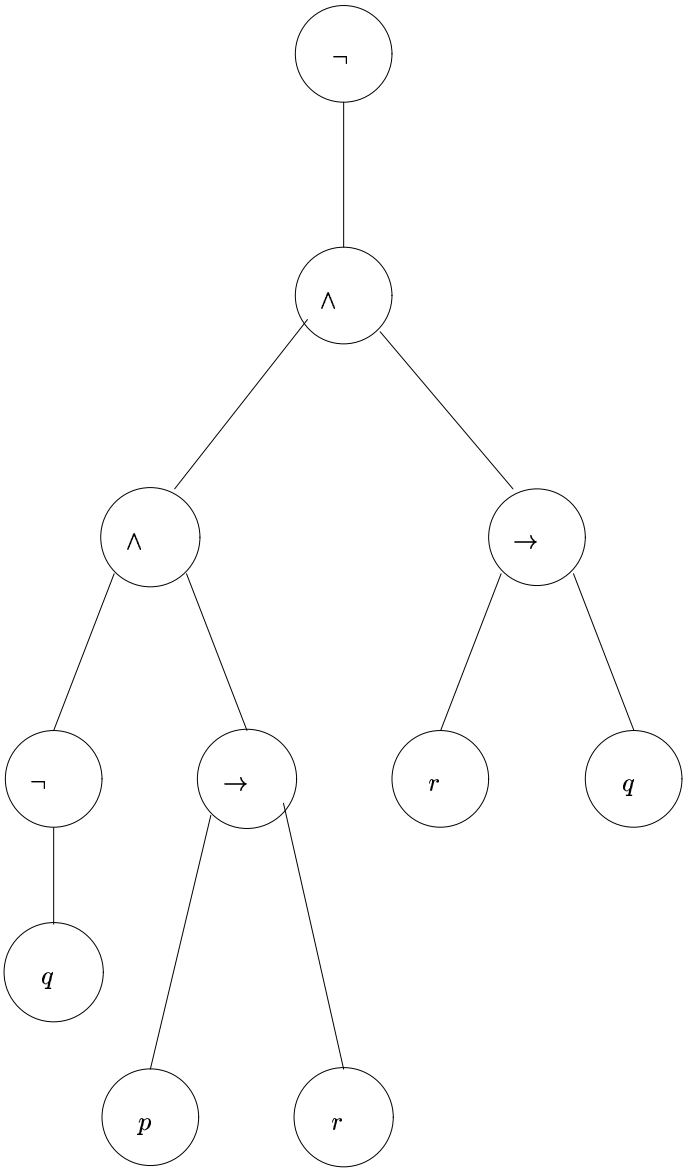
1(b). The parse tree of $p \wedge q$ is



1(d). The parse tree of $p \wedge (\neg q \to \neg p)$ is



1(f). The parse tree of $\neg((\neg q \wedge (p \to r)) \wedge (r \to q))$ is



2(a). The list of *all* subformulas of the formula $p \to (\neg p \vee (\neg\neg q \to (p \wedge q)))$ is

p

q

$\neg p$

$\neg q$

$\neg\neg q$

$p \wedge q$

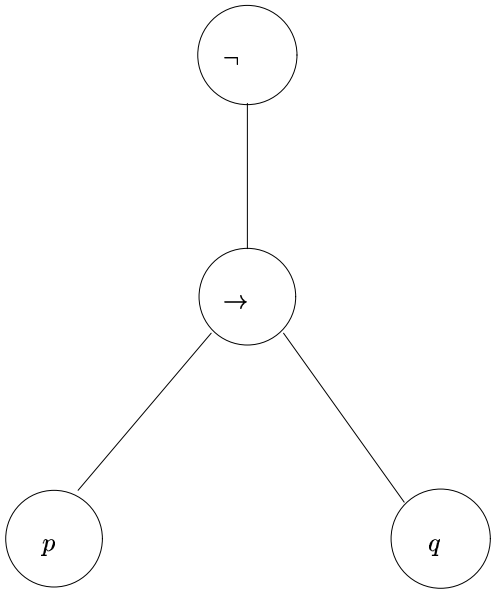
$\neg\neg q \to (p \wedge q)$

$\neg p \vee (\neg\neg q \to (p \wedge q))$

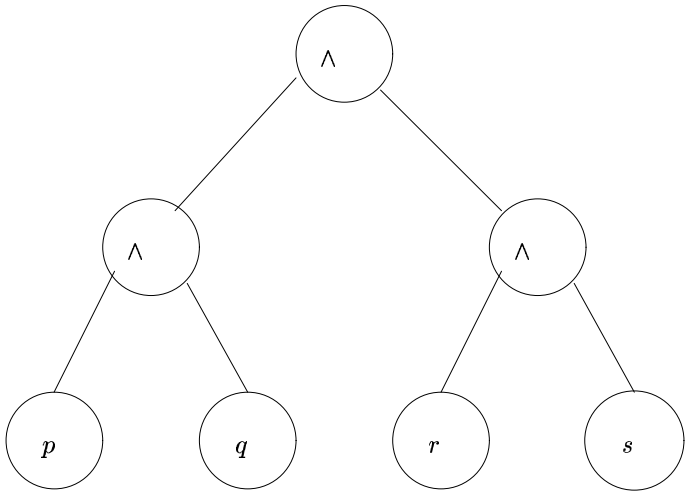
$p \to (\neg p \vee (\neg\neg q \to (p \wedge q)))$.

Note that $\neg q$ and $\neg\neg q$ are two different subformulas.

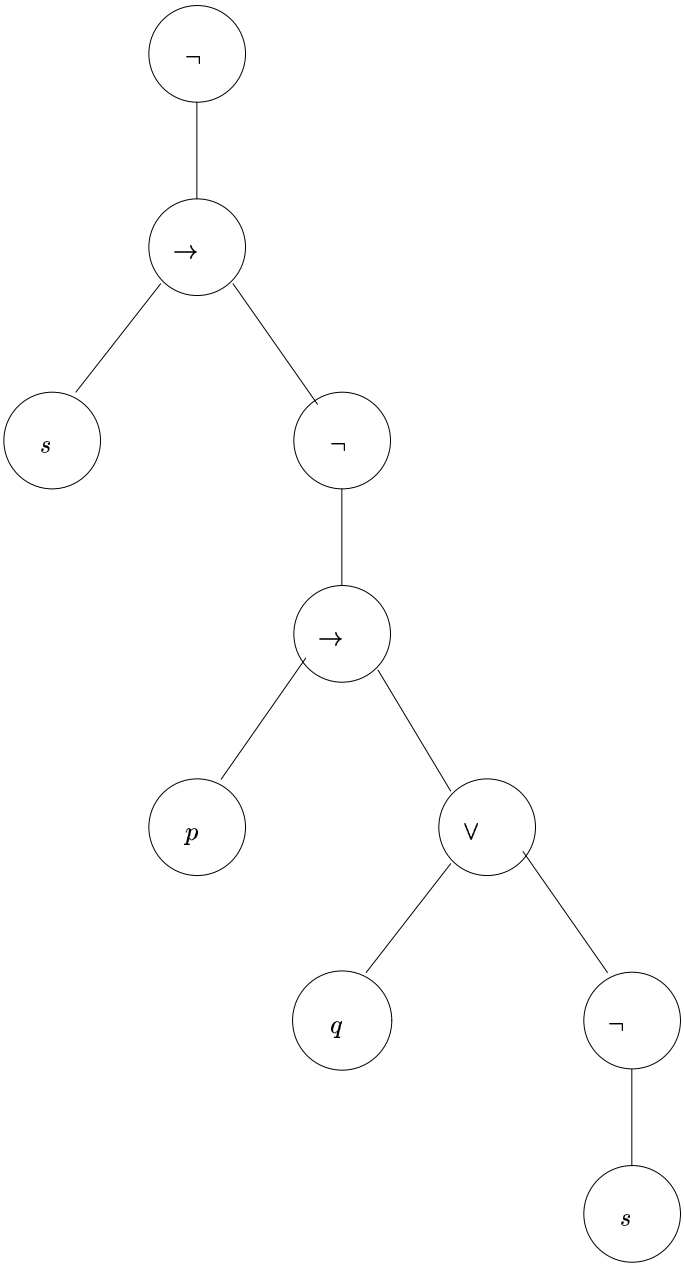
3(a). An example of a parse tree of a propositional logic formula which is a negation of an implication:



3(c). An example of a parse tree of a formula which is a conjunction of conjunctions is



4(a). The parse tree of $\neg(s \to (\neg(p \to (q \vee \neg s))))$ is



Its list of subformulas is

p

q

s

$\neg s$

$q \vee \neg s$

$p \rightarrow (q \vee \neg s)$

$\neg(p \rightarrow (q \vee \neg s))$

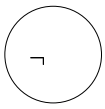
$s \rightarrow (\neg(p \rightarrow (q \vee \neg s)))$

$\neg(s \rightarrow (\neg(p \rightarrow (q \vee \neg s))))$.

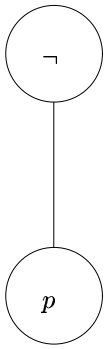
5. The logical formula represented by the parse tree in Figure 1.22 is

$$\neg(\neg(p \vee (q \wedge \neg p)) \rightarrow r).$$

7(a). The parse tree

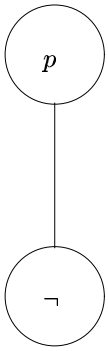


does not correspond to a well-formed formula, but its extension



does.

7(b). The parse tree



is *inherently* ill-formed; i.e. any extension thereof could not correspond to a well-formed formula. (Why?)

EXERCISES 1.4 (p.82)

1. The truth table for $\neg p \vee q$ is

p	q	$\neg p$	$\neg p \vee q$
T	T	F	T
T	F	F	F
F	T	T	T
F	F	T	T

and matches the one for $p \to q$.

2(a). The complete truth table of the formula $((p \to q) \to p) \to p$ is given by

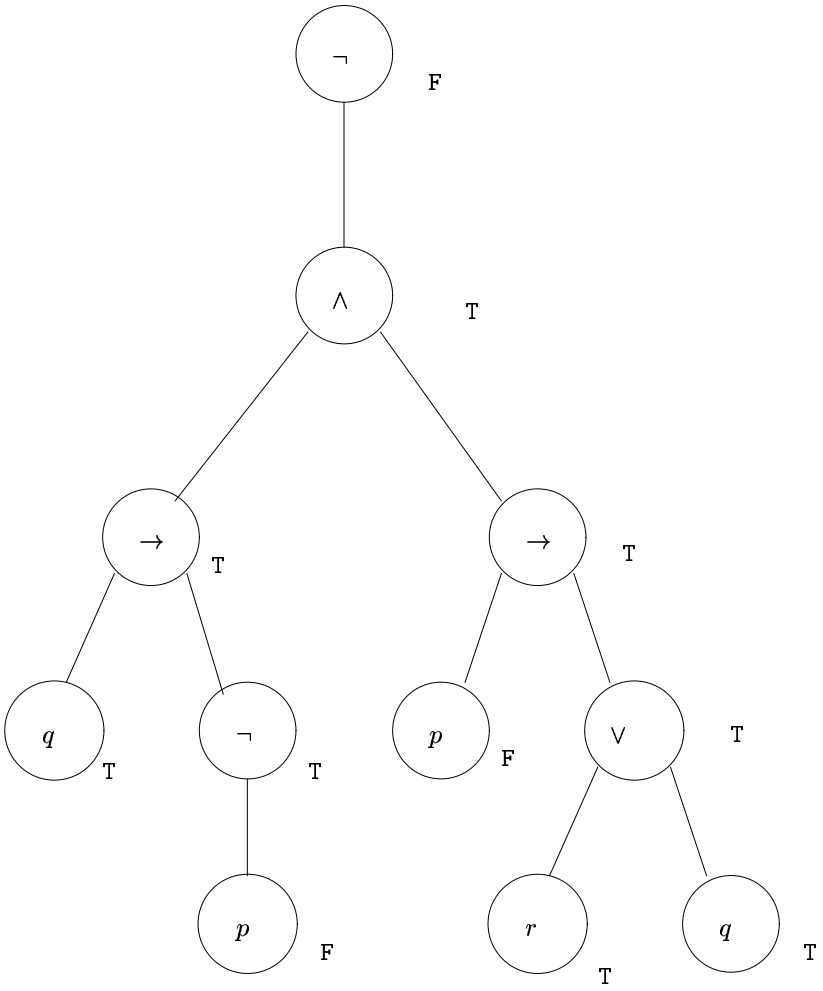
p	q	$p \to q$	$(p \to q) \to p$	$((p \to q) \to p) \to p$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	F	T

Note that this formula is valid since it always evaluates to T.

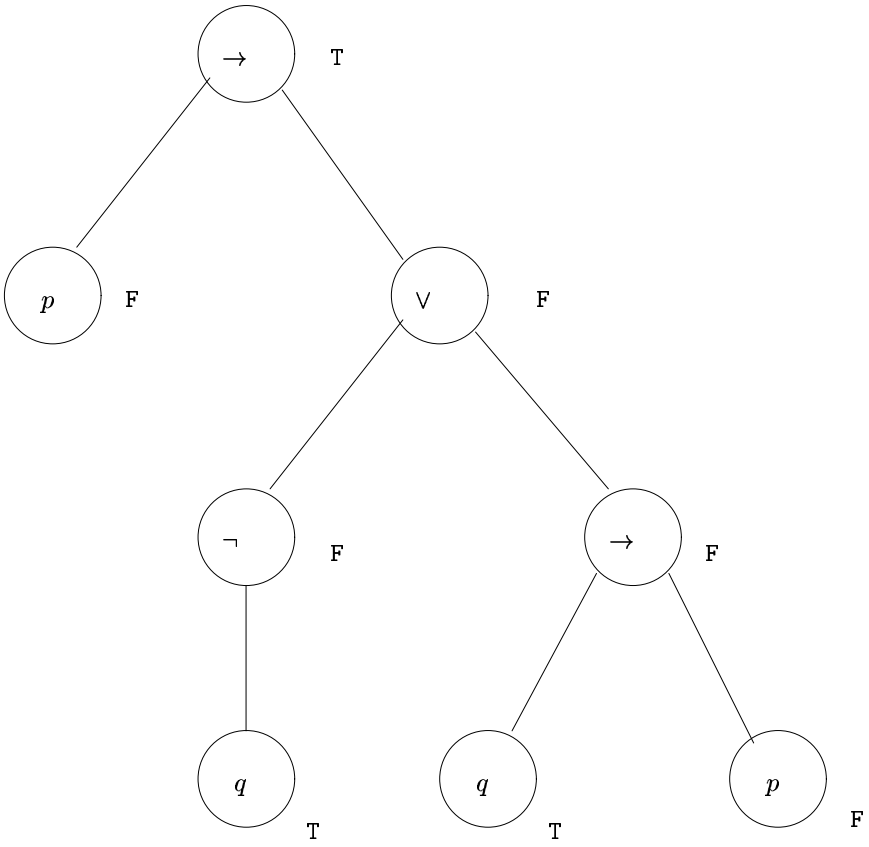
2(c). The *complete* truth table for $p \vee (\neg(q \wedge (r \to q)))$ is

p	q	r	$r \to q$	$q \wedge (r \to q)$	$\neg(q \wedge (r \to q))$	$p \vee (\neg(q \wedge (r \to q)))$
T	T	T	T	T	F	T
T	T	F	T	T	F	T
T	F	T	F	F	T	T
F	T	T	T	T	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	F
F	F	T	F	F	T	T
F	F	F	T	F	T	T

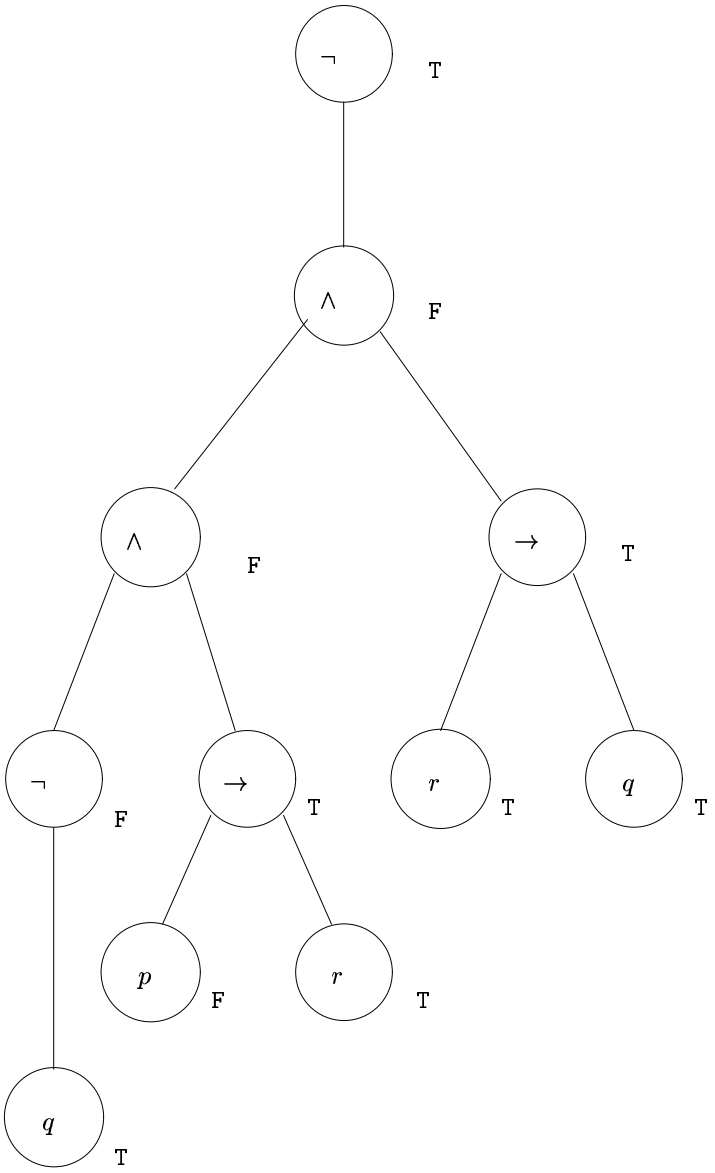
3(a). The requested truth value of the formula in Figure 1.10 on page 44 computed in a bottom-up fashion:



4(a). We compute the truth value of $p \to (\neg q \vee (q \to p))$ in a bottom-up fashion on its parse tree:



4(b). Similarly, we compute the truth value of $\neg((\neg q \wedge (p \to r)) \wedge (r \to q))$ in a bottom-up fashion on its parse tree:



5. The formula of the parse tree in Figure 1.10 on page 44 is not valid since it already evaluates to `F` for any assigment in which p and q evaluate to `F`. However, this formula is satisfiable: for example, if q and p evaluate to `T` then $q \to \neg p$ renders `F` and so the entire formula evaluates to `T`.

7(c). We prove

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6} \tag{1.1}$$

for all natural numbers $n \geq 1$ by mathematical induction on n.

1. **Base Case:** If $n = 1$, then the lefthand side of (1.1) equals 1^2 which is 1; the righthand side equals $1 \cdot (1+1) \cdot (2 \cdot 1 + 1)/6 = 2 \cdot 3/6 = 1$ as well. Thus, equation (1.1) holds for our base case.
2. **Inductive Step:** Our **induction hypothesis** is that (1.1) holds for n. Our task is to use that information to show that (1.1) also holds for $n+1$. The lefthand side for $n+1$ equals $1^2+2^2+3^2+\cdots+n^2+(n+1)^2$ which, by our induction hypothesis, equals

$$\frac{n \cdot (n+1) \cdot (2n+1)}{6} + (n+1)^2. \tag{1.2}$$

The righthand side of (1.1) for $n+1$ equals

$$\frac{(n+1) \cdot ((n+1)+1) \cdot (2(n+1)+1)}{6} \tag{1.3}$$

Thus, we are done if the expressions in (1.2) and (1.3) are equal. Using our algebra skills acquired in high-school, we see that both expressions equal

$$\frac{(n+1) \cdot (2n^2 + 7n + 6)}{6}. \tag{1.4}$$

7(d). We use mathematical induction to show that $2^n \geq n + 12$ for all natural numbers $n \geq 4$. Before we do that we inspect whether we could improve this statement somehow: if $n = 1, 2$ or 3 then 2^n is smaller than $n + 12$; e.g. $2^3 = 8$ is smaller than $3 + 12 = 15$. Thus, we can only hope to prove $2^n \geq n + 12$ for $n \geq 4$; there is no room for improvement.

1. **Base Case:** Our base case lets n be 4. Then $2^n = 2^4 = 16$ is greater or equal to $4 + 12 = 16$.
2. **Inductive Step:** We need to show that $2^{n+1} \geq (n+1) + 12$, where our **induction hypothesis** assumes that $2^n \geq n + 12$. Thus,

$$\begin{aligned} 2^{n+1} &= 2 \cdot 2^n \\ &\geq 2 \cdot (n+12) \qquad \text{by our induction hypothesis} \end{aligned}$$

$$\begin{aligned} &= [(n+1)+12]+(n+11) \\ &\geq (n+1)+12 \end{aligned}$$

guarantees that our claim holds for $n+1$. Note that the first step of this computation also uses that multiplication with a positive number is *monotone*: $x \geq y$ implies $2 \cdot x \geq 2 \cdot y$.

8. The Fibonacci numbers are defined by

$$\begin{aligned} F_1 &\stackrel{\text{def}}{=} 1 \\ F_2 &\stackrel{\text{def}}{=} 1 \\ F_{n+1} &\stackrel{\text{def}}{=} F_n + F_{n-1} \qquad \text{for all } n \geq 2. \end{aligned} \tag{1.5}$$

We use mathematical induction to prove that F_{3n} is even for all $n \geq 1$.

1. **Base Case:** For $n = 1$, we compute $F_{3\cdot 1} = F_3 = F_2 + F_1$ by (1.5), but the latter is just $1 + 1 = 2$ which is even.
2. **Inductive Step:** Our **induction hypothesis** assumes that F_{3n} be even. We need to show that $F_{3\cdot(n+1)}$ is even as well. This is a bit tricky as we have to decide on where to apply the inductive definition of (1.5):

$$\begin{aligned} F_{3\cdot(n+1)} &= F_{3n+3} \\ &= F_{(3n+2)+1} \\ &= F_{3n+2} + F_{(3n+2)-1} \qquad \text{by (1.5)} \\ &= F_{(3n+1)+1} + F_{3n+1} \\ &= (F_{3n+1} + F_{3n}) + F_{3n+1} \qquad \text{unfolding } F_{(3n+1)+1} \text{ with (1.5)} \\ &= 2 \cdot F_{3n+1} + F_{3n}. \end{aligned}$$

Since F_{3n} is assumed to be even and since $2 \cdot F_{3n+1}$ clearly is even, we conclude that $2 \cdot F_{3n+1} + F_{3n}$, and therefore $F_{3\cdot(n+1)}$, is even as well.
Note that it was crucial not to unfold F_{3n+1} as well; otherwise, we would obtain four summands but no inductive argument. (Why?)

10. Consider the assertion

"*The number* $n^2 + 5n + 1$ *is even for all* $n \geq 1$."

(a) We prove the **inductive step** of that assertion as follows: we

simply assume that $n^2 + 5n + 1$ is even and we need to show that $(n+1)^2 + 5(n+1) + 1$ is then even as well. But

$$\begin{aligned}(n+1)^2 + 5(n+1) + 1 &= (n^2 + 2n + 1) + (5n + 5) + 1 \\ &= (n^2 + 5n + 1) + (2n + 6)\end{aligned}$$

must be even since

- $n^2 + 5n + 1$ is assumed to be even,
- $2n + 6 = 2 \cdot (n + 3)$ is always even,
- and the sum of two even numbers is even again.

(b) However, the **Base Case** fails to hold for this assertion since $n^2 + 5n + 1 = 1^2 + 5 \cdot 1 + 1 = 1 + 5 + 1 = 7$ is odd if $n = 1$.

(c) Thus, the assertion is false, for it is simply not true for $n = 1$.

(d) We use mathematical induction to show that $n^2 + 5n + 1$ is *odd* for all $n \geq 1$.

1. **Base Case:** If $n = 1$, then we have already computed that $n^2 + 5n + 1 = 7$ is odd.
2. **Inductive Step:** Our **induction hypothesis** assumes that $n^2 + 5n + 1$ is odd. Above we already computed

$$(n+1)^2 + 5(n+1) + 1 = (n^2 + 5n + 1) + (2n + 6) \quad (1.6)$$

(this computation involves only high-school algebra and has nothing to do with possible induction hypotheses). Thus,

$$(n+1)^2 + 5(n+1) + 1$$

must be odd, for

- $n^2 + 5n + 1$ is assumed to be odd,
- $2n + 6$ is always even,
- and the sum of an odd and an even number is odd.

12(c). We prove that the sequent $p \to (q \to r) \vdash p \to (r \to q)$ is not valid. Using soundness of our natural deduction calculus with respect to the truth-table semantics, it suffices to find an assignment of truth values to p, q and r such that $p \to (q \to r)$ evaluates to `T`, but $p \to (r \to q)$ evaluates to `F`. The latter can only occur if $p = \mathtt{T}$ and $q \to r = \mathtt{F}$. So $p = q = \mathtt{T}$ and $r = \mathtt{F}$ is the only choice which can defeat the claimed validity of this sequent. However, for this choice we easily compute that $p \to (q \to r)$ evaluates to `T` as this amounts to $\mathtt{T} \to (\mathtt{F} \to \mathtt{T})$ which computes to `T`.

13(a) Let p denote "France is a country." and q denote "France won the Euro 2004 Soccer Championships." Then $p \vee q$ holds in the world we currently live in, but $p \wedge q$ does not.

13(b) Here one can choose for p and q the same meaning as in item 13(a): $\neg p \to \neg q$ holds as p is true, but $\neg q \to \neg p$ is false as $\neg q$ is true whereas $\neg p$ is false.

17(b) This formula is indeed valid. To make it evaluate to `F` we need that both disjuncts evaluate, in particular $\cdot \to p$, have to evaluate to `F`. This p needs to evaluate to `F` and $\neg(p \to q)$ needs to evaluate to `T`. But the latter contradicts that p is `F`.

EXERCISES 1.5 (p.87)

2(a) (Bonus) We have $q \vee (\neg p \vee r) \equiv \neg p \vee (q \vee r) \equiv p \to (q \vee r)$ since $\vee$ is associative and implication can be decomposed in this way. Since $\equiv$ is transitive, we conclude that $q \vee (\neg p \vee r) \equiv p \to (q \vee r)$.

(b) $q \wedge \neg r \to p$ is not equivalent to $p \to q \vee r$ since $q \wedge \neg r \to p \not\models p \to (q \vee r)$: for let q and r be `F` and p be `T`. Then $q \wedge \neg r \to p$ computes `T`, whereas $p \to q \vee r$ computes `F`.

(c) (Bonus) We have $p \wedge \neg r \to q \equiv \neg(p \wedge \neg r) \vee q \equiv (\neg p \vee \neg\neg r) \vee q \equiv q \vee (\neg p \vee r)$ which is the formula in (a). Thus, $p \wedge \neg r \to q$ is equivalent to $p \to q \vee r$. Note that this made use of the identity $\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi$ which is shown in item 10(a) of Exercises 1.13.

(d) We have $\neg q \wedge \neg r \to \neg p \equiv \neg(\neg q \wedge \neg r) \vee \neg p \equiv (\neg\neg q \vee \neg\neg r) \vee \neg p \equiv q \vee (\neg p \vee r)$ and, as in the previous item, we conclude that $\neg q \wedge \neg r \to \neg p$ is equivalent to $p \to q \vee r$.

6(d)i. To show $\phi \wedge (\phi \vee \eta) \equiv \phi$, we need to show $\phi \wedge (\phi \vee \eta) \models \phi$ and $\phi \models \phi \wedge (\phi \vee \eta)$. To see $\phi \wedge (\phi \vee \eta) \models \phi$, assume that $\phi \wedge (\phi \vee \eta)$ evaluates to `T`. Then ϕ has to evaluate to `T` as well. To see $\phi \models \phi \wedge (\phi \vee \eta)$, assume that ϕ evaluates to `T`. Then $\phi \vee \eta$ evaluates to `T` as well. Thus, $\phi \wedge (\phi \vee \eta)$ evaluates to `T`, too.

6(e)ii. To show $\phi \vee (\psi \wedge \eta) \models (\phi \vee \psi) \wedge (\phi \vee \eta)$, assume that $\phi \vee (\psi \wedge \eta)$ evaluates to `T`.

Case 1: Suppose that ϕ evaluates to `T`. Then $\phi \vee \psi$ and $\phi \vee \eta$ evaluate to `T`. Therefore, $(\phi \vee \psi) \wedge (\phi \vee \eta)$ evaluates to `T` as well.

Case 2: Suppose that $\psi \wedge \eta$ evaluates to `T`. Then both ψ and η evaluate to `T`. Thus, $\phi \vee \psi$ and $\phi \vee \eta$ evaluate to `T`. Therefore, $(\phi \vee \psi) \wedge (\phi \vee \eta)$ evaluates to `T`.

Note that these two cases are exhaustive. (Why?)
To show $(\phi \vee \psi) \wedge (\phi \vee \eta) \models \phi \vee (\psi \wedge \eta)$, assume that $(\phi \vee \psi) \wedge (\phi \vee \eta)$ evaluates to T. Then $\phi \vee \psi$ and $\psi \vee \eta$ evaluate to T.

Case 1: Suppose that ϕ evaluates to F. Then we conclude that ψ and η have to evaluate to T. Thus, $\psi \wedge \eta$ evaluates to T. Therefore, $\phi \vee (\psi \wedge \eta)$ evaluates to T.

Case 2: Suppose that ϕ evaluates to T. Then $\phi \vee (\psi \wedge \eta)$ evaluates to T as well.

6(g)ii. To show $\neg(\phi \vee \psi) \models \neg\phi \wedge \neg\psi$, assume that $\neg(\phi \vee \psi)$ evaluates to T. Then $\phi \vee \psi$ evaluates to F. Thus, ϕ and ψ evaluate to F. So $\neg\phi$ and $\neg\psi$ evaluate to T. Therefore, $\neg\phi \wedge \neg\psi$ evaluates to T as well.
To show $\neg\phi \wedge \neg\psi \models \neg(\phi \vee \psi)$, assume that $\neg\phi \wedge \neg\psi$ evaluates to T. Then $\neg\phi$ and $\neg\psi$ evaluate to T. Thus, ϕ and ψ evaluate to F. This entails that $\phi \vee \psi$ evaluate to F. Therefore, $\neg(\phi \vee \psi)$ evaluates to T.

7(a). The formula ϕ_1 in CNF which we construct from the truth table

p	q	ϕ_1
T	T	F
F	T	F
T	F	F
F	F	T

is

$$(\neg p \vee \neg q) \wedge (p \vee \neg q) \wedge (\neg p \vee q).$$

Note how these principal conjuncts correspond to the lines in the table above, where the ϕ_1 entry is F: e.g. the third line T | F | F results in the conjunct $(\neg p \vee q)$.

7(b). The formula ϕ_2 in CNF based on the truth table

p	q	r	ϕ_2
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	T
F	T	F	F
F	F	T	T
F	F	F	F

is

$$(\neg p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (\neg p \vee q \vee r) \wedge (p \vee \neg q \vee r) \wedge (p \vee q \vee r).$$

Incidentally, if you are not sure about whether your answer is correct you can also try to verify it by ensuring that your answer has the same truth table as the above.

8. We write pseudo-code for a recursive function `IMPL_FREE` which expects a (parse tree of a) propositional formula as input and produces an equivalent formula as output such that the result contains no implications. Since our logic has five data constructors (= five ways of building a formula), namely atoms, negation, conjunction, disjunction, and implication, we have to consider five cases. Only in the case of implication do we have to do work beyond mere book-keeping:

function `IMPL_FREE` (ϕ) :
/* computes a formula without implication equivalent to ϕ */
begin function
case
ϕ is an atom: **return** ϕ
ϕ is $\neg\phi_1$: **return** $\neg$`IMPL_FREE` (ϕ_1)
ϕ is $\phi_1 \wedge \phi_2$: **return** `IMPL_FREE` $(\phi_1) \wedge$ `IMPL_FREE` (ϕ_2)
ϕ is $\phi_1 \vee \phi_2$: **return** `IMPL_FREE` $(\phi_1) \vee$ `IMPL_FREE` (ϕ_2)
ϕ is $\phi_1 \to \phi_2$: **return** `IMPL_FREE` $(\neg\phi_1 \vee \phi_2)$
end case
end function

There are quite a few other solutions possible. For example, we could have optimized this code in the last case by saying

return $\neg$(`IMPL_FREE` ϕ_1) $\vee$ (`IMPL_FREE` ϕ_2)

which would save us a couple of computation steps. Furthermore, it is possible to overlap the patterns of the first two cases by returning ϕ if ϕ is a literal in the first clause. Notice how the two clauses agree in this case. Such a programming style, while correct, it not recommended as it makes it harder to read somebody else's code.

9. We use our algorithm `IMPL_FREE` together with the functions `NNF` and `CNF` to compute `CNF(NNF(IMPL_FREE` ϕ`))`, where ϕ is the formula

$\neg(p \to (\neg(q \wedge (\neg p \to q))))$. In the solution below we skipped some of the obvious and tedious intermediate computation steps. Removing all implications results in:

$$\begin{aligned}
\texttt{IMPL_FREE}\ \phi &= \neg\texttt{IMPL_FREE}\ (p \to (\neg(q \wedge (\neg p \to q)))) \\
&= \neg(\neg p \vee \texttt{IMPL_FREE}\ (\neg(q \wedge (\neg p \to q)))) \\
&= \neg(\neg p \vee \neg(q \wedge \texttt{IMPL_FREE}\ (\neg p \to q))) \\
&= \neg(\neg p \vee \neg(q \wedge (\neg\neg p \vee q)))
\end{aligned}$$

Computing the corresponding negation normalform yields:

$$\begin{aligned}
\texttt{NNF}\ \neg(\neg p \vee \neg(q \wedge (\neg\neg p \vee q))) &= (\texttt{NNF}\ \neg\neg p) \wedge (\texttt{NNF}\ \neg\neg(q \wedge (\neg\neg p \vee q))) \\
&= (\texttt{NNF}\ p) \wedge (\texttt{NNF}\ (q \wedge (\neg\neg p \vee q))) \\
&= p \wedge (q \wedge (\texttt{NNF}\ (\neg\neg p \vee q))) \\
&= p \wedge (q \wedge ((\texttt{NNF}\ \neg\neg p) \vee (\texttt{NNF}\ q))) \\
&= p \wedge (q \wedge (p \vee q)) \\
&= p \wedge q \wedge (p \vee q)
\end{aligned}$$

which is already in conjunctive normalform so we won't have to call `CNF` anymore. Notice that this formula is equivalent to the CNF $p \wedge q$.

15(a) The marking algorithm first marks r, q, and u through clauses of the form $\top \to \cdot$. Then it marks p through clause $r \to p$ and s through clause $u \to s$. The w never gets marked and so $p \wedge q \wedge w$ can never trigger a marking for $\bot$. Thus the algorithm determines that the formula is satisfiable.

15(g). Applying the marking algorithm to $(\top \to q) \wedge (\top \to s) \wedge (w \to \bot) \wedge (p \wedge q \wedge s \to \bot) \wedge (v \to s) \wedge (\top \to r) \wedge (r \to p)$, it marks q, s, and r in the first iteration of the while-loop. In the second iteration, p gets marked and in the third iteration an inconsistency is found: $(p \wedge q \wedge s \to \bot)$ is a subformula of the original formula and p, q, s are all marked. Thus, the Horn formula is not satisfiable.

17. (Bonus) Note that Horn formulas are already of the form $\psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_m$, where each ψ_i is of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_{i_k} \to q_i$. We know that the latter formula is equivalent to $\neg(p_1 \wedge p_2 \wedge \ldots \wedge p_{i_k}) \vee q_i$, i.e. it is equivalent to $\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_{i_k} \vee q_i$. Thus, we may convert any Horn formula into a CNF where each conjuction clause has at

most one positive literal in it. Note that this also covers some special cases such as $\top \to q \equiv \neg\top \vee q \equiv \bot \vee q \equiv q$.

EXERCISES 1.6 (p.90)

2. We illustrate the arguments for some of these rules.

- For example, if a $\wedge$-node has to evaluate to `T` in order for the overall formula to be satisfiable, then both its arguments have to evaluate to `T` in order for the overall formula to be satisfiable.
- If both arguments have to evaluate to `T` in order for the overall formula to be satisfiable, then the $\wedge$-node needs to evaluate to `T` as well for the overall formula to be satisfiable.
- If we know that the overall formula can only be satisfiable if an $\wedge$-node evaluates to `F` and one of its arguments evaluates to `T`, then we know that the overall formula can only be satisfiable if all this is true and the other argument evaluates to `F`.
- Etc.

4. There are many equivalent DAGs one could construct for this. We chose as a formula $\neg((p \vee q) \wedge (p \to r) \to r)$ which can be translated as $\neg(\neg p \wedge \neg q) \wedge (\neg(p \wedge \neg r) \wedge \neg r)$ into the SAT solver's adequate fragment. One can then draw the corresponding DAG and will see that the linear SAT solver computes a satisfiablity witness for which p and r evaluate to `F` and q evaluates to `T`.

11. A semi-formal argument could go like this: the linear solver will compute all constraints it can infer from the current permanent markings so the order of evaluation won't matter for it as all constraints will be found and so it will always detect a contradiction if present. The cubic solver may test nodes in a different order but if *any* test creates a new permanent constraint, the cubic solver will test again all unconstrained notes so the set of constraints can only increase. Moreover, a contradiction found by testing in some order will also be found by testing in some other order as the eventual discovery of constraints is independent of the order of tests by the argument just made.

2

Predicate logic

EXERCISES 2.1 (p.157)

1. (a) $\forall x(P(x) \to A(m,x))$

(b) $\exists x(P(x) \wedge A(x,m))$

(c) $A(m,m)$

(d) $\forall x(S(x) \to (\exists y(L(y) \wedge \neg B(x,y))))$, or, equivalently, $\neg\exists x(S(x) \wedge \forall y(L(y) \to B(x,y)))$.

(e) $\forall x(L(x) \to \exists y(S(y) \wedge \neg B(y,x)))$, or, equivalently, $\neg\exists y(L(y) \wedge \forall x(S(x) \to B(x,y)))$.

(f) $\forall x(L(x) \to \forall y(S(y) \to \neg B(y,x)))$, or, equivalently, $\forall x \forall y(L(x) \wedge S(y) \to \neg B(y,x))$.

3. (a) $\forall x(\texttt{Red}(x) \to \texttt{Inbox}(x))$

(b) $\forall x(\texttt{Inbox}(x) \to \texttt{Red}(x))$

(c) $\forall x(\texttt{Animal}(x) \to (\neg\texttt{Cat}(x) \vee \neg\texttt{Dog}(x)))$, or, equivalently, $\neg\exists x(\texttt{Animal}(x) \wedge \texttt{Cat}(x) \wedge \texttt{Dog}(x))$.

(d) $\forall x(\texttt{Prize}(x) \to \exists y(\texttt{Boy}(y) \wedge (\texttt{Win}(y,x))))$

(e) $\exists y(\texttt{Boy}(y) \wedge \forall x(\texttt{Prize}(x) \to \texttt{Win}(y,x)))$ Note carefully the difference between (d) and (e), which look so similar when expressed in natural language. Item (d) says that every prize was won by a boy, possibly a different one for each prize. But item (e) says it is the same boy that won all the prizes.

EXERCISES 2.2 (p.158)

1(a)ii. The string $f(x, g(y,z), d)$ is not a term since f requires two arguments, not three. Likewise, g requires three arguments, not two.

1(a)iii. $\surd$

1(c). 1. There is only one term of height 1 (without variables); it is simply d.

2. There are two terms of height 2, namely $f(d,d)$ and $g(d,d,d)$.
3. There are quite a few terms of height 3: the root node has to be f or g. Depending on that choice, we have 2 or 3 arguments. Since the overall term has to have height 3 it follows that at least one of these arguments has to be a term of height 2. Thus, we may list all these terms systematically:
   - $f(d, f(d,d))$
   - $f(f(d,d), d)$
   - $f(g(d,d,d), d)$
   - $f(d, g(d,d,d))$
   - $f(f(d,d), f(d,d))$
   - $f(g(d,d,d), g(d,d,d))$
   - $f(f(d,d), g(d,d,d))$
   - $f(g(d,d,d), f(d,d))$

   are all the terms with f as a root. The terms with g as a root are of the form $g(\#1, \#2, \#3)$, where at least one of the three arguments has height 2. Thus, $g(d,d,d)$ is not allowed, but other than that each argument can be d, $f(d,d)$, or $g(d,d,d)$. This gives us $3 \cdot 3 \cdot 3 - 1 = 26$ different terms of height 3. The first ten are
   - $g(d,d,g(d,d,d))$
   - $g(d,g(d,d,d),d)$
   - $g(g(d,d,d),d,d)$
   - $g(d,d,f(d,d))$
   - $g(d,f(d,d),d)$
   - $g(f(d,d),d,d)$
   - $g(d,f(d,d),f(d,d))$
   - $g(d,f(d,d),g(d,d,d))$
   - $g(d,g(d,d,d),f(d,d))$
   - $g(d,g(d,d,d),g(d,d,d))$

   and you are welcome to write down the remaining 16 terms if you wish. In conclusion, there are $1+2+8+26 = 37$ different terms already of height less than 4.

3(a). (i) $\surd$

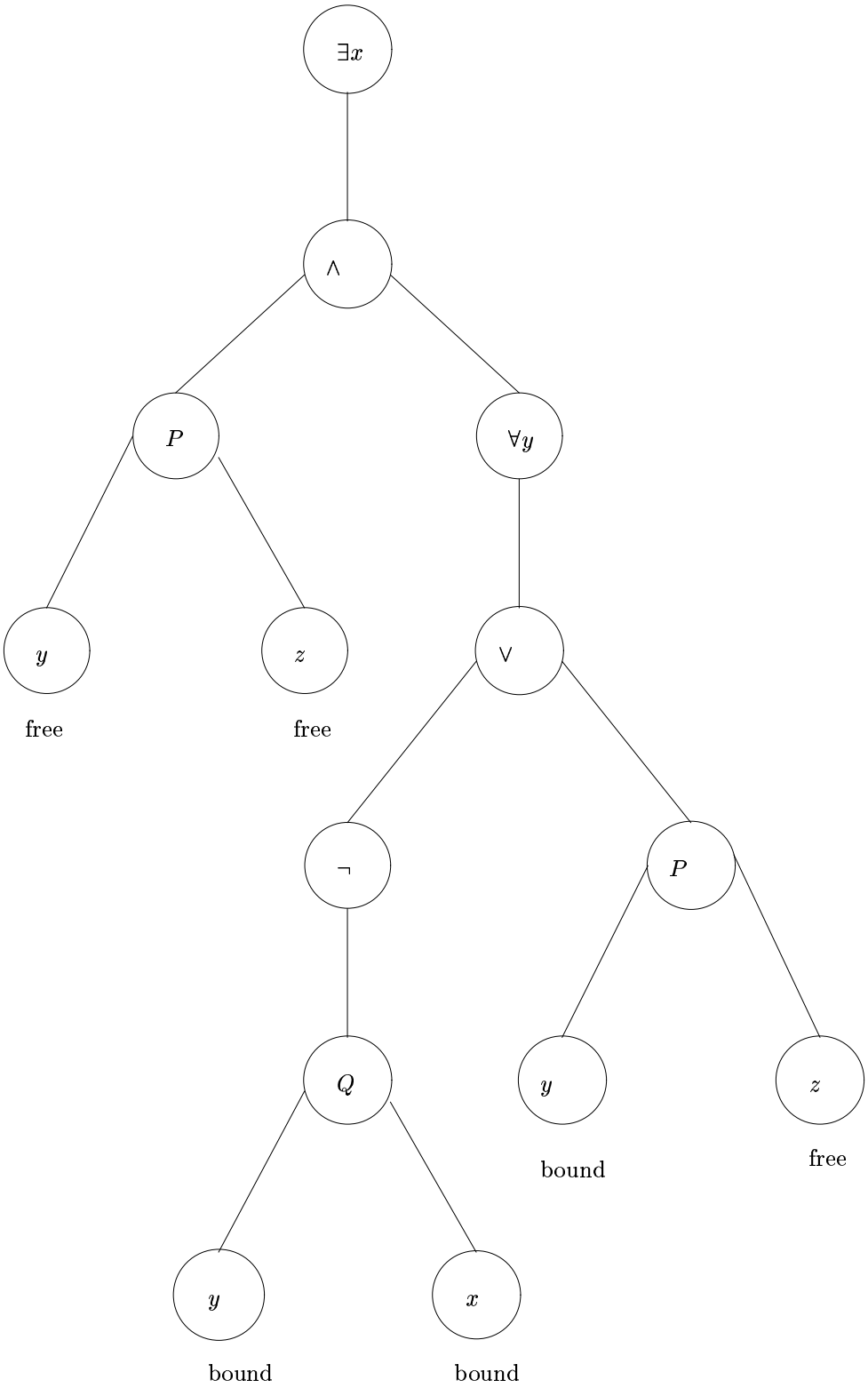
(ii) $\surd$

(iii) $\times$; $f(m)$ denotes a term, not a formula.

(iv) $\times$; in predicate logic, we can't nest predicates: the missing argument in $B(?, y)$ has to be a *term*, but $B(m, x)$ is a *formula*.

(v) $\times$; again, we can't nest predicates (B, S are predicates).
(vi) $\surd$
(vii) $\surd$
(viii) $\times$; again, we can't nest predicates.

4(a). The parsetree is


∃x
∧
P
∀y
y
z
∨
free
free
¬
P
Q
y
z
bound
free
y
x
bound
bound

4(b). From the parsetree of the previous item we see that all occurrences of z are free, all occurrences of x are bound, and the leftmost occurrence of y is free, whereas the other two occurrences of y are bound.

4(d).(i) – $\phi[w/x]$ is simply ϕ again since there are no free occurrences of x in ϕ that could be replaced by w;
– $\phi[w/y]$ is $\exists x(P(w,z) \wedge \forall y(\neg Q(y,x) \vee P(y,z)))$ since we replace the sole free occurrence of y with w;
– $\phi[f(x)/y]$ is $\exists x(P(f(x),z) \wedge \forall y(\neg Q(y,x) \vee P(y,z)))$ since we replace the sole free occurrence of y with $f(x)$;
– $\phi[g(y,z)/z]$ $\exists x(P(y,g(y,z)) \wedge \forall y(\neg Q(y,x) \vee P(y,g(y,z))))$ since we replace all (free) occurrences of z with $g(y,z)$.

(ii) All of them, for there are no free occurrences of x in ϕ to begin with.

(iii) The terms w and $g(y,z)$ are free for y in ϕ, since the sole free occurrence of y only has $\exists x$ as a quantifier above it. For that very reason, $f(x)$ is *not* free for y in ϕ, since the x in $f(x)$ would be captured by $\exists x$ in that substitution process.

4(f). Now, the scope of $\exists x$ is only the formula $P(y,z)$, since the inner quantifier $\forall x$ binds the x (and overrides the binding of $\exists x$ in the formula $\neg Q(x,x) \vee P(x,z)$).

EXERCISES 2.3 (p.160)

3(a). A formula $\phi_{=3}$ whose models are exactly those which have three distinct elements is

$$\exists x\, \exists y\, \exists z\, (((\neg(x=y) \wedge \neg(x=z)) \wedge \neg(y=x)) \\ \wedge \forall w\, (((w=x) \vee (w=y)) \vee (w=z))).$$

3(c). As in item 3 above, we may generally construct formulas $\phi_{=n}$ for each $n = 1, 2, 3, \ldots$ such that the models of $\phi_{=n}$ are exactly those with n distinct elements. Then a model is finite iff it satisfies $\phi_{=n}$ *for some* $n \geq 1$. Therefore,

$$\bigvee_{n \geq 1} \phi_{=n}$$

would do the trick, but, alas, this conjunction is *infinite* and not part of the syntax of predicate logic.

7(a). We prove the validity of $\forall x \forall y P(x,y) \vdash \forall u \, \forall v \, P(u,v)$ by

1	$\forall x \forall y P(x,y)$	assum
2	x_0	
3	y_0	
4	$\forall y \, P(x_0,y)$	$\forall x$ e 1
5	$P(x_0,y_0)$	$\forall y$ e 4
6	$\forall v \, P(x_0,v)$	$\forall v$ i 3–5
7	$\forall u \, \forall v \, P(u,v)$	$\forall u$ i 2–6

7(c). We prove the validity of $\exists x \, \forall y \, P(x,y) \vdash \forall y \, \exists x \, P(x,y)$ by

1	$\exists x \, \forall y \, P(x,y)$	assum
2	y_0	
3	x_0	
4	$\forall y \, P(x_0,y)$	assum
5	$P(x_0,y_0)$	$\forall y$ e 4
6	$\exists x \, P(x,y_0)$	$\exists x$ i 5
7	$\exists x P(x,y_0)$	$\exists x$ e 1, 3–6
8	$\forall y \, \exists x \, P(x,y)$	$\forall y$ i 2–7

Note that we have to open a y_0-box first, since we mean to show a formula of the form $\forall y \, \psi$. Then we could open an x_0-box to use $\exists x$ e.

9(a). We prove the validity of $\exists x \, (S \to Q(x)) \vdash S \to \exists x \, Q(x)$ by

1	$\exists x \, (S \to Q(x))$	prem
2	S	assum
3	x_0	
4	$S \to Q(x_0)$	assum
5	$Q(x_0)$	$\to$e 4, 2
6	$\exists x \, Q(x)$	$\exists x$ i 5
7	$\exists x \, Q(x)$	$\exists x$ e 1, 3–6
8	$S \to \exists x \, Q(x)$	$\to$i 2–7

9(d). We prove the validity of $\forall x\, P(x) \to S \vdash \exists x\, (P(x) \to S)$ by

1	$\forall x\, P(x) \to S$	prem
2	$\neg\exists x\, (P(x) \to S)$	assum
3	x_0	
4	$\neg P(x_0)$	assum
5	$P(x_0)$	assum
6	$\bot$	$\neg$e 5, 4
7	S	$\bot$e 6
8	$P(x_0) \to S$	$\to$i 5–7
9	$\exists x\, (P(x) \to S)$	$\exists x$ i 8
10	$\bot$	$\neg$e 9, 2
11	$\neg\neg P(x_0)$	$\neg$i 4–10
12	$P(x_0)$	$\neg\neg$e 11
13	$\forall x\, P(x)$	$\forall x$ i 3–12
14	S	$\to$e 1, 13
15	$P(t)$	assum
16	S	copy 14
17	$P(t) \to S$	$\to$i 15–16
18	$\exists x\, (P(x) \to S)$	$\exists x$ i 17
19	$\bot$	$\neg$e 18, 2
20	$\neg\neg\exists x\, (P(x) \to S)$	$\neg$i 2–19
21	$\exists x\, (P(x) \to S)$	$\neg\neg$e 20

9(k). We prove the validity of $\forall x\,(P(x) \wedge Q(x)) \vdash \forall x\,P(x) \wedge \forall x\,Q(x)$ by

1	$\forall x\,(P(x) \wedge Q(x))$	prem
2	x_0	
3	$P(x_0) \wedge Q(x_0)$	$\forall x$ e 1
4	$P(x_0)$	$\wedge \mathrm{e}_1$ 3
5	$\forall x\,P(x)$	$\forall x$ i 2–4
6	x_0	
7	$P(x_0) \wedge Q(x_0)$	$\forall x$ e 1
8	$Q(x_0)$	$\wedge \mathrm{e}_2$ 7
9	$\forall x\,Q(x)$	$\forall x$ i 6–8
10	$\forall x\,P(x) \wedge \forall x\,Q(x)$	$\wedge$i 5, 9

9(l). We prove the validity of $\forall x\,P(x) \vee \forall x\,Q(x) \vdash \forall x\,(P(x) \vee Q(x))$ by

1	$\forall x\,P(x) \vee \forall x\,Q(x)$			prem
2	x_0			
3	$\forall x\,P(x)$	assum	$\forall x\,Q(x)$	assum
4	$P(x_0)$	$\forall x$ e 3	$Q(x_0)$	$\forall x$ e 3
5	$P(x_0) \vee Q(x_0)$	$\vee \mathrm{i}_1$ 4	$P(x_0) \vee Q(x_0)$	$\vee \mathrm{i}_2$ 4
6	$P(x_0) \vee Q(x_0)$			$\vee$e 1, 3–5, 3–5
7	$\forall x\,(P(x) \vee Q(x))$			$\forall x$ i 2–6

9(m). We prove the validity of $\exists x\,(P(x) \wedge Q(x)) \vdash \exists x\,P(x) \wedge \exists x\,Q(x)$ by

1	$\exists x\,(P(x) \wedge Q(x))$	prem
2	x_0	
3	$P(x_0) \wedge Q(x_0)$	assum
4	$P(x_0)$	$\wedge \mathrm{e}_1$ 3
5	$\exists x\,P(x)$	$\exists x$ i 4
6	$Q(x_0)$	$\wedge \mathrm{e}_2$ 3
7	$\exists x\,Q(x)$	$\exists x$ i 6
8	$\exists x\,P(x) \wedge \exists x\,Q(x)$	$\wedge$i 5, 7
9	$\exists x\,P(x) \wedge \exists x\,Q(x)$	$\exists x$ e 1, 2–8

9(n). We prove the validity of $\exists x\, F(x) \vee \exists x\, G(x) \vdash \exists x\, (F(x) \vee G(x))$ by

1	$\exists x\, F(x) \vee \exists x\, G(x)$				prem
2	$\exists x\, F(x)$	assum		$\exists x\, G(x)$	assum
3	x_0			x_0	
4	$F(x_0)$	assum		$G(x_0)$	assum
5	$F(x_0) \vee G(x_0)$	$\vee \mathrm{i}_1$ 4		$F(x_0) \vee G(x_0)$	$\vee \mathrm{i}_2$ 4
6	$\exists x\, (F(x) \vee G(x))$	$\exists x$ i 5		$\exists x\, (F(x) \vee G(x))$	$\exists x$ i 5
7	$\exists x\, (F(x) \vee G(x))$	$\exists x$ e 2, 3–6		$\exists x\, (F(x) \vee G(x))$	$\exists x$ e 2, 3–6
8	$\exists x\, (F(x) \vee G(x))$				$\vee$e 1, 2–7, 2–7

9(p). We prove the validity of $\neg\forall x \neg P(x) \vdash \exists x P(x)$ by

1	$\neg\forall x\, \neg P(x)$	prem
2	$\neg\exists x\, P(x)$	assum
3	x_0	
4	$P(x_0)$	assum
5	$\exists x\, P(x)$	$\exists x$ i 4
6	$\bot$	$\neg$e 5, 2
7	$\neg P(x_0)$	$\neg$i 4–6
8	$\forall x\, \neg P(x)$	$\forall x$ i 3–7
9	$\bot$	$\neg$e 8, 1
10	$\exists x\, P(x)$	RAA 2–9

9(q). We prove the validity of $\forall x\, \neg P(x) \vdash \neg\exists x P(x)$ by

1	$\forall x\, \neg P(x)$	prem
2	$\exists x\, P(x)$	assum
3	x_0	
4	$P(x_0)$	assum
5	$\neg P(x_0)$	$\forall x$ e 1
6	$\bot$	$\neg$e 4, 5
7	$\bot$	$\exists x$ e 2, 3–6
8	$\neg\exists x P(x)$	$\neg$i 2–7

9(r). We prove the validity of $\neg\exists x\, P(x) \vdash \forall x\, \neg P(x)$ by

1		$\neg\exists x\, P(x)$	prem
2	x_0		
3		$P(x_0)$	assum
4		$\exists x\, P(x)$	$\exists x$ i 3
5		$\bot$	$\neg$e 4, 1
6		$\neg P(x_0)$	$\neg$i 3–5
7		$\forall x\, \neg P(x)$	$\forall x$ i 2–6

Note that we mean to show $\forall x \neg P(x)$, so the x_0-box has to show the instance $\neg P(x_0)$ which is achieved via a proof by contradiction.

11(a). We prove the validity of $P(b) \vdash \forall x\, ((x = b) \to P(x))$ by

1		$P(b)$	prem
2	x_0		
3		$x_0 = b$	assum
4		$P(x_0)$	=e 3, 1
5		$(x_0 = b) \to P(x_0)$	$\to$i 3–4
6		$\forall x\, ((x = b) \to P(x))$	$\forall x$ i 2–5

11(c). We prove the validity of $\exists x \exists y (H(x, y) \vee H(y, x)), \neg\exists x H(x, x) \vdash$

$\exists x \exists y \neg (x = y)$ by

1	$\exists x\, \exists y\, (H(x,y) \vee H(y,x))$	prem
2	$\neg \exists x\, H(x,x)$	prem
3	x_o	
4	$\exists y\, (H(x_0,y) \vee H(y,x_0))$	assum
5	y_0	
6	$H(x_0,y_0) \vee H(y_0,x_0)$	assum
7	$x_0 = y_0$	assum
8	$H(x_0,y_0)$	assum
9	$H(y_0,y_0)$	=e 7, 8
10	$\exists x\, H(x,x)$	$\exists x$ i 9
11	$\bot$	$\neg$e 10, 2
12	$H(y_0,x_0)$	assum
13	$H(y_0,y_0)$	=e 7, 12
14	$\exists x\, H(x,x)$	$\exists x$ i 13
15	$\bot$	$\neg$e 14, 2
16	$\bot$	$\vee$e 6, 8−11, 12−15
17	$\neg(x_0 = y_0)$	$\neg$i 7−16
18	$\exists y\, \neg(x_0 = y)$	$\exists y$ i 17
19	$\exists x\, \exists y\, \neg(x = y)$	$\exists x$ i 18
20	$\exists x\, \exists y\, \neg(x = y)$	$\exists y$ e 4, 5−19
21	$\exists x\, \exists y\, \neg(x = y)$	$\exists x$ e 1, 3−20

12. We prove the validity of $S \to \forall x\, Q(x) \vdash \forall x\, (S \to Q(x))$ by

1	$S \to \forall x\, Q(x)$	prem
2	x_0	
3	$\quad S$	assum
4	$\quad \forall x\, Q(x)$	$\to$e 1, 3
5	$\quad Q(x_0)$	$\forall x$ e 4
6	$S \to Q(x_0)$	$\to$i 3–5
7	$\forall x\, (S \to Q(x))$	$\forall x$ i 2–6

13(a). We show the validity of

$$\forall x\, P(a, x, x), \forall x\, \forall y\, \forall z\, (P(x, y, z) \to P(f(x), y, f(z))) \vdash P(f(a), a, f(a))$$

by

1	$\forall x\, P(a, x, x)$	prem
2	$\forall x\, \forall y\, \forall z\, (P(x, y, z) \to P(f(x), y, f(z)))$	prem
3	$P(a, a, a)$	$\forall x$ e 1
4	$\forall y\, \forall z\, (P(a, y, z) \to P(f(a), y, f(z)))$	$\forall x$ e 2
5	$\forall z\, (P(a, a, z) \to P(f(a), a, f(z)))$	$\forall y$ e 4
6	$P(a, a, a) \to P(f(a), a, f(a))$	$\forall z$ e 5
7	$P(f(a), a, f(a))$	$\to$e 6, 3

13(b). We show the validity of

$$\forall x P(a, x, x), \forall x\, \forall y\, \forall z\, (P(x, y, z) \to P(f(x), y, f(z))) \vdash \exists z\, P(f(a), z, f(f(a)))$$

by

1	$\forall x P(a, x, x)$	prem
2	$\forall x\, \forall y\, \forall z\, (P(x, y, z) \to P(f(x), y, f(z)))$	prem
3	$P(a, f(a), f(a))$	$\forall x$ e 1
4	$\forall y\, \forall z\, (P(a, y, z) \to P(f(a), y, f(z)))$	$\forall x$ e 2
5	$\forall z\, (P(a, f(a), z) \to P(f(a), f(a), f(z)))$	$\forall y$ e 4
6	$P(a, f(a), f(a)) \to P(f(a), f(a), f(f(a)))$	$\forall z$ e 5
7	$P(f(a), f(a), f(f(a)))$	$\to$e 6, 3
8	$\exists z\, P(f(a), z, f(f(a)))$	$\exists z$ i 7

Note that we just had to add one more line to the proof of the previous item and instantiate x, y and z with $f(a)$ instead of a in lines 3, 4 and 5, respectively.

13(c). We prove the validity of

$$\forall y\, Q(b,y), \forall x\, \forall y\, (Q(x,y) \to Q(s(x),s(y))) \vdash \exists z\, (Q(b,z) \wedge Q(z,s(s(b))))$$

by

1	$\forall y\, Q(b,y)$	prem
2	$\forall x\, \forall y\, (Q(x,y) \to Q(s(x),s(y)))$	prem
3	$\forall y\, (Q(b,y) \to Q(s(b),s(y)))$	$\forall x$ e 2
4	$Q(b,s(b)) \to Q(s(b),s(s(b)))$	$\forall y$ e 3
5	$Q(b,s(b))$	$\forall x$ e 1
6	$Q(s(b),s(s(b)))$	$\to$e 4, 5
7	$Q(b,s(b)) \wedge Q(s(b),s(s(b))$	$\wedge$i 5, 6
8	$\exists z\, (Q(b,z) \wedge Q(z,s(s(b)))$	$\exists x$ i 7

EXERCISES 2.4 (p.163)

1. The truth value of $\forall x\, \forall y\, Q(g(x,y),g(y,y),z)$ depends only on the values that valuations assign to its free variables, i.e to the occurrence of z. We choose A to be the set of integers, $g^{\mathcal{M}}(a,a')$ is the result of subtracting a' from a, and a triple of integers (a,b,c) is in $Q^{\mathcal{M}}$ if, and only if, c equals the product of a and b. That way $g(y,y)$ is interpreted as 0 and, consequently, our formula says that 0 equals the value assigned to z by our valuation. So for $l(z) \stackrel{\text{def}}{=} 0$ the formula holds in our model, whereas for $l'(z) \stackrel{\text{def}}{=} 1$ it is false.

5(a). The formula $\forall x\, \forall y\, \exists z\, (R(x,y) \to R(y,z))$ is not true in the model $\mathcal{M}$: for example, we have $(b,a) \in R^{\mathcal{M}}$, but there is no $m \in A$ with $(a,m) \in R^{\mathcal{M}}$ contrary to what the formula claims. Thus, we may "defeat" this formula by choosing b for x and a for y to construct a counter-witness to the truth of this formula over the given model.

5(b). The formula $\forall x\, \forall y\, \exists z\, (R(x,y) \to R(y,z))$ *is* true in the model $\mathcal{M}'$: we may list all elements of R in a "cyclic" way as (a,b) (b,c) (c,b), the cycle being the last two pairs. Thus, for any choice of x and y we can find some z so that the implication $R(x,y) \to R(y,z)$ is true.

6. • We choose a model with A being the set of integers. We define $(n,m) \in P^{\mathcal{M}}$ if, and only if, n is less than or equal to m ($n \leq m$). Evidently,

this interpretation of P is reflexive (every integer is less than or equal to itself) and transitive: $n \leq m$ and $m \leq k$ imply $n \leq k$. However, $2 \leq 3$ and $3 \not\leq 2$ show that this interpretation cannot be symmetric.

- We choose as set A the sons of J. S. Bach. We interpret $P(x, y)$ as "x is a brother of y". Clearly, this relation is transitive and symmetric, but not reflexive.
- We define $A \stackrel{\text{def}}{=} \{a, b, c\}$ and

$$P^{\mathcal{M}} \stackrel{\text{def}}{=} \{(a,a), (b,b), (c,c), (a,c), (a,b), (b,a), (c,a)\}.$$

Note that this interpretation is reflexive and symmetric. We also have that (b, a) and (a, c) are in $P^{\mathcal{M}}$. Thus, we would need $(b, c) \in P^{\mathcal{M}}$ to secure transitivity of $P^{\mathcal{M}}$. Since this is not the case, we infer that this interpretation of P is not transitive.

8. To show the semantic entailment $\forall x\, P(x) \vee \forall x\, Q(x) \vDash \forall x\, (P(x) \vee Q(x))$ let $\mathcal{M}$ be any model such that $\mathcal{M} \vDash \forall x\, P(x) \vee \forall x\, Q(x)$. Then $\mathcal{M} \vDash \forall x\, P(x)$ or $\mathcal{M} \vDash \forall x\, Q(x)$. Thus, either all elements of A are in $P^{\mathcal{M}}$, or they are all in $Q^{\mathcal{M}}$. In any case, every element of A is in the union $P^{\mathcal{M}} \cup Q^{\mathcal{M}}$. Therefore, $\mathcal{M} \vDash \forall x\, (P(x) \vee Q(x))$ follows.

9(b). To see that $\phi \wedge \eta \vDash \psi$ does not imply $\phi \vDash \psi$ and $\eta \vDash \psi$ in general consider the special case where ψ is just ϕ again. Then $\phi \wedge \eta \vDash \phi$ and $\phi \vDash \phi$ are clearly the case, but we cannot guarantee that $\eta \vDash \phi$ holds as well. For example, we could pick η to be $\exists x\, P(x)$ and ϕ to be $\forall x\, P(x)$.

11(b). The collection of formulas $\forall x\, \neg S(x, x)$, $\forall x\, \exists y\, S(x, y)$, $\forall x\, \forall y\, \forall z\, ((S(x, y) \wedge S(y, z)) \rightarrow S(x, z))$ says that the interpretation of S is irreflexive, serial (see Chapter 5), and transitive. There are plenty of such relations around. For example, consider $S(x, y)$ to be interpreted over the natural numbers as meaning "x is strictly less than y" $(x < y)$. Clearly, this is irreflexive and transitive. It is also serial since $n < n + 1$ for all natural numbers n.

11(d). The collection of formulas $\exists x\, S(x, x)$, $\forall x\, \forall y\, (S(x, y) \rightarrow (x = y))$ says that S should be interpreted as equality and that at least one element is equal to itself. But since the interpretation of equality over any model is reflexive ($(a, a) \in =^{\mathcal{M}}$ for all $a \in A$) this is true for all models which interpret equality in the standard way (and where A is non-empty, which we assumed in our definition of models).

12(b) This is valid. To prove it, use $\exists y\, \mathrm{i}$ with dummy variable y_0 at the toplevel and within its box use $\rightarrow \mathrm{i}$ to show $(\forall x P(x)) \rightarrow P(y_0)$ and then close the $\exists y\, \mathrm{i}$ box to conclude the desired formula.

12(g) This formula claims that any relation that is serial and anti-symmetric has no minimal element. Let the model be the set of natural numbers with where S is interpreted as "less than or equal." Then this relation is serial and anti-symmetric but 0 is a minimal element. So this formula is not valid.

EXERCISES 2.5 (p.164)

1(b). We interpret the sequent

$$\forall x(P(x) \to R(x)), \forall x(Q(x) \to R(x)) \vdash \exists x(P(x) \wedge Q(x))$$

by

$$\begin{aligned} P(x) &: \text{“}x \text{ is a cat.”} \\ Q(x) &: \text{“}x \text{ is a dog.”} \\ R(x) &: \text{“}x \text{ is an animal.”} \end{aligned}$$

Then $\forall x(P(x) \to R(x))$ and $\forall x(Q(x) \to R(x))$ are obviously true, but $\exists x(P(x) \wedge Q(x))$ is false, despite recent efforts in microbiology.

1(d). We interpret the sequent $\forall x \exists y S(x,y) \vdash \exists y \forall x S(x,y)$ over the integers by

$$S(x,y) \quad : \quad \text{“}y \text{ equals } 2 \cdot x\text{.”}$$

Clearly, every x has a y, namely $2 \cdot x$, such that $S(x,y)$ holds. But there is no integer y which equals $2 \cdot x$ *for all choices of* x.

1(f). For the sequent $\exists x\,(\neg P(x) \wedge Q(x)) \vdash \forall x\,(P(x) \to Q(x))$ no proof is possible since we can interpret P and Q over the integers such that $P(x)$ is saying "x is odd" and $Q(x)$ is saying that "x is even". In this model we have $\mathcal{M} \vDash \exists x\,(\neg P(x) \wedge Q(x))$, e.g. take 2 as a value of x, but we cannot have $\mathcal{M} \vDash \forall x\,(P(x) \to Q(x))$ since not all odd numbers are even (in fact, none of them are!).

1(g). For the sequent $\exists x\,(\neg P(x) \vee \neg Q(x)) \vdash \forall x\,(P(x) \vee Q(x))$ no proof is possible since we may interpret P and Q over the same domain of integers, but now $P(x)$ says that "x is divisible by 2", whereas $Q(x)$ says that "x is divisible by 3". Then we have $\mathcal{M}' \vDash \exists x\,(\neg P(x) \vee \neg Q(x))$ for this model, e.g. take 9 as the value of x; however, we cannot have $\mathcal{M}' \vDash \forall x\,(P(x) \vee Q(x))$ since not all integers are divisible by 2 or 3 (e.g. choose x to be 13).

EXERCISES 2.6 (p.165)

1(a) The formula $\exists P\phi$ was meant to be the one that specifies unreachability. Therefore, this does not hold here as s_1 is reachable from s_3 through the path $s_3 \to s_0 \to s_1$.

4. There are infinitely many such paths (you are welcome to think about the degree of infinity here). For example, we can travel from s_3 to s_0 and then cycle between s_1 and s_0 for any finite number of times before we travel from s_1 to s_2.

5(a) We can specify such a formula by $\exists P(\forall x \forall y(P(x,y) \to R(x,y))) \wedge (\forall x P(x,x)) \wedge (\forall x \forall y(P(x,y) \to P(y,x)))$.

5(e) We specify this by saying there is some P which contains exactly all pairs that have a directed R-path of length 2 and that is transitive. So this may read as $\exists P(\forall x \forall y(P(x,y) \Leftrightarrow \exists z(R(x,z) \wedge R(z,y)))) \wedge (\forall x \forall y \forall z(P(x,y) \wedge P(y,z) \to P(x,z)))$.

6. We use an instance of the law of the excluded middle. Either A is in A or it isn't. If A is in A, then by definition of A it is not in A, contradiction. If A is not in A, then by definition of A it is actually of member of A, a contradiction. So no matter what case, we derive a contradiction.

8(a) This formula claims that there is a binary relation P with the property that any P-edge excludes its reverse P-edge and that any R-edge has a reverse P-edge. This cannot be in the given model. For example, there are R-edges from s_1 to s_0 and vice versa so they would imply P-edges from s_0 to s_1 and vice versa, respectively. But this contradicts the requirement on P-edges, namely that a P-edge from s_0 to s_1 excludes the possibility of a P-edge from s_1 to s_0.

9(b) We can achieve this by using universal second-order logic to quantify over all transitive relations that contain R and to then make the desired statement about all such transitive relations: $\forall P(\forall x \forall y(R(x,y) \to P(x,y))) \wedge (\forall x \forall y \forall z(P(x,y) \wedge P(y,z) \to P(x,z))) \to (\forall x P(x,x))$.

9(e) The equivalence relation that identifies no two distinct elements is maximal in this sense and so $\forall x \forall y R(x,y) \leftrightarrow (x = y)$ is a correct encoding in predicate logic.

EXERCISES 2.7 (p.166)

1(a) Both `Person_1` and `Person_2` are members of `cast`, but `Person_2` loves `Person_0` which is `alma` and the `Person_1` loves `alma` and `Person_2`. Therefore this is a counterexample to `OfLovers` as no person in this graph loves him or herself.

1(c). A model with only two persons is such that one of them has to be `alma` and then the other person has to love `alma`. But the only way to then get a counterexample is to then have another love relationship which, necessarily, has to involve self-love.

2(b)
```
fun SevenNodesEachHavingFiveReachableNodes(G : Graph) {
  with G {
    # nodes = 7
    all n : nodes | # n.^edges = 5
  }
} run SevenNodesEachHavingFiveReachableNodes for 7 but 1 Graph
```

4(a-e) Here is a file `ColoredGraphs.als` with a possible solution:

```
module ColoredGraphs

sig Color {}
sig Node {
  color : Color
}

sig AboutColoredGraphs {
  nodes : set Node,
  edges : nodes -> nodes
}{ edges = ~ edges && all x : nodes, y : x.edges | not x.color = y.color }

fun TwoColorable(g : AboutColoredGraphs) {
  with g {
    # nodes.color = 2
    # nodes >= 3
    all x : nodes | some x.edges -- to see more interesting graphs
  }
} run TwoColorable for 4 but 1 AboutColoredGraphs


fun ThreeColorable(g : AboutColoredGraphs) {
  with g {
    # nodes.color = 3
    # nodes >= 3
    all x : nodes | some x.edges
  }
} run ThreeColorable for 4 but 1 AboutColoredGraphs
```

```
fun FourColorable(g : AboutColoredGraphs) {
  with g {
    # nodes.color = 4
    # nodes >= 3
    all x : nodes | some x.edges
  }
} run FourColorable for 4 but 1 AboutColoredGraphs

-- we cannot specify that a graph is 3-colorable but not
2-colorable, for in order to do this -- we need to say ``there is
a 3-coloring'' that works for this graph and ``all 2-colorings''
-- don't work; the presence of the existential and universal
quantifiers over relations are -- what makes this not expressible
in Alloy or any other tool that attempts to reduce this to -- SAT
solving for propositional logic
```

5(a-f) Here is a file `KripkeModels.als` with a possible solution:

```
module KripkeModel sig Prop {}

sig State {
  labels : set Prop -- those propositions that are true at this state
}

sig StateMachine {
  states : set State,
  init, final : set states,
  next : states -> states
}{ some init }

fun Reaches(m : StateMachine, s : set State) {
  with m {
    s = init.*next
  }
} run Reaches for 3 but 1 StateMachine

fun DeadlockFree(m : StateMachine) {
  with m {
    init.*next & { x : states | no x.next } in final
```

```
  }
} run DeadlockFree for 3 but 1 StateMachine

fun Deterministic(m : StateMachine) {
  with m {
    all x : init.*next | sole x.next
  }
} run Deterministic for 3 but 1 StateMachine

fun Reachability(m : StateMachine, p : Prop) {
  with m {
    some init.*next & { s : states | p in s.labels }
  }
} run Reachability for 3 but 1 StateMachine

fun Liveness(m : StateMachine, p : Prop) {
  with m {
    all x : init.*next | some x.*next & { s : states | p in s.labels }
  }
} run Liveness for 3 but 1 StateMachine

assert Implies {
  all m : StateMachine, p : Prop | Liveness(m,p) => Reachability(m,p)
} check Implies for 3 but 1 StateMachine

assert Converse {
  all m : StateMachine, p : Prop | Reachability(m,p) => Liveness(m,p)
} check Converse for 3 but 1 StateMachine

fun SimulationForFiveDf(m : StateMachine) {
  with m {
    # states = 3
    some x : states | not sole x.next
    all x : states | no x.next => x in final
    all x,y : states | { p : Prop | p in x.labels } =
                       { p : Prop | p in y.labels } => x = y
  }
} run SimulationForFiveDf for 3 but 1 StateMachine
```

6(a) We write a signature for groups along with the necessary constraints (group axioms):

```
sig Group {
  elements: set Element,
  unit: elements,
  mult: elements -> elements ->! elements,
  inv: elements ->+ elements
}{ all a,b,c: elements | c.((b.(a.mult)).mult) =
(c.(b.mult)).(a.mult)
   all a: elements | a = unit.(a.mult) && a = a.(unit.mult)
   all a: elements | unit = a.((a.inv).mult) && (a.inv).(a.mult) = unit
}
```

where `elements` models G, `unit` models e, and `mult` models $\star$. Please note

- that the three axioms for multiplication are encoded as constraints attached to the signature; we display multiplication in a "curried" form);
- the role of `!` which ensures that any two group elements have a unique result of their multiplication; and
- the role of `+` stating that each element has at least one inverse.

6(b) The following `fun`-statement generates a group with three elements:

```
fun AGroup(G: Group) {
  # G.elements = 3
} run AGroup for 3 but 1 Group
```

6(c) We declare

```
assert Inverse {
  all G: Group, e: G.elements | sole e.(G.inv)
} check Inverse for 5 but 1 Group
```

and no counter-example is found within that scope; `sole S` denotes that `S` contains at most one element. The small-scope hypothesis therefore suggests that inverses are unique in all finite groups. In this case, the small-scope hypothesis got it right: inverses are unique in groups, even in infinite ones.

6(d) i. We declare

```
fun Commutes (G: Group) {
  all a,b: G.elements | a.(b.(G.mult)) = b.(a.(G.mult))
} run Commutes for 3 but 1 Group

assert Commutative {
  all G: Group | Commutes(G)
} check Commutative for 5 but 1 Group
```

ii. Analyzing the assertion above we find no solution. The small-scope hypothesis therefore suggests that all finite groups are commutative. This time, the small-scope hypothesis got it wrong! There are finite groups that are not commutative.

iii. In fact, increasing the scope from 5 to 6 reveals a violation to our goal. Please run this analysis yourself and inspect the navigable tree to determine where and how commutativity is broken.

6(e) Yes, the assertions are formulas that make a claim about *all* groups. So a counter-example exists iff it exists for a single group. We already achieved the restriction to one group with the `but 1 Group` in the `check` and `run` directives.

3

Verification by model checking

EXERCISES 3.1 (p.245)

2(b). • $q_3, q_4, q_3, q_2, q_2, q_2, \ldots$
- No, because the path $q_3, q_1, q_2, q_2, \ldots$ does not satisfy a U b.

2(e). • $q_3, q_4, q_3, q_2, q_2, q_2, \ldots$
- No, because the path $q_3, q_1, q_2, q_2, \ldots$ does not satisfy $X(a \wedge b)$.

3. For each equivalence, there are two directions to prove. We prove the two directions for the first equivalence.

- ϕ U $\psi \rightarrow \phi$ W $\psi \wedge$ Fψ. Suppose $\pi \vDash \phi$ U ψ. Then take i such that $\pi^i \vDash \psi$ and $\pi^j \vDash \phi$ for all $j < i$. These facts are enough to prove $\pi \vDash \phi$ W ψ and $\pi \vDash$ Fψ.
- ϕ W $\psi \wedge$ F$\psi \rightarrow \phi$ U ψ. Suppose $\pi \vDash \phi$ W $\psi \wedge$ Fψ. Then $\pi \vDash$ Fψ, from which we derive the existence of i such that $\pi^i \vDash \psi$. Take the smallest such i. From $\pi \vDash \phi$ W ψ, it follows that $\pi^j \vDash \phi$ for all $j < i$. This shows that $\pi \vDash \phi$ U ψ.

5. The subformulas are p,$\neg p$, r, Fr, q, $\neg q$, G$\neg q$, $\neg r$, q W $\neg r$, F$r \vee G\neg q$, F$r \vee$ G$\neg q \rightarrow q$ W $\neg r$, $\neg p$ U (F$r \vee$ G$\neg q \rightarrow q$ W $\neg r$).

8. Add the clauses:

ϕ is Xϕ_1: return X NNF(ϕ_1)
ϕ is $\neg$Xϕ_1: return X NNF($\neg\phi_1$)
ϕ is Fϕ_1: return F NNF(ϕ_1)
ϕ is $\neg$Fϕ_1: return G NNF($\neg\phi_1$)
ϕ is Gϕ_1: return G NNF(ϕ_1)
ϕ is $\neg$Gϕ_1: return F NNF($\neg\phi_1$)
ϕ is ϕ_1 U ϕ_2: return NNF(ϕ_1) U NNF(ϕ_2)
ϕ is $\neg(\phi_1$ U $\phi_2)$: return NNF($\neg\phi_1$) R NNF($\neg\phi_2$)
ϕ is ϕ_1 R ϕ_2: return NNF(ϕ_1) R NNF(ϕ_2)
ϕ is $\neg(\phi_1$ R $\phi_2)$: return NNF($\neg\phi_1$) U NNF($\neg\phi_2$)

ϕ is ϕ_1 W ϕ_2: return NNF(ϕ_1) W NNF(ϕ_2)
ϕ is $\neg(\phi_1$ W $\phi_2)$: return NNF($\neg(\phi_2$ R $(\phi_1 \vee \phi_2))$).

EXERCISES 3.3 (p.247)

1(a). The two states `req.busy` and `!req.busy` satisfy F`status` = `busy` since they satisfy `status = busy`. The state `req.ready` also satisfies this formula since all its successors (the two states above) satisfy it. Since the only two states which satisfy `req` are among the above, we infer that all states satisfy G(`request` $\rightarrow$ F`status` = `busy`). In particular, this will be true for all states which are reachable from some initial state (which is this case amounts to the same thing as "all states").

3. The hint given in the remark is incorrect as it stands. To make it correct, we need two additional assumptions:

- The `msg_chan.forget` and `ack_chan.forget` are both false;
- The components are composed synchronously (i.e. without the `process` keyword).

Under these assumptions and the assumptions about `msg_chan.output1` and `mack_chan.output` being constant 0, the system is deterministic.

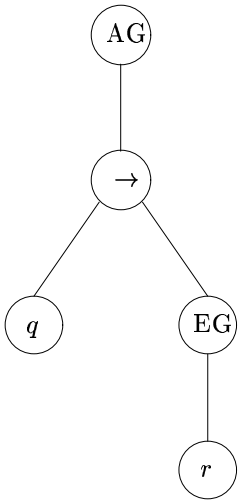
The first few states are given as follows:

var	alias	s_0	s_1	s_2	...
`s.st`		`sending`	`sending`		
`s.message1`	`msg_chan.input1`	0	0		
`s.message2`	`msg_chan.input2`	0	0		
`r.st`		`receiving`	`receiving`		
`r.ack`	`ack_chan.input`	1	1		
`r.expected`		0	0		
`msg_chan.output1`	`r.message1`	0	0		
`msg_chan.output2`	`r.message2`	1	0		
`msg_chan.forget`		0	0		
`ack_chan.output`	`s.ack`	1	1		
`ack_chan.forget`		0	0		

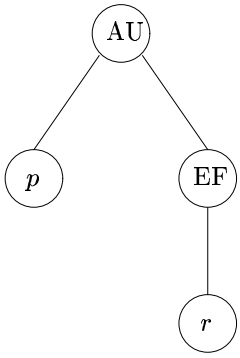
EXERCISES 3.4 (p.247)

1(a). The parsetree is:

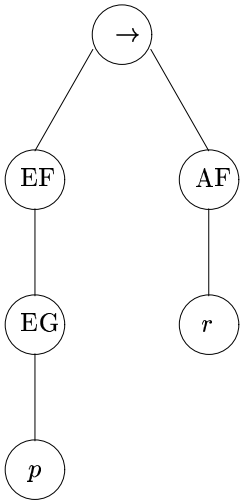
1(b). The parsetree is:



1(c). The parsetree is:



1(d). The parsetree is:

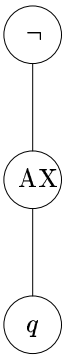


2(a). In CTL every F and every G has to be combined with a path quantifier A or E. For example, AF EG r and AF AG r etc. are such well-formed CTL formulas.

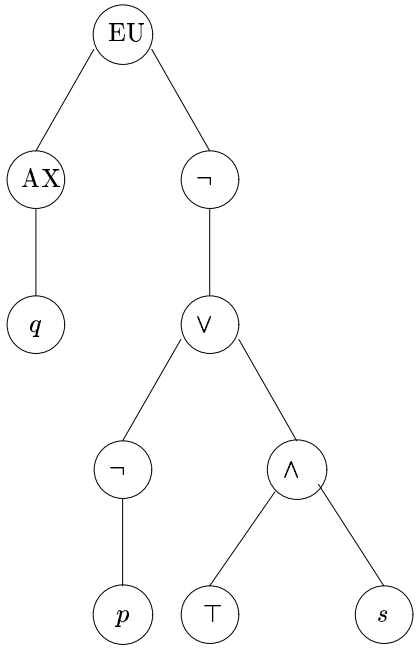
2(f). The formula AEF r is not in CTL, since we cannot put a path quantifier A in front of a CTL formula (here: EF r).

2(g). The formula A$[(p \text{ U } q) \wedge (p \text{ U } r)]$ is not in CTL, since there is no clause of the form A$[\phi_1 \wedge \phi_2]$ in the grammar of CTL. NB: There is also no clause of the form A$[(\phi_1 \text{ U } \phi_2) \wedge (\phi_3 \text{ U } \phi_4)]$.

3(c). This is a CTL formula; its parsetree is:

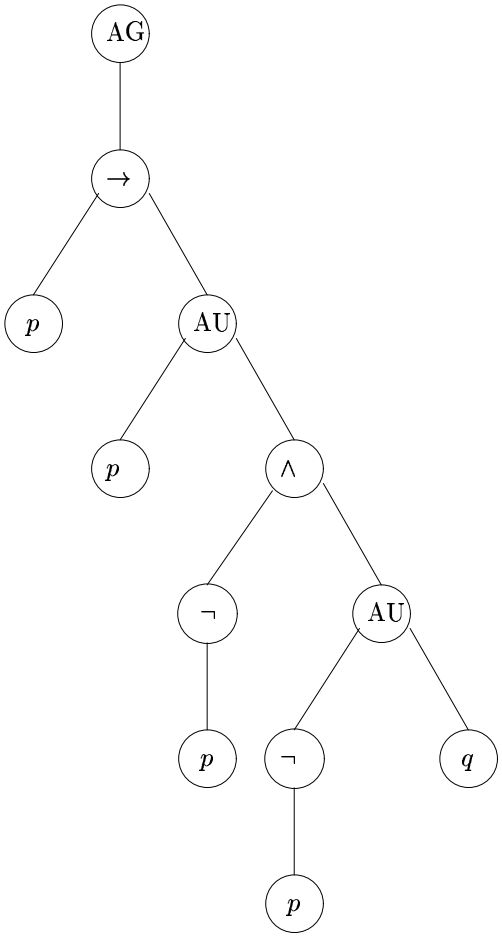


3(e). This is a CTL formula; its parsetree is:



3(f). For $\mathrm{F}\, r \wedge \mathrm{AG}\, q$ to be a CTL formula it would have to be the case that $\mathrm{F}\, r$ is a CTL formula as there is only one clause for constructing conjunctions, but this is not the case.

4. We list all subformulas of the formula $\mathrm{AG}\,(p \rightarrow \mathrm{A}[p\ \mathrm{U}\ (\neg p \wedge \mathrm{A}[\neg p\ \mathrm{U}\ q])])$ by first drawing its parsetree; then it is simple to read off all subformulas from that tree:



The list of subformulas is therefore

$$
\begin{aligned}
&p\\
&q\\
&\neg p\\
&\mathrm{A}[\neg p \ \mathrm{U} \ q]\\
&\neg p \wedge \mathrm{A}[\neg p \ \mathrm{U} \ q]\\
&\mathrm{A}[p \ \mathrm{U} \ \neg p \wedge \mathrm{A}[\neg p \ \mathrm{U} \ q]]\\
&p \rightarrow \mathrm{A}[p \ \mathrm{U} \ \neg p \wedge \mathrm{A}[\neg p \ \mathrm{U} \ q]]\\
&\mathrm{AG}\,(p \rightarrow \mathrm{A}[p \ \mathrm{U} \ (\neg p \wedge \mathrm{A}[\neg p \ \mathrm{U} \ q])]).
\end{aligned}
$$

6(b)(i). Since $r \in L(s_0)$, i.e. r is listed "inside" state s_0, we have $\mathcal{M}, s_0 \vDash r$. Thus, we infer that $\mathcal{M}, s_0 \vDash \neg p \rightarrow r$, for $\cdot \rightarrow \mathtt{T} = \mathtt{T}$.

6(b)(iii). Since $\mathcal{M}, s_0 \models r$ (see item 1b(i)) and since there is an infinite path $s_0 \to s_0 \to s_0 \to \cdots$, we have $\mathcal{M}, s_0 \models \mathrm{EG}\, r$. Therefore, we infer $\mathcal{M}, s_0 \not\models \neg\mathrm{EG}\, r$.

7(a). We have $[\![\top]\!] = S$ since by definition $s \models \top$ for all $s \in S$.

7(f). We have $s \models \phi_1 \to \phi_2$ if $s \models \phi_2$ or $s \not\models \phi_1$. This immediately renders $[\![\phi_1 \to \phi_2]\!] = (S - [\![\phi_1]\!]) \cup [\![\phi_2]\!]$.

7(g). We already saw that $\mathrm{AX}\,\phi$ and $\neg\mathrm{EX}\,\neg\phi$ are equivalent. Thus, this identity is clear.

9. We begin with the modified versions based on G and F. Independent of the path quantifier A or E, we need to modify F and G to stand for a strict future, respectively a strict globality. This can be done by wrapping the respective CTL connective with an EX or an AX respectively:

new $\mathrm{AG}\,\phi$**:** $\mathrm{AX}\,(\mathrm{AG}\,\phi)$ (Why is $\mathrm{AG}\,(\mathrm{AX}\,\phi)$ incorrect?)
new $\mathrm{EG}\,\phi$**:** $\mathrm{EX}\,(\mathrm{AG}\,\phi)$
new $\mathrm{AF}\,\phi$**:** $\mathrm{AX}\,(\mathrm{AF}\,\phi)$
new $\mathrm{EG}\,\phi$**:** $\mathrm{EX}\,(\mathrm{EG}\,\phi)$.

As for the U connective, we basically want to maintain the nature of the ϕ_1 U ϕ_2 pattern, but what changes is that we ban the extreme case of having ϕ_2 at the first state. Thus, we have to make sure that ϕ_1 is true in the current state and conjoin this with the shifted AU, respectively EU operators:

new AU**:** $\phi_1 \wedge \mathrm{AX}\,(\mathrm{A}[\phi_1\ \mathrm{U}\ \phi_2])$
new EU**:** $\phi_1 \wedge \mathrm{EX}\,(\mathrm{E}[\phi_1\ \mathrm{U}\ \phi_2])$.

10(b).

1. First, assume that $s \models \mathrm{EF}\,\phi \vee \mathrm{EF}\,\psi$. Then, without loss of generality, we may assume that $s \models \mathrm{EF}\,\phi$ (the other case is shown in the same manner). This means that there is a future state s_n, reachable from s, such that $s_n \models \phi$. But then $s_n \models \phi \vee \psi$ follows. But this means that there is a state reachable from s which satisfies $\phi \vee \psi$. Thus, $s \models \mathrm{EF}\,(\phi \vee \psi)$ follows.
2. Second, assume that $s \models \mathrm{EF}\,(\phi \vee \psi)$. Then there exists a state s_m, reachable from s, such that $s_m \models \phi \vee \psi$. Without loss of generality, we may assume that $s_m \models \psi$. But then we can conclude that $s \models \mathrm{EF}\,\psi$, as s_m is reachable from s. Therefore, we also have $s \models \mathrm{EF}\,\psi \vee \mathrm{EF}\,\psi$.

10(c). While we have that $s \models (\mathrm{AF}\,\phi \vee \mathrm{AF}\,\psi)$ implies $s \models \mathrm{AF}\,(\phi \vee \psi)$, the converse is not true. Therefore, the formulas $\mathrm{AF}\,\phi \vee \mathrm{AF}\,\psi$ and $\mathrm{AF}\,(\phi \vee \psi)$ are not equivalent. To see that the converse fails, consider

a model with three states i, s and t such that $i \to s$, $i \to t$, $s \to s$, $s \to t$, $t \to s$, and $t \to t$ are the state transitions. If we think of ϕ and ψ to be atoms p, respectively q, we create labelings $L(i) = \emptyset$, $L(s) = \{p\}$ and $L(t) = \{q\}$. Since s and t satisfy $p \vee q$ we have $i \vDash \mathrm{AF}\,(p \vee q)$. However, we do not have $i \vDash \mathrm{AF}\,p \vee \mathrm{AF}\,q$:

- To see $i \nvDash \mathrm{AF}\,p$ we can chose the path $i \to t \to t \to t \to t \to \cdots$.
- To see $i \nvDash \mathrm{AF}\,q$ we symmetrically choose the path $i \to s \to s \to s \to \cdots$.

10(e). This is not an equivalence (it would have to be $\neg\mathrm{AG}\,\phi$ instead of $\neg\mathrm{AF}\,\phi$). Consider the model from item 1c. We have $s \vDash \mathrm{EF}\,\neg p$ since we have the initial path segment $s \to t \to \ldots$. But we do not have $s \vDash \neg\mathrm{AF}\,p$, for the present is part of the future in CTL.

10(h). Saying that a CTL formula ϕ is equivalent to $\top$ is just paraphrazing that ϕ is true at all states in all models. (Why?) But this is not the case for $\mathrm{EG}\,\phi \to \mathrm{AG}\,\phi$. Consider the model of item 1c. Clearly, we have $s \vDash \mathrm{EG}\,p$, but we certainly don't have $s \vDash \mathrm{AG}\,p$. (What about $\top$ and $\mathrm{AG}\,\phi \to \mathrm{EG}\,\phi$, though?)

11(a). The formula $\mathrm{AG}\,(\phi \wedge \psi)$ is equivalent to $\mathrm{AG}\,\phi \wedge \mathrm{AG}\,\psi$ since both express that ϕ, as well as ψ, is true for all states reachable from the state under consideration.

14.

function TRANSLATE (ϕ)
/* translates a CTL formula ϕ into an equivalent one from an adequate fragment */
begin
case
ϕ is $\top$: **return** $\neg\bot$
ϕ is $\bot$: **return** $\bot$
$\phi \in$ `Atoms` : **return** ϕ
ϕ is $\neg\phi_1$: **return** $\neg$TRANSLATE ϕ
ϕ is $\phi_1 \wedge \phi_2$: **return** (TRANSLATE ϕ_1) $\wedge$ TRANSLATE ϕ_2
ϕ is $\phi_1 \vee \phi_2$: **return** TRANSLATE $(\neg\phi_1 \wedge \neg\phi_2)$
ϕ is $\phi_1 \to \phi_2$: **return** TRANSLATE $(\neg\phi \vee \phi_2)$
ϕ is $\mathrm{AX}\,\phi_1$: **return** TRANSLATE $\neg\mathrm{EX}\,\neg\phi_1$
ϕ is $\mathrm{EX}\,\phi_1$: **return** EX TRANSLATE ϕ_1
ϕ is $\mathrm{A}(\phi_1\ \mathrm{U}\ \phi_2)$: **return** TRANSLATE $\neg(\mathrm{E}[\neg\phi_2\ \mathrm{U}\ (\neg\phi_1 \wedge \neg\phi_2)] \vee \mathrm{EG}\,\neg\phi_2)$
ϕ is $\mathrm{E}(\phi_1\ \mathrm{U}\ \phi_2)$: **return** E[TRANSLATE ϕ_1 U TRANSLATE ϕ_2]
ϕ is $\mathrm{EF}\,\phi_1$: **return** TRANSLATE E[$\top$ U ϕ_1]
ϕ is $\mathrm{EG}\,\phi_1$: **return** $\neg\mathrm{AF}\,\neg$TRANSLATE ϕ_1
ϕ is $\mathrm{AF}\,\phi_1$: **return** AF TRANSLATE ϕ_1
ϕ is $\mathrm{AG}\,\phi_1$: **return** $\neg\mathrm{EF}\,\neg$TRANSLATE ϕ_1

end case
end function

You can now prove, by mathematical induction on the height of ϕ's parse tree, that the call TRANSLATE ϕ terminates and that the resulting formula has connectives only from the set $\{\bot, \neg, \wedge, \mathrm{AF}, \mathrm{EU}, \mathrm{EX}\}$.

EXERCISES 3.5 (p.250)

1(a). The process of translating informal requirements into formal specifications is subject to various pitfalls. One of them is simply ambiguity. For example, it is unclear whether "after some finite steps" means "at least one, but finitely many steps", or whether zero steps are allowed as well. It may also be debatable what "then" exactly means in "... then the system enters ...". We chose to solve this problem for the case when zero steps are not admissible, mostly since "followed by" suggests a real state transition to take place. The CTL formula we came up with is

$$\mathrm{AG}\,(p \rightarrow \mathrm{AX}\,\mathrm{AG}\,(\neg q \vee A[\neg r\ \mathrm{U}\ t]))$$

which in LTL may be expressed as

$$\mathrm{G}(p \rightarrow \mathrm{XG}(\neg q \vee \neg r\ \mathrm{U}\ t)).$$

It says: At any state, if p is true, then at any state which one can reach with at least one state transition from here, either q is false, or r is false until t becomes true (for all continuations of the computation path). This is evidently the property we intent to model. Variaous other "equivalent" solutions can be given.

1(f). The informal specification is ambiguous. Assuming that we mean a particular path we have to say that there exists some path on which p is true every second state (this is the global part). We assume that the informal description meant to set this off such that p is true at the first (=current), third etc. state. The CTL* formula thus reads as

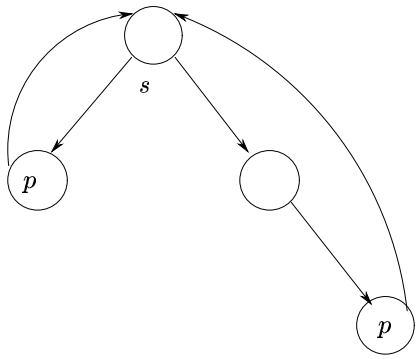
$$\mathrm{E}[\mathrm{G}(p \wedge \mathrm{XX}\,p)].$$

Note that this is indeed a CTL* formula and you can check that it insists on a path $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ where $s_0, s_2, s_4, \ldots$ all satisfy p.

6(b). Notice that the first clause in the grammar for path formulas says that "every state formula is a path formula as well". This is a *casting* in programming terminology. Semantically, it is justified since we can evaluate a state formula on a given path by simply evaluating it in the first state of that path. With this convention we can analyze the meanings of the two given CTL* formulas:

- A state s satisfies $\mathrm{AG\,F}p$ iff for all paths π with initial state s satisfies $\mathrm{GF}p$, i.e. p is true infinitely often on the path π. Thus, $\mathrm{AG\,F}p$ is equivalent to $\mathrm{AG\,(AF}\,p)$.
- The last observation makes clear why this cannot be equivalent to $\mathrm{AG\,(EF}\,p)$, although the former always implies the latter. Recalling our model of Exercise 3.4, item 1(c), note that $i \vDash \mathrm{AG\,(EF}\,p)$, since every state can reach state s, but that $i \nvDash \mathrm{AG\,(AF}\,p)$ (e.g. the path $i \to t \to t \to t \to \ldots$).

6(d). Clearly, $\mathrm{AX}\,p \vee \mathrm{AX\,AX}\,p$ implies $\mathrm{A}[\mathrm{X}\,p \vee \mathrm{XX}\,p]$ at any state and any model (why?). To see that the other implication fails in general, consider the model below:



State s satisfies $\mathrm{A}[\mathrm{X}\,p \vee \mathrm{XX}\,p]$ since every path either has to turn left ($\mathrm{X}p$) or right ($\mathrm{XX}\,p$). But state s neither satisfies $\mathrm{AX}\,p$ (turn right), nor $\mathrm{AX\,AX}\,p$ (turn left). Thus, it does not satisfy $\mathrm{AX}\,p \vee \mathrm{AX\,AX}\,p$.

7(a) $\mathrm{E}[\mathrm{F}\,p \wedge (q\ \mathrm{U}\ r)]$ is equivalent to $\mathrm{E}[q\ \mathrm{U}\ (p \wedge \mathrm{E}[q\ \mathrm{U}\ r])] \vee \mathrm{E}[q\ \mathrm{U}\ (r \wedge \mathrm{EF}\,p)]$.

7(b) $\mathrm{E}[\mathrm{F}\,p \wedge \mathrm{G}\,q]$ is equivalent to $\mathrm{E}[q\ \mathrm{U}\ (p \wedge \mathrm{EG}\,q)]$.

7(d) $\mathrm{A}[(p\ \mathrm{U}\ q) \wedge \mathrm{G}\,p]$ is equivalent to $\mathrm{A}[p\ \mathrm{U}\ q] \wedge \mathrm{AG}\,p$.

7(e) $\mathrm{A}[\mathrm{F}\,p \to \mathrm{F}\,q]$ is equivalent to $\neg\mathrm{E}[\mathrm{F}\,p \wedge \mathrm{G}\neg q]$, which we can then write as $\neg\mathrm{E}[\neg q\ \mathrm{U}\ (p \wedge \mathrm{EG}\,\neg q)]$ which is in CTL. If we allow R in CTL formulas, it can be written more succinctly as $\mathrm{A}[q\ \mathrm{R}\ (p \to \mathrm{AF}\,q)]$.

EXERCISES 3.6 (p.251) ϕ_1 to ϕ_4 refer to the Safety, Liveness, Non-blocking and No-strict-sequencing properties given on p.189.

1. • The specification `SPEC AG!((pr1.st = c) & (pr2.st = c))` holds since no state in that transition system is of the form $cc0$ or $cc1$. This does not require any fairness constraints.

• The specification `SPEC AG((pr1.st = t) -> AF (pr1.st = c))` is true, but depends on the use of fairness constraints. There are six different states in which `pr1.st = t` holds: $tt0$, $tt1$, $tc0$, $tn0$, $tn1$, and $tc1$. Note that there are transitions from some of these states to themselves, and a path which just ended up continuously in any of these states would violate the specification we mean to verify. This is exactly where fairness needs to come into the picture.

– The state $tt0$ eventually must transition to $ct0$, since `Fairness running` makes process 1 move at some point. Similarly, state $tt1$ must transition to $tc1$ eventually.

– The states $tn0$ and $tn1$ can remain where they are if process 2 makes a move by deciding to stay in its non-critical mode. But process 1 will eventually take a move (because of FAIRNESS running), and these states will then transition to $ct0$ and $ct1$ respectively.

– The system cannot remain in the states $tc0$ and $tc1$ forever because process 2 must eventually get selected to run, and the fairness constraint `!(st = c)` means that it must eventually leave its critical section.

Given the fact that the system can never stay in any of these six states forever, we see that eventually process 2 enters its critical section from any such state. (Why?)

• The argument for the specification

```
SPEC AG((pr2.st = t) -> AF (pr2.st = c))
```

is symmtric.

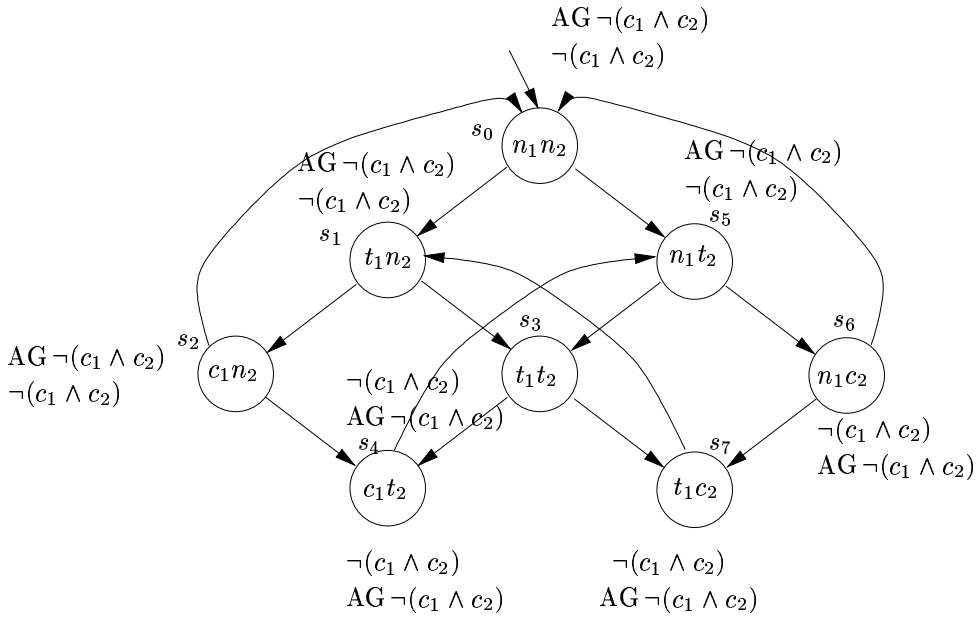
• The specification

```
SPEC EF(pr1.st=c &
        E[pr1.st=c U (!pr1.st=c & E[! pr2.st=c U pr1.st=c ])])
```

is true without any fairness assumptions, for it simply states the possibility of a certain execution pattern and we can read off such a path which enters c_1 twice in a row before it enters c_2. Indeed, if

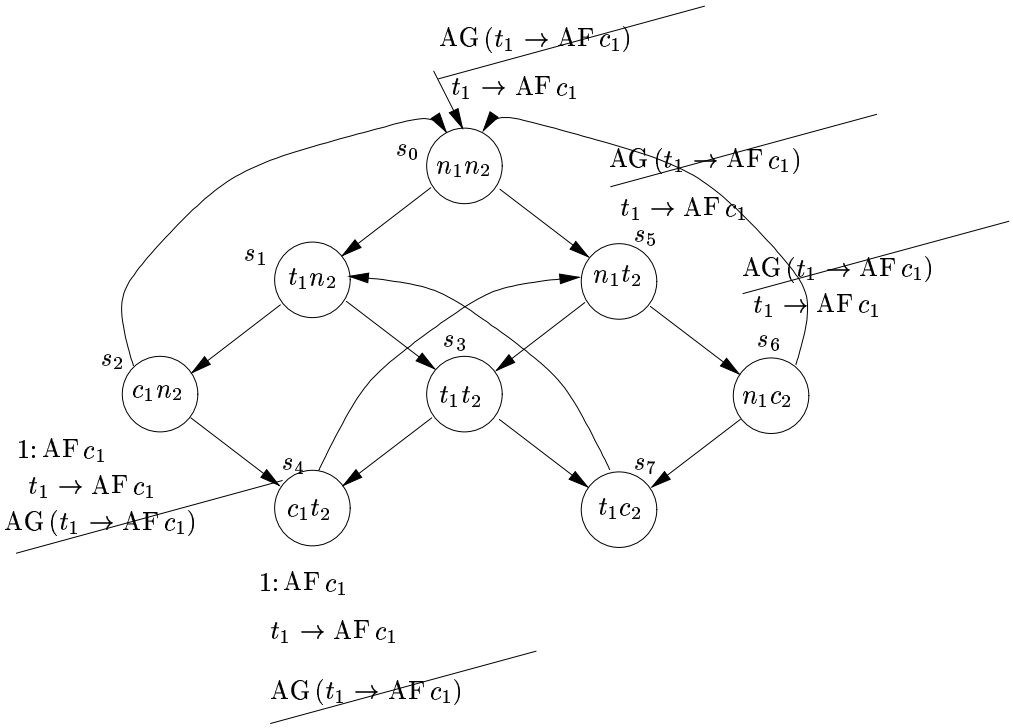
we remove all fairness constraints (even `FAIRNESS running`), this property still holds.

3. 1. $\mathrm{AG}\,\neg(c_1 \wedge c_2)$



No state gets a $c_1 \wedge c_2$ label. Therefore, all states get an $\neg(c_1 \wedge c_2)$ label. Thus, all states get, and keep, an $\mathrm{AG}\,\neg(c_1 \wedge c_2)$ label.

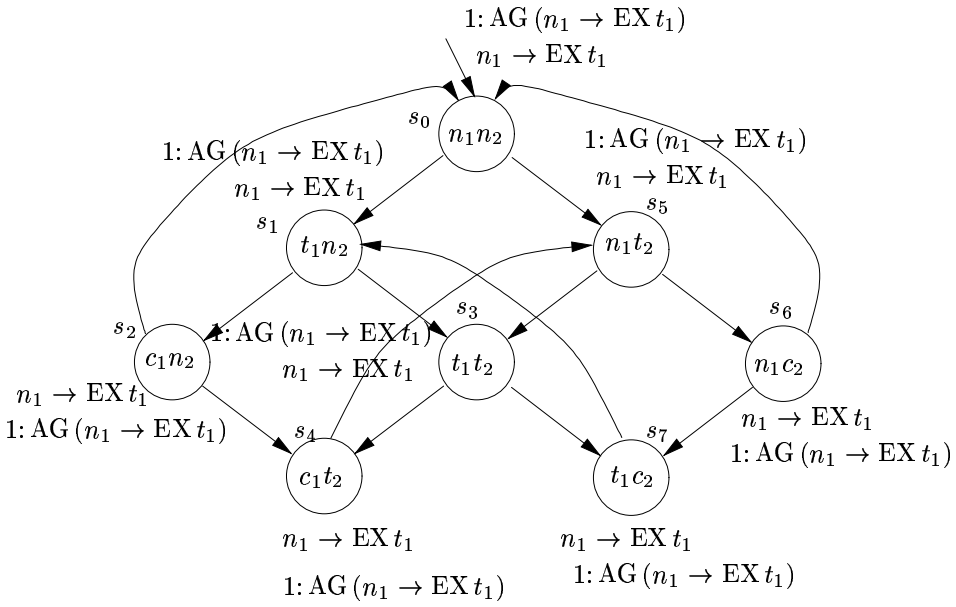
2. $\mathrm{AG}\,(t_1 \rightarrow \mathrm{AF}\,c_1)$



First, we determine all labels for AF c_1; only states s_2 and s_4 get such a label. Then we label a state with $t_1 \to$ AF c_1 if it does not have a t_1-label, or if it does have an AF c_1-label. Note that a state gets such a label if both cases apply. Indeed, this procedure does exactly what the truth table of $t_1 \to$ AF c_1 would compute if we think of `T` as "having a label" and `F` as "not having a label".

Second, we label all "$t_1 \to$ AF c_1 states" with AG $(t_1 \to$ AF $c_1)$, but in the next round those labels are deleted for states s_0, s_5, and s_6, causing the deletion of the remaining such labels in the next round. Thus, *no* state satisfies AG $(t_1 \to AFc_1)$.

3. AG $(n_1 \to$ EX $t_1)$

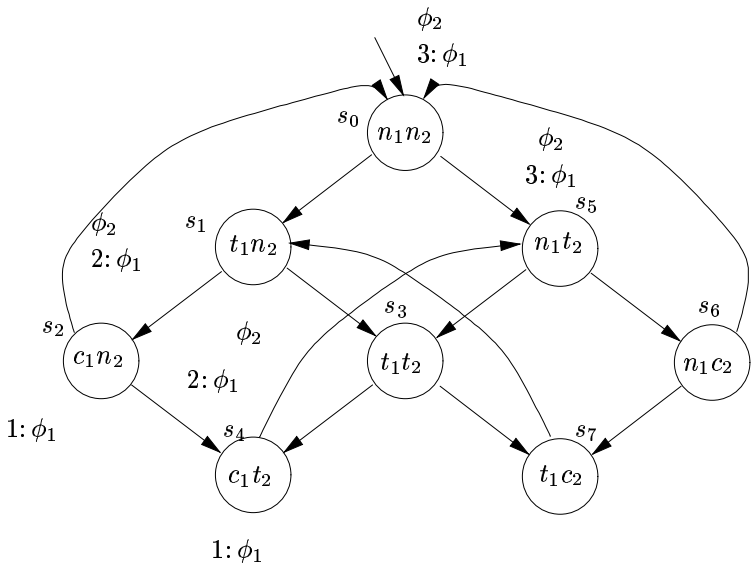


Similar to the case of $t_1 \to \mathrm{AF}\, c_1$ we label states with $n_1 \to \mathrm{EX}\, t_1$ if they don't have an n_1-label, or if they have an $\mathrm{EX}\, t_1$-label, which we do not show in the figure above. It turns out that all states get this label. Thus, we have to label all states with $\mathrm{AG}\,(n_1 \to \mathrm{EX}\, t_1)$ initially and so all of these labels survive the next round, i.e. all states get this label for good.

4. $\mathrm{EF}\,(c_1 \wedge \mathrm{E}[c_1\ \mathrm{U}\ (\neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1])])$.

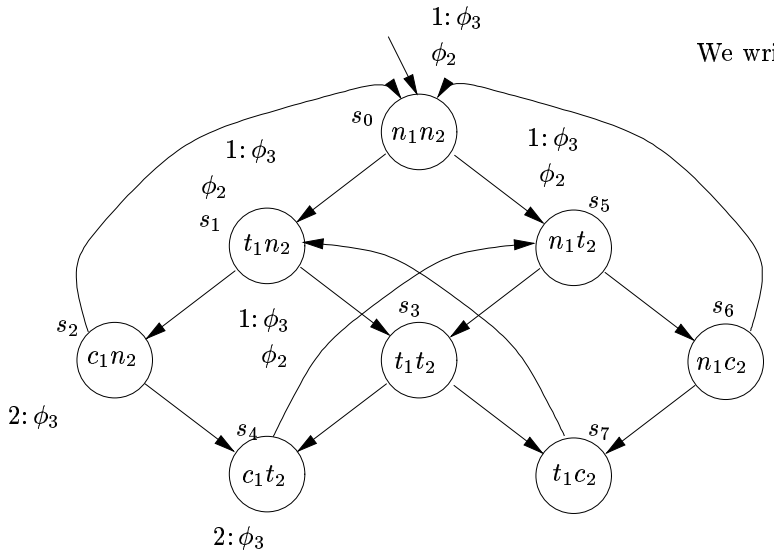
We write ϕ_1 for $\mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]$.

We write ϕ_2 for $\neg c_1 \wedge \phi_1$.



The interative procedure for determining the $\mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]$ labels takes three rounds. Then we "throw out" those labeled states which have a c_1-label. The resulting labels for $\neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]$ are now the target formula for the next *EU* operator:

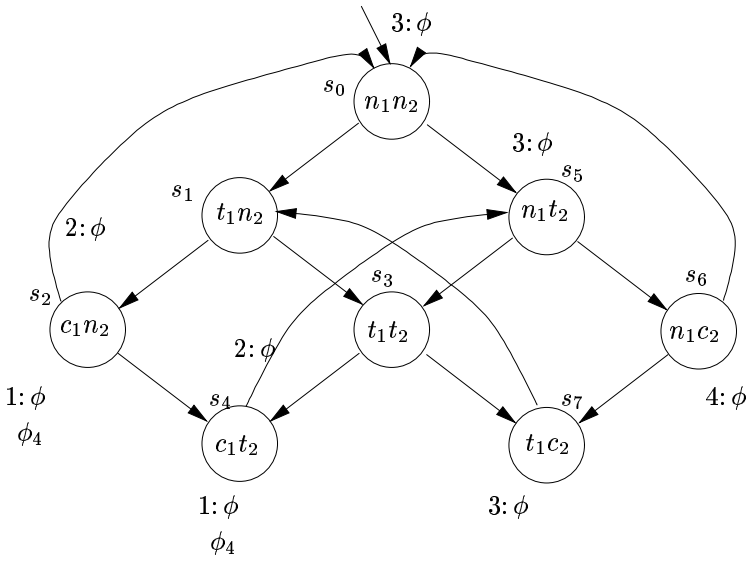
We write ϕ_3 for $\mathrm{E}[c_1\ \mathrm{U}\ \phi_2]$.



The iteration for this EU-labeling (ϕ_3) requires only two

rounds. Next, we refine ϕ_3 to those states which also satisfy c_1 (label for ϕ_4):

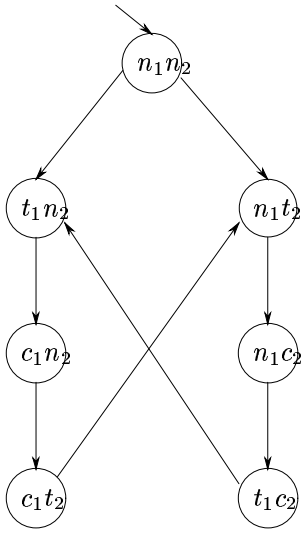
We write ϕ_4 for $c_1 \wedge \phi_3$.
We write ϕ for $\mathrm{EF}\,\phi_4$.



which triggers the iterative procedure for EF, which terminates after four rounds, concluding that all states satisfy $\mathrm{EF}\,(c_1 \wedge \mathrm{E}[c_1\ \mathrm{U}\ (\neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1])])$.

8.
```
function SAT_EG (φ)
/* determines the set of states satisfying EG φ */
begin
local var W, X, Y
begin
    Y := SAT (φ);
    X := S;
    repeat until X = Y
    begin
        X := Y;
        Y := Y ∩ {s | exists s' such that s → s' and s' ∈ Y}
    end
    return Y
end
```

9. There are lots of ways of scheduling this in a strictly alternating fashion. We chose a fairly deterministic version:



Note that the states on the left half, including n_1n_2, are the only ones that satisfy $\mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]$. Only t_1n_2 and n_1n_2 satisfy $\neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]$. These two are also the only states which satisfy $\mathrm{E}[c_1\ \mathrm{U}\ \neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]]$. Since none of them satisfies c_1, there is no state in the system which satisfies $c_1 \wedge \mathrm{E}[c_1\ \mathrm{U}\ \neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]]$. Therefore, no state of that system can satisfy $\mathrm{EF}\,(c_1 \wedge \mathrm{E}[c_1\ \mathrm{U}\ \neg c_1 \wedge \mathrm{E}[\neg c_2\ \mathrm{U}\ c_1]])$.

11. • Let ϕ be true infinitely often on the computation path $s_0 \to s_1 \to s_2 \to \ldots$. Proof by contradiction: Suppose that the negation of our claim is true. This means that there exists some $n \geq 0$ such that for all $m \geq n$ we have $s_m \not\models \phi$. But then ϕ could only be true on finitely many states of that path, namely at most at the states $s_0, s_1, \ldots, s_{n-1}$.

• Suppose that for every $n \geq 0$ there is some $m \geq n$ with $s_m \models \phi$. Proof by contradiction: Assume that ϕ is only true at finitely many states on that path. Then there has to be a maximal number n_0 such that no s_m with $m > n_0$ satisfies ϕ. By this is a contradiction to the assumption above which applied "for all $n \geq 0$", so in particular it applies to that n_0.

15. All the states are labelled $\mathrm{E}_C\mathrm{G}\top$ (i.e., all the states have a fair path leading from them).

EXERCISES 3.7 (p.252)

1(a). • The function H_1 is monotone: if Y is a subset of Y' then H_1 only removes the elements 1, 4, and 7 from Y and Y' if applicable. Since $Y \subseteq Y'$ this ensures that $H_1(Y) \subseteq H_1(Y')$.

- The function H_2 is not montone: e.g. $\{2\} \subseteq \{2,5\}$, but $H_2(\{2\})$, which is $\{5,9\}$, is not a subset of $H_2(\{2,5\})$, which is $\{9\}$.
- The function H_3 is monotone since union and intersection are monotone and the composition of monotone operations is again monotone.

1(b). We compute the least and greatest fixed-point of H_3 with just one iteration each: it is $\{2,4\}$ in each of these cases. So why can we say that this is the only fixed-point of H_3?

2(a) Let $X \subseteq X'$ and $s \in F_1(X)$. Then $s \in A \cap F(X)$ implies $s \in A$ and $s \in F(X)$. So in order to show $s \in F_1(X') = A \cap F(X')$ it suffices to show $s \in F(X')$. But this follows from $s \in F(X)$ as $X \subseteq X'$ and since F is monotone.

2(b) Let $X \subseteq X'$ and consider $s \in F_2(X)$.

- If $s \in A$, then $s \in A \cup (B \cap F(X') = F_2(X')$ follows.
- If $s \notin A$, then $s \in F_2(X) = A \cup (B \cap F(X))$ implies $s \in B \cap F(X)$, but we already saw in item (a) that this implies $s \in B \cap F(X')$ and so $s \in A \cup (B \cap F(X')) = F_2(X')$ does it.

5. • Let $X \subseteq X'$ and $s \in H(X) = [\![\phi]\!] \cap \{s_0 \in S \mid s_0 \to s_1 \text{ implies } s_1 \in X\}$. Then $s \in [\![\phi]\!]$ is clear. If $s' \in S$ with $s \to s'$, then $s \in H(X)$ implies $s' \in X$. Since $X \subseteq X'$, we infer from that $s' \in X'$. But this shows $s \in [\![\phi]\!] \cap \{s_0 \in S \mid s_0 \to s_1 \text{ implies } s_1 \in X'\} = H(X')$.

- We compute

$$\begin{aligned} H([\![\mathrm{AG}\,\phi]\!]) &= [\![\phi]\!] \cap \{s_0 \in S \mid s_0 \to s_1 \text{ implies } s_1 \in [\![\mathrm{AG}\,\phi]\!]\} \\ &= [\![\phi]\!] \cap [\![\mathrm{AX}\,\mathrm{AG}\,\phi]\!] \\ &= [\![\mathrm{AG}\,\phi]\!]. \end{aligned}$$

- We have already seen that $[\![\mathrm{AG}\,\phi]\!]$ is a fixed point of H. To show that it is the greatest fixed point, it suffices to show here that any set X with $H(X) = X$ has to be contained in $[\![\mathrm{AG}\,\phi]\!]$. So let s_0 be an element of such a fixed point X. We need to show that s_0 is in $[\![\mathrm{AG}\,\phi]\!]$ as well. For that we use the fact that

$$s_0 \in X = H(X) = [\![\phi]\!] \cap \{s \in S \mid s \to s_1 \text{ implies } s_1 \in X\}$$

to infer that $s_1 \in X$ for all s_1 with $s_0 \to s_1$. But, since s_1 is in X, we may apply that same argument to $s_1 \in X = H(X) = [\![\phi]\!] \cap \{s \in S \mid s \to s_2 \text{ implies } s_2 \in X\}$ and we get $s_2 \in X$ for all s_2 with $s_1 \to s_2$. By mathematical induction, we can therefore show that $s_n \in X$, where s_n is any state that is reachable from s_0. Since all states in X satisfy ϕ, this means that s is in $[\![\mathrm{AG}\,\phi]\!]$.

(d) The pseudo-code for $\mathrm{SAT}_{\mathrm{AG}}$:

```
function SAT_AG (φ)
/* determines the set of states satisfying AG φ */
local var X, Y
begin
    Y := SAT(φ);
    X := ∅;
    repeat until X = Y
    begin
        X := Y;
        Y := Y ∩ {s ∈ S | s → s' implies s' ∈ Y}
    end
    return Y
end
```

4

Program verification

EXERCISES 4.1 (p.299)

1. This is an open-ended and thought-provoking exercise. For example, Software Development Environments can improve the reliability of code by tracking version numbers and ensuring that all modified code is re-compiled, or linked when necessary. Editors can enhance the realibility of software by depicting source code through use of colors and fonts that check the integrity of syntax for the respective programming language. Programming language constructs may also improve or worsen the reliability of software. High-level abstractions and encapsulation principles are generally perceived as aiding the robustness of software whereas explicit pointer management may result in efficient but flawed software. Etc.

EXERCISES 4.2 (p.299)

1. In what circumstances would `if` B $\{C_1\}$ `else` $\{C_2\}$ fail to terminate? (We assume that the program is well-typed.) Our core language is such that boolean expressions B in themselves cannot diverge, i.e. they will return a truth value. Thus, this code can only diverge if C_1 or C_2 diverge and if the control flow passes execution on to the respective diverging command C_1 or C_2. Notice that this program necessarily diverges if both C_1 and C_2 diverge.

2. We can express such a for-loop in our core language as

```
C_1;
while B {
  C_3;
  C_2;
}
```

Note that this form first executes the body C_3 before it performs the update operation C_2 for the boolean condition B.

EXERCISES 4.3 (p.300)

1(a) Since $l(x) = -2$, $l(y) = 5$, and $l(z) = -1$ we have $l \not\models x + y < z$ as $-2 + 5$ is not less than -1. Thus $l \models (x + y < z) \to \neg(x * y = z)$ as the assumption of the implication does not hold.

1(c) We have $l \models (x + y - z) < x * y * z$ since the expression on the left evaluates to $-2 + 5 - (-1) = 4$ and the one on the right to $(-2) \cdot 5 \cdot (-1) = 10$.

2. For any ϕ, ψ, and P we have that $\models_{\mathsf{tot}} \{\!|\phi|\!\}\, P\, \{\!|\psi|\!\}$ implies $\models_{\mathsf{par}} \{\!|\phi|\!\}\, P\, \{\!|\psi|\!\}$. For let $\models_{\mathsf{tot}} \{\!|\phi|\!\}\, P\, \{\!|\psi|\!\}$ be the case. Assuming that program P is executed in a store meeting the precondition ϕ we then know that it will terminate its execution and end up in a final state meeting the postcondition ψ. If we ignore that this secures termination we have just shown $\models_{\mathsf{par}} \{\!|\phi|\!\}\, P\, \{\!|\psi|\!\}$.

Please remember that you can only understand proofs of $\{\!|\phi|\!\}\, P\, \{\!|\psi|\!\}$ by reading them in a bottom-up fashion!

5(a). We show $\vdash_{\mathsf{par}} \{\!|x > 0|\!\}$ `y = x + 1;` $\{\!|y > 1|\!\}$ by

$$
\begin{array}{ll}
\{\!|x > 0|\!\} & \\
\{\!|(x + 1) > 1|\!\} & \text{Implied} \\
\texttt{y = x + 1;} & \\
\{\!|y > 1|\!\} & \text{Assignment}
\end{array}
$$

5(b). We show $\vdash_{\mathsf{par}} \{\!|\top|\!\}$ `y = x; y = x + x + y;` $\{\!|y = 3 \cdot x|\!\}$ by

$$
\begin{array}{ll}
\{\!|\top|\!\} & \\
\{\!|x + x + x = 3 \cdot x|\!\} & \text{Implied} \\
\texttt{y = x;} & \\
\{\!|(x + x + y) = 3 \cdot x|\!\} & \text{Assignment} \\
\texttt{y = x + x + y;} & \\
\{\!|y = 3 \cdot x|\!\} & \text{Assignment}
\end{array}
$$

5(c). We show $\vdash_{\mathsf{par}} \{\!|x > 1|\!\}$ `a = 1; y = x; y = y-a;` $\{\!|y > 0 \wedge x > y|\!\}$ by

$(\!|x > 1|\!)$

$(\!|x - 1 > 0 \wedge x > x - 1|\!)$ Implied

```
a = 1;
```

$(\!|x - a > 0 \wedge x > x - a|\!)$ Assignment

```
y = x;
```

$(\!|(y - a) > 0 \wedge x > (y - a)|\!)$ Assignment

```
y = y - a;
```

$(\!|y > 0 \wedge x > y|\!)$ Assignment

6(a). There are infinitely many programs which meet these input/output behaviors. To implement $(\!|\top|\!)\, P\, (\!|y = x + 2|\!)$ we made a fairly obvious choice, namely the assignment `y = x +2;` The proof is

$(\!|\top|\!)$

$(\!|(x + 2) = x + 2|\!)$ Implied

```
y = x + 2;
```

$(\!|y = x + 2|\!)$ Assignment

6(b). There are infinitely many programs which meet these input/output behaviors. To implement $(\!|\top|\!)\, P\, (\!|z > x + y + 4|\!)$ we chose a non-optimal, but still correct, sequence of assignments:

$(\!|\top|\!)$

$(\!|10 > 4|\!)$ Implied

$(\!|x + (y + 10) > x + y + 4|\!)$ Implied

```
u = y + 10;
```

$(\!|(x + u) > x + y + 4|\!)$ Assignment

```
z = x + u;
```

$(\!|z > x + y + 4|\!)$ Assignment

9(c). We show that any instance of the rule If-Statement can be replaced by a proof in the proof system without this rule:

$$\dfrac{\dfrac{\dfrac{(\!|B \wedge \phi_1|\!)\, C_1\, (\!|\psi|\!)}{(\!|(B \to \phi_1) \wedge B|\!)\, C_1\, (\!|\psi|\!)}\ \text{Implied}}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2) \wedge B|\!)\, C_1\, (\!|\psi|\!)}\ \text{Implied} \qquad \dfrac{\dfrac{(\!|\neg B \wedge \phi_2|\!)\, C_2\, (\!|\psi|\!)}{(\!|(\neg B \to \phi_2) \wedge \neg B|\!)\, C_2\, (\!|\psi|\!)}\ \text{Implied}}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2) \wedge \neg B|\!)\, C_2\, (\!|\psi|\!)}\ \text{Implied}}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2)|\!)\ \texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-statement}$$

10. We show $\vdash_{\mathsf{par}} \{\top\}\, P\, \{z = \min(x, y)\}$, where $\min(x, y)$ is the smallest number of x and y (e.g. $\min(7, 3) = 3$) for the code P given below:

$\{\top\}$	
`if x > y {`	
$\{\top \wedge (x > y)\}$	If-statement
$\{y = \min(x, y)\}$	Implied
`z = y;`	
$\{z = \min(x, y)\}$	Assignment
`} else {`	
$\{\top \wedge \neg(x > y)\}$	If-statement
$\{x = \min(x, y)\}$	Implied
`z = x;`	
$\{z = \min(x, y)\}$	Assignment
`}`	
$\{z = \min(x, y)\}$	If-statement

If x equals y at runtime then the second branch of the if-statement gets executed and the value of x will be the result. This is fine as this value then also equals the value of y. Logically, we cannot really improve our description of $\min(x, y)$ (in the sense that we could impose more or less properties on it), but human beings might benefit from the fact the you draw their attention to special or overlapping cases of your specification (such as the one where x equals y). (How do you prove this with the modified proof rule for if-statements?)

11(a). We write code P which satisfies $\{\top\}\, P\, \{z = \max(x, y)\}$ under partial correctness, where $\max(x, y)$ denotes the larger of x and y:

$\{\!|\top|\!\}$
`if x < y {`
  $\{\!|\top \wedge (x < y)|\!\}$ If-statement
  $\{\!|y = \max(x, y)|\!\}$ Implied
  `z = y;`
  $\{\!|z = \max(x, y)|\!\}$ Assignment
`} else {`
  $\{\!|\top \wedge \neg(x < y)|\!\}$ If-statement
  $\{\!|x = \max(x, y)|\!\}$ Implied
  `z = x;`
  $\{\!|z = \max(x, y)|\!\}$ Assignment
`}`
$\{\!|z = \max(x, y)|\!\}$ If-statement

Notice that we simply took the code and the proof for min and changed $>$ into $<$ and min into max.

11(b). We seek a program P which satisfies

$$\vdash_{\mathsf{par}} \{\!|\top|\!\}\, P\, \{\!|((x = 5) \to (y = 3)) \wedge ((x = 3) \to (y = 1))|\!\}. \qquad (4.1)$$

Again, there are many programs one could write here as a solution. We chose a fairly natural one. We prove (4.1) as follows:

$\{\!|\top|\!\}$
`if x = 5 {`
  $\{\!|\top \wedge (x = 5)|\!\}$ If-statement
  $\{\!|((x = 5) \to (3 = 3)) \wedge ((x = 3) \to (3 = 1))|\!\}$ Implied
  `y = 3;`
  $\{\!|((x = 5) \to (y = 3)) \wedge ((x = 3) \to (y = 1))|\!\}$ Assignment
`} else {`
  $\{\!|\top \wedge \neg(x = 5)|\!\}$ If-statement
  $\{\!|((x = 5) \to (1 = 3)) \wedge ((x = 3) \to (1 = 1))|\!\}$ Implied
  `y = 1;`
  $\{\!|((x = 5) \to (y = 3)) \wedge ((x = 3) \to (y = 1))|\!\}$ Assignment
`}`
$\{\!|((x = 5) \to (y = 3)) \wedge ((x = 3) \to (y = 1))|\!\}$ If-statement

Note that the applications of the rule Implied in lines 4 and 9 was valid since $\mathtt{F} \to \cdot = \mathtt{T}$ and $\mathtt{T} \to \mathtt{T} = \mathtt{T}$.
Two other solutions are possible: one could program a "nested" if-statement which is not really needed since the precondition only means to distinguish between two values of x, so we may identify its computation for all values of x other than 5. The second solution is an optimization: the program

```
y = x - 2;
```

also satisfies the required pre- and postconditions. (Why?)

13. We show $\vdash_{\mathsf{par}} \{\!| x \geq 0 |\!\}\ \mathtt{Copy1}\ \{\!| x = y |\!\}$. First, we note which variables get updated in the body of the while-statement: y and a. Second, we compute their values after each iteration for, say, the first four iterations. Third, upon inspection of these values we suggest $y = x - a$ as a candidate for an invariant. The proof is

$\{\!\| x \geq 0 \|\!\}$	
$\{\!\| 0 = x - x \|\!\}$	Implied
`a = x;`	
$\{\!\| 0 = x - a \|\!\}$	Assignment
`y = 0;`	
$\{\!\| y = x - a \|\!\}$	Assignment
`while !(a = 0) {`	
$\quad\{\!\| y = x - a \|\!\}$	Invariant Hyp. $\wedge$ guard
$\quad\{\!\| (y + 1) = x - (a - 1) \|\!\}$	Implied
`y = y + 1;`	
$\quad\{\!\| y = x - (a - 1) \|\!\}$	Assignment
`a = a - 1;`	
$\quad\{\!\| y = x - a \|\!\}$	Assignment
`}`	
$\{\!\| (y = x - a) \wedge \neg\neg(a = 0) \|\!\}$	Partial-while
$\{\!\| y = x \|\!\}$	Implied

Note that we never really made use of the precondition $x \geq 0$; this is in contrast to the corresponding proof of *total* correctness which additionally secures program termination; then this precondition is instrumental in securing program correctness.

14. We show $\vdash_{\mathsf{par}} \{y \geq 0\}\ \texttt{Multi1}\ \{z = x \cdot y\}$. First, we note which variables get updated in the body of the while-statement: z and a. Second, we compute their values after each iteration for, say, the first four iterations. Third, upon inspection of these values we suggest $z = x \cdot a$ as a candidate for an invariant. The proof is

$\{y \geq 0\}$	
$\{0 = x \cdot 0\}$	Implied
`a = 0;`	
$\{0 = x \cdot a\}$	Assignment
`z = 0;`	
$\{z = x \cdot a\}$	Assignment
`while !(a = y) {`	
$\quad\{z = x \cdot a\}$	Invariant Hyp. $\wedge$ guard
$\quad\{z + x = x \cdot (a + 1)\}$	Implied
`z = z + x;`	
$\quad\{z = x \cdot (a + 1)\}$	Assignment
`a = a + 1;`	
$\quad\{z = x \cdot a\}$	Assignment
`}`	
$\{(z = x \cdot a) \wedge \neg\neg(a = y)\}$	Partial-while
$\{z = x \cdot y\}$	Implied

Again, we never used the precondition $y \geq 0$ for this *partial* correctness proof, but it is crucial in securing total correctness later on.

18. We show $\vdash_{\mathsf{par}} \{x \geq 0\}\ \texttt{Downfac}\ \{y = x!\}$. First, we note which variables get updated in the body of the while-statement: y and a. Second, we compute their values after each iteration for, say, the first four iterations. Third, upon inspection of these values we suggest $y \cdot (a!) = x!$ as a candidate for an invariant. However, since $\neg(a > 0)$ does not imply $a = 0$, we need to strengthen this invariant to $(y \cdot (a!) = x!) \wedge (a \geq 0)$. The proof is

$\{(x \geq 0)\}$	
$\{(1 \cdot (x!) = x!) \wedge (x \geq 0)\}$	Implied
`a = x;`	
$\{(1 \cdot (a!) = x!) \wedge (a \geq 0)\}$	Assignment
`y = 1;`	
$\{(y \cdot (a!) = x!) \wedge (a \geq 0)\}$	Assignment
`while a > 0 {`	
$\{(y \cdot (a!) = x!) \wedge (a \geq 0) \wedge (a > 0)\}$	Invariant Hyp. $\wedge$ guard
$\{(y \cdot ((a-1)! \cdot a) = x!) \wedge (a - 1 \geq 0)\}$	Implied
$\{((y \cdot a) \cdot ((a-1)!) = x!) \wedge (a - 1 \geq 0)\}$	Implied
`y = y * a;`	
$\{(y \cdot ((a-1)!) = x!) \wedge (a - 1 \geq 0)\}$	Assignment
`a = a - 1;`	
$\{(y \cdot (a!) = x!) \wedge (a \geq 0)\}$	Assignment
}	
$\{(y \cdot (a!) = x!) \wedge (a \geq 0) \wedge \neg(a > 0)\}$	Partial-while
$\{y = x!\}$	Implied

Note that we had to strenghten our invariant hypothesis by adding a conjunct $(a \geq 0)$. This was needed since $\neg(a > 0)$ does not in itself imply the desired $(a = 0)$ to secure that our invariant implies the postcondition.

21(a). We simulate the code for each of these arrays in a table which lists

the values of `k`, `t`, and `s` where "time" runs downwards.

k	t	s
2		
	−3	
		−3
	−2	
		−3
3		
	−4	
		−4
4		
	−1	
		−4
5		
	−9	
		−9
6		

Notice that the minimal-sum section is unique in this case: it is the sum of the entire array which is −9, the last value stored in `s`.

21(c). We proceed as in item 1:

k	t	s
2		
	−1	
		−1
	−3	
		−3
3		
	−6	
		−6
4		
	−10	
		−10
5		
	1087	
		−10
6		

Again, there is a unique minimal-sum section: the entire array but the last entry.

23. A direct attempt in proving the specification **S2** for `Min_Sum`would look like this:

$\{\top\}$
$\{a[1] = S(1,1)\}$ Implied
$\{\exists i,j\,(i \le j \le n \wedge a[1] = S(i,j))\}$ Implied
`k = 2;`
$\{\exists i,j\,(i \le j \le n \wedge a[1] = S(i,j))\}$ Assignment
`t = a[1];`
$\{\exists i,j\,(i \le j \le n \wedge a[1] = S(i,j))\}$ Assignment
`s = a[1];`
$\{\exists i,j\,(i \le j \le n \wedge s = S(i,j))\}$ Assignment
`while (k != n + 1) {`
$\{\exists i,j\,(i \le j \le n \wedge s = S(i,j))\}$ Invariant Hyp. $\wedge$ guard
$\{\exists i,j\,(i \le j \le n \wedge \min(s, \min(t + a[k], a[k])) = S(i,j))\}$ Implied
`t = min(t + a[k], a[k]);`
$\{\exists i,j\,(i \le j \le n \wedge \min(s,t) = S(i,j))\}$ Assignment
`s = min(s,t);`
$\{\exists i,j\,(i \le j \le n \wedge s = S(i,j))\}$ Assignment
`k = k + 1;`
$\{\exists i,j\,(i \le j \le n \wedge s = S(i,j))\}$ Assignment
`}`
$\{\exists i,j\,(i \le j \le n \wedge s = S(i,j))\}$ Partial-while

It looks fine if it were not for the justification of the Implied rule in the while-loop. Knowing that s is the sum of some section does not immediately imply that $\min(s, \min(t+a[k], a[k]))$ is the sum of some section as well. Evidently, we need to reason that $\min(t + a[k], a[k])$ is the sum of some section for all values of `k` that the program is computing. This proof is similar to the one given above and we omit it here.

EXERCISES 4.4 (p.303)

1(a). We prove $\vdash_{\mathsf{tot}} \{x \ge 0\}\ \texttt{Copy1}\ \{x = y\}$ as follows: since the while-loop in `Copy1` has as guard B the boolean expression $\neg(a = 0)$ and,

since $a \geq 0$ is an invariant of the entire program (assuming the precondition $x \geq 0$), we may instantiate the formula $0 \leq E = E_0$ to $0 \leq a = E_0$. Of course, we may reuse the invariant ϕ_I, which is $a + y = x$ from the partial correctness proof. With these data at hand we simply have to follow the proof pattern for total correctness proofs:

$\{\!\mid x \geq 0 \mid\!\}$	
$\{\!\mid (x + 0 = x) \wedge (x \geq 0) \mid\!\}$	Implied
`a = x;`	
$\{\!\mid (a + 0 = x) \wedge (a \geq 0) \mid\!\}$	Assignment
`y = 0;`	
$\{\!\mid (a + y = x) \wedge (a \geq 0) \mid\!\}$	Assignment
`while !(a = 0) {`	
$\{\!\mid (a + y = x) \wedge \neg(a = 0) \wedge (0 \leq a = E_0) \mid\!\}$	Invariant Hyp. $\wedge$ guard
$\{\!\mid (a - 1 + y + 1 = x) \wedge (0 \leq a - 1 < E_0) \mid\!\}$	Implied
`y = y + 1;`	
$\{\!\mid (a - 1 + y = x) \wedge (0 \leq a - 1 < E_0) \mid\!\}$	Assignment
`a = a - 1;`	
$\{\!\mid (a + y = x) \wedge (0 \leq a < E_0) \mid\!\}$	Assignment
`}`	
$\{\!\mid (a + y = x) \wedge \neg\neg(a = 0) \mid\!\}$	Total-while
$\{\!\mid x = y \mid\!\}$	Implied

Note that the Implied in line 9 is valid, since $\neg(a = 0)$ and $0 \leq a = E_0$ together imply $0 \leq a - 1 < E_0$.

1(b). We show $\vdash_{\text{tot}} \{\!\mid y \geq 0 \mid\!\}$ `Multi1` $\{\!\mid z = x * y \mid\!\}$, where the code for `Multi1` is

```
a = 0;
z = 0;
while (a != y) {
  z = z + x;
  a = a + 1;
}
```

The boolean guard `a != y` is equivalent to `y - a != 0`, so we may instantiate $0 \leq E = E_0$ to $0 \leq y - a = E_0$. This works out as

$0 \leq y - a$ turns out to be an invariant of the entire program. Since we already showed that $\{\!|y \geq 0|\!\}$ `Multi1` $\{\!|z = x \cdot y|\!\}$ holds under *partial* correctness, we may reuse the invariant ϕ_I, which is $z = x \cdot a$, from that proof. Now everything is in place for crafting the proof in a bottom-up fashion:

$\{\!\|y \geq 0\|\!\}$	
$\{\!\|(0 = x \cdot 0) \wedge (0 \leq y - 0)\|\!\}$	Implied
`a = 0;`	
$\{\!\|(0 = x \cdot a) \wedge (0 \leq y - a)\|\!\}$	Assignment
`z = 0;`	
$\{\!\|(z = x * a) \wedge (0 \leq y - a)\|\!\}$	Assignment
`while (y - a != 0) {`	
$\{\!\|(z = x \cdot a) \wedge \neg(y - a = 0) \wedge (0 \leq y - a = E_0)\|\!\}$	Invariant Hyp. $\wedge$ guard
$\{\!\|(z + x = x \cdot (a + 1)) \wedge (0 \leq y - (a + 1) < E_0)\|\!\}$	Implied
`z = z + x;`	
$\{\!\|(z = x \cdot (a + 1)) \wedge (0 \leq y - (a + 1) < E_0)\|\!\}$	Assignment
`a = a + 1;`	
$\{\!\|(z = x \cdot a) \wedge (0 \leq y - a < E_0)\|\!\}$	Assignment
}	
$\{\!\|(z = x \cdot a) \wedge \neg\neg(y - a = 0)\|\!\}$	Total-while
$\{\!\|z = x \cdot y\|\!\}$	Implied

Again, the use of $y \geq 0$ was crucial in justifying line 2; this program diverges whenever y is negative. Make sure that you understand why the application of Implied in line 9 is justified.

1(d). In order to prove $\vdash_{\text{tot}} \{\!|x \geq 0|\!\}$ `Downfac` $\{\!|y = x!|\!\}$ we may think of the Boolean guard `a > 0` as `a != 0`, for $a \geq 0$ turns out to be an invariant of the entire program. This suggests that $0 \leq E = E_0$ is just $0 \leq a = E_0$. We need as full invariant $(y \cdot (a!) = x!) \wedge (a \geq 0)$ from the case of partial correctness; the formula $y \cdot (a!) = x!$ alone will not do, since we mean to identify $\neg(a = 0)$ with $a > 0$. Thus, the entire proof may be constructed as follows:

$\{\!\mid x \geq 0 \mid\!\}$	
$\{\!\mid (1 * (x!) = x!) \wedge (x \geq 0) \wedge (0 \leq x) \mid\!\}$	Implied
`a = x;`	
$\{\!\mid (1 * (a!) = x!) \wedge (a \geq 0) \wedge (0 \leq a) \mid\!\}$	Assignment
`y = 1;`	
$\{\!\mid (y * (a!) = x!) \wedge (a \geq 0) \wedge (0 \leq a) \mid\!\}$	Assignment
`while a != 0 {`	
$\{\!\mid (y \cdot (a!) = x!) \wedge (a \geq 0) \wedge (\neg(a = 0) \wedge (0 \leq a = E_0) \mid\!\}$	Invariant Hyp. $\wedge$ guard
$\{\!\mid (y \cdot ((a-1)! \cdot a) = x!) \wedge (a \geq 0) \wedge (0 \leq a - 1 < E_0) \mid\!\}$	Implied
$\{\!\mid ((y \cdot a) \cdot ((a-1)!) = x!) \wedge (a \geq 0) \wedge (0 \leq a - 1 < E_0) \mid\!\}$	Implied
`y = y * a;`	
$\{\!\mid (y \cdot ((a-1)!) = x!) \wedge (a \geq 0) \wedge (0 \leq a - 1 < E_0) \mid\!\}$	Assignment
`a = a - 1;`	
$\{\!\mid (y \cdot (a!) = x!) \wedge (a \geq 0) \wedge (0 \leq a < E_0) \mid\!\}$	Assignment
`}`	
$\{\!\mid (y \cdot (a!) = x!) \wedge (a \geq 0) \wedge \neg\neg(a = 0) \mid\!\}$	Total-while
$\{\!\mid y = x! \mid\!\}$	Implied

Note that the $a \geq 0$ in line 6 stems from the invariant, whereas the second $a \geq 0$ results from the proof pattern $0 \leq E$.

1(e). We prove $\vdash_{\mathsf{tot}} \{\!\mid x \geq 0 \mid\!\}$ `Copy2` $\{\!\mid x = y \mid\!\}$, where `Copy2` is the code

```
y = 0;
while (y != x) {
  y = y + 1;
}
```

The boolean expression B is `y != x` which we can rewrite to the equivalent expression `x - y`. Therefore, the precondition $x \geq 0$ suggests to instantiate $0 \leq E = E_0$ to $0 \leq x - y = E_0$. An invariant ϕ_I requires that $\phi_I \wedge (x - y = 0)$ implies $x = y$. Since the latter is already a consequence of $x - y = 0$, we may simply choose $\top$ for ϕ_I.

$\{x \geq 0\}$	
$\{\top \wedge (x - 0 \geq 0)\}$	Implied
`y = 0;`	
$\{\top \wedge (x - y \geq 0)\}$	Assignment
`while x - y != 0 {`	
$\{\top \wedge \neg(x - y = 0) \wedge (0 \leq x - y = E_0)\}$	Invariant Hyp. $\wedge$ guard
$\{\top \wedge (0 \leq x - (y + 1) < E_0)\}$	Implied
`y = y + 1;`	
$\{\top \wedge (0 \leq x - y < E_0)\}$	Assignment
`}`	
$\{\top \wedge \neg\neg(x - y = 0)\}$	Total-while
$\{x = y\}$	Implied

Note that we actively made use of the precondition $x \geq 0$ and that this code diverges if $x < 0$.

EXERCISES 4.5 (p.304)

1(a). The method `certify_V` may itself call other certifications methods, e.g. two such methods `certify_V1` and `certify_V2`, and may accept a cerificate if both calls to `V1` and `V2` judge the certificate to be valid. Programming by contract is useful as it can specify necessary or sufficient conditions for accepting certificates.

1(b). One may face self-certifying services. For example, if `V1` calls `certify_V` and that method relies on some method call to `V1`, then `V1` could instrument that method code so as to influence the judgment of `V` withing `certify_V`.

2(a). We write a contract for method `isGood`:

`method name :`	`isGood`
`input :`	`amount of Type int`
`assumes :`	`access to 'balance'`
`guarantees :`	`isGood(amount) iff balance >= amount`
`output :`	`of Type boolean`
`modifies only :`	`nothing modified`

2(b). The method `withdraw` only modifies `balance` and accepts `amount` of type `int`. Therefore it suffices to show that it meets its guarantee, assuming that `balance <= 0` is true. If the boolean guard in its body

evaluates to false, then `balance` is not being modified and so satisfies `balance <= 0` by assumption. If the boolean guard evaluates to true, then the value of `amount` is negative and `isGood(amount)` returns true. By the guarantee of `isGood` the latter means that the value of `balance` is greater or equal to the value of `amount`. But then the execution of the assignment for `balance` ensures that its new value is greater or equal to zero.

5

Modal logics and agents

EXERCISES 5.2 (p.350)

1(a)iii. The relation $a \Vdash q$ holds as state a is labelled with q in the figure.

1(a)iv. The relation $a \Vdash \Box\Box q$ holds iff $x \Vdash \Box q$ holds for all x with $R(a, x)$. Since e and b are the only instances of x which satisfy $R(a, x)$, we see that $a \Vdash \Box\Box q$ holds iff $e \Vdash \Box q$ and $b \Vdash \Box q$ hold. But none of this is the case. For example, we have $R(b, e)$ and $e \nVdash q$, so $b \nVdash \Box q$ follows.

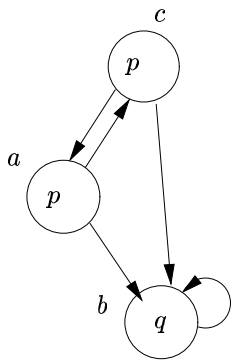
1(a)vi. The relation $a \Vdash \Box\Diamond\neg q$ holds iff $e \Vdash \Diamond\neg q$ and $b \Vdash \Diamond\neg q$ holds, since e and b are the only x with $R(a, x)$. First, we have $e \Vdash \Diamond\neg q$, since there is some x with $R(e, x)$ and $x \Vdash \neg q$; choose x to be e. Second, we also have $b \Vdash \Diamond\neg q$, because we may choose again e as x with $R(b, e)$ and $e \Vdash \neg q$. In conclusion, $a \Vdash \Box\Diamond\neg q$ holds.

1(a)x. We have $c \Vdash \Box\bot$ iff $x \Vdash \bot$ for all x with $R(c, x)$. By definition of $\Vdash$, we have $x \nVdash \bot$ for all x, so $c \Vdash \Box\bot$ holds iff there is no x with $R(c, x)$. Inspecting the figure, this is evidently the case. Thus, $c \Vdash \Box\bot$ holds.

1(b)iii. We seek a world which satisfies $\Diamond p \vee \Diamond q$, i.e. a world x such that there are worlds y and y' with $R(x, y)$ and $R(x, y')$ such that $y \Vdash p$ and $y' \Vdash q$. If we choose c for x and b for y and y', this is clearly realized. Therefore, $c \Vdash \Diamond p \vee \Diamond q$ follows.

1(b)iv. We have $x \Vdash \Diamond(p \vee \Diamond q)$ iff there is some y with $R(x, y)$ and $y \Vdash p \vee \Diamond q$. This can be realized. For example, choose x and y to be the worlds b and c, respectively.

4(b). • We seek a model in which $p \to \Box\Diamond q$ is true. By definition, we seek a model in which *all its worlds* satisfy $p \to \Box\Diamond q$. Certainly, any model in which no world satisfies p would do. (Why?) More interestingly, consider the figure below.



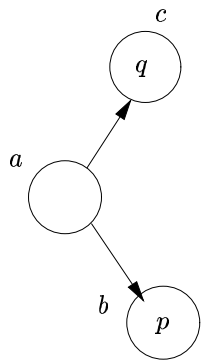
The world b satisfies $p \to \Box\Diamond q$, since $c \not\Vdash p$. For the worlds a and c, we have $a \Vdash p$ and $c \Vdash p$, so we need to secure $a \Vdash \Box\Diamond q$ and $c \Vdash \Box\Diamond q$. But this can be done, since all three worlds, x, of this model have an immediate successor state, x', such that $x' \Vdash q$. (Why does this imply what we require, and what choices of x' for x would you make?)

- We seek a model in which $p \to \Box\Diamond q$ is not true. By definition, we seek a model in which *not every world* satisfies $p \to \Box\Diamond q$. The model in Figure 5.5 of the textbook qualifies, for $a \not\Vdash p \to \Box\Diamond q$. (Why?)

5(d). The relation $x \Vdash \Diamond(p \wedge q)$ means that there is a world y with $R(x, y)$ and $y \Vdash p \wedge q$. On the other hand, $x \Vdash \Diamond p \wedge \Diamond q$ means that there are worlds y' and y'' with

$$R(x, y') \text{ and } y' \Vdash p$$
$$R(x, y'') \text{ and } y'' \Vdash q.$$

With that in mind, it is relatively easy to find a model which distinguishes these two formulas:



Note that $a \not\Vdash \Diamond(p \wedge q)$, whereas $a \Vdash \Diamond p \wedge \Diamond q$.

5(f).(i) First, we have $x \Vdash \Diamond(p \vee q)$ iff there is a world y with $R(x,y)$ and $y \Vdash p \vee q$. But then $y \Vdash p$ or $y \Vdash q$.

Case 1: If $y \Vdash p$, then $x \Vdash \Diamond p$, and so $x \Vdash \Diamond p \vee \Diamond q$ follows.

Case 2: If $y \Vdash q$, we argue in the symmetric way: $x \Vdash \Diamond q$, and so $x \Vdash \Diamond p \vee \Diamond q$ follows.

(ii) Second, if we have $x \Vdash \Diamond p \vee \Diamond q$, then we have $x \Vdash \Diamond p$, or $p \Vdash \Diamond q$, not necessarily exclusive.

Case 1: If $x \Vdash \Diamond p$, then there exists a world y' with $R(x,y')$ and $y \Vdash p$. This implies $y' \Vdash p \vee q$, and so $x \Vdash \Diamond(p \vee q)$ follows as $R(x,y')$.

Case 2: Symmetrically, if $x \Vdash \Diamond q$, then there exists a world y'' with $R(x,y'')$ and $y \Vdash q$. This implies $y'' \Vdash p \vee q$, and so $x \Vdash \Diamond(p \vee q)$ follows as $R(x,y'')$.

6(a). To show that $\Box(\phi \wedge \psi) \leftrightarrow (\Box\phi \wedge \Box\psi)$ is valid is suffices to show that $x \Vdash \Box(\phi \wedge \psi)$ implies $x \Vdash \Box\phi \wedge \Box\psi$, and vice versa, where x is any world of any model. (Why?)

(i) If $x \Vdash \Box(\phi \wedge \psi$, then $R(x,y)$ implies $y \Vdash \phi \wedge \psi$, and so $y \Vdash \phi$ and $y \Vdash \psi$ follow. Since y is an arbitrary element with $R(x,y)$, we conclude $x \Vdash \Box\phi$ and $x \Vdash \Box\psi$, and so $x \Vdash \Box\phi \wedge \Box\psi$ follows.

(ii) Conversely, if $x \Vdash \Box\phi \wedge \Box\psi$, we infer $x \Vdash \Box\phi$ and $x \Vdash \Box\psi$. If y is any element with $R(x,y)$, then $x \Vdash \Box\phi$ implies $y \Vdash \phi$, and $x \Vdash \Box\psi$ implies $y \Vdash \psi$. Therefore, $y \Vdash \phi \wedge \psi$ holds. Since y is an arbitrary element with $R(x,y)$, we infer that $x \Vdash \Box(\phi \wedge \psi)$ holds.

(Why is the argument above still valid if there is no y with $R(x,y)$?)

6(c). As in item 8(a), it suffices to show that $x \Vdash \Box\top$ implies $x \Vdash \top$, and vice versa, where x is any world of any model.

(i) If $x \Vdash \Box\top$, then $x \Vdash \top$ holds, since the latter holds simply by the definition of $\Vdash$.

(ii) If $x \Vdash \top$, then we have $y \Vdash \top$ for all y with $R(x,y)$, since we have $y \Vdash \top$ for all worlds y, by definition of $\Vdash$. Thus, $x \Vdash \Box\top$ is proved.

EXERCISES 5.3 (p.351)

1(a). We interpret the formula $(\phi \to \Box\phi) \to (\phi \to \Diamond\phi)$ according to the interpretations of $\Box$ and $\Diamond$ in Tables 5.7 and 5.6.

(i) It is necessarily true that ϕ: The formula says

"If ϕ's truth implies that ϕ is necessarily true, then ϕ's truth implies that ϕ is possibly true."

This should be valid.

(ii) It will always be true that ϕ: The formula says

"If ϕ's truth implies that ϕ is always true, then ϕ's truth implies that ϕ is true sometimes in the future."

This should be valid.

(iii) It ought to be that ϕ: The formula says

"If ϕ's truth implies that ϕ ought to be true, then ϕ's truth implies that it is permitted that ϕ (is true)."

This may be contested by some, but should generally be valid.

(iv) Agent Q believes that ϕ: The formula says

"If ϕ's truth implies that agent Q believes ϕ, then ϕ's truth implies that ϕ is consistent with Q's beliefs."

This should be valid, provided that agent Q values consistency of beliefs.

(v) Agent Q knows that ϕ: The formula says

"If ϕ's truth implies that agent Q knows that ϕ, then ϕ's truth implies that ϕ is true, for all agent Q knows."

This should be valid.

(vi) After any execution of program P, ϕ holds: The formula says

"If ϕ's truth implies that, after any execution of program P, ϕ holds, then ϕ's truth implies that ϕ holds after some execution of program P.

This should clearly be valid.

2(a) We have $l \models \neg\langle P\rangle\neg\phi$ iff we have $l \not\models \langle P\rangle\neg\phi$ iff (it is not the case that some execution beginning in store l terminates in a state satisfying $\neg\phi$) iff (all executions beginning in state l terminates in a state satisfying ϕ or don't terminate at all).

3(a). For natural numbers $n, m \geq 1$, we have $R(n, m)$ iff $n < m$. For example, $2 < 5$, but $5 \not< 2$.

- R is not reflexive, since $n \not< n$ for any $n \geq 1$.
- R is not symmetric, since $n < m$ implies $m \not< n$.
- R is serial, since for each $n \geq 1$ there is some m, for example $m \stackrel{\text{def}}{=} n + 1$, such that $n < m$.
- R is transitive, since $n < m$ and $m < k$ clearly imply $n < k$.
- R is not Euclidean; otherwise $n < m$ and $n < k$ would always imply $m < k$, but this is not the case. For example, we have $3 < 11$ and $3 < 5$, but $11 \not< 5$.
- R is not functional, since, e.g., we have $2 < 4$ and $2 < 5$ with 4 and 5 being distinct elements.

- R is linear, since $n > m$ and $n < k$ always imply that m and k are equal, or $k < m$, or $m < k$.
- R is not total; otherwise, we would have $n < m$ or $m < n$ for all $n, m \geq 1$, but this is not the case when n equals m.
- R is not an equivalence relation, since we saw above that it is neither reflexive, nor symmetric.

3(d) This is very similar to the previous solution in its methodology but some of the cases are a bit trickier.

- To see whether R is reflexive we need to check that for all real numbers x there are positive numbers a and b such that $x = a \cdot x + b$. This cannot be for $x = 0$ as b has to be positive.
- This relation is transitive. For if a, b, c, d are positive real numbers with $x = a \cdot y + b$ and $y = c \cdot z + d$, then $a \cdot c$ and $a \cdot d + b$ are positive as well and $x = a \cdot c \cdot z + (a \cdot d + b)$.
- This is clearly not functions, e.g. 2 equals $1 \cdot 1 + t$ and also equals $2 \cdot 0.5 + 1$.
- Etc.

4. Let R be a reflexive, transitive and Euclidean relation. We need to show that R is an equivalence relation. Since R is already reflexive and transitive, it suffices to show that R is symmetric. To that end, assume that $R(a, b)$. We have to show that $R(b, a)$ holds as well. Since R is Euclidean, we have that $R(x, y)$ and $R(x, z)$ imply $R(y, z)$ for all choices of x, y and z. So if we instantiate this with $x \stackrel{\text{def}}{=} a$, $y \stackrel{\text{def}}{=} b$ and $z \stackrel{\text{def}}{=} a$, then we have $R(x, y)$ by assumption (as $R(a, b)$), but we also have $R(x, z)$ since R is reflexive (so $R(a, a)$ holds). Using that R is Euclidean, we obtain $R(y, z)$ which is $R(b, a)$ as desired. Notice that this local argument made no use of the transitivity of R. (Why and where does the global argument use this transitivity?)

9(a) This is not valid in intuitionistic propositional logic. Consider a model with three worlds x, y, and z such that $L(x) = \{\}$, $L(y) = \{p\}$, and $L(z) = \{q\}$. Further, let $R(x, y)$ and $R(x, z)$ be the only non-reflexive instances of R. Note that L is monotone with respect to R. Then $x \not\Vdash p \to q$ since y is accessible by x and $p \to q$ does not hold at y. Similarly, $x \not\Vdash q \to r$ since z is accessible from x and does not satisfy $q \to r$.

9(f) This is intuitionistically valid. There is an obvious proof that involves only the proof rules $\neg$i and $\neg$e and these proof rules are intuitionistically valid.

11(b). Let $\Box\phi$ mean that "agent Q believes ϕ". Then the formula scheme $\Box\phi \vee \Box\neg\phi$ means

"For any formula ϕ, agent Q either believes ϕ, or she believes $\neg\phi$."

13(a). Reading $\Box\phi$ as "agent Q knows ϕ", we read $R(x, y)$ as "y could be the actual world according to agent Q's knowledge at x".

(i) R should indeed be symmetric. Suppose $R(x, y)$, but not $R(y, x)$. Then, as far as Q's knowledge at x is concerned, y could be the actual world, but his knowledge at y rules out the possibility of x being the actual world. Therefore, at y he knows something preventing x from being actual; so at x he knows he knows something preventing x from being actual. Therefore, he (simply) knows something preventing x being actual, i.e. he knows something which is false. This contradicts our model of knowledge.

(ii) The totality of R means that for all worlds x and y we have $R(x, y)$ or $R(y, x)$. But x and y may contain mutually exclusive knowledge (e.g. agent Q may know all prime ministers of Canada at world x, but fail to remember some of them at world y). In that case, neither $R(x, y)$ nor $R(y, x)$ are valid, so R is not total in general.

13(c). Let $\Box\phi$ model "always in the future, ϕ holds". Note that $R(x, y)$ here means "y is in the future of x".

- Whether R is reflexive, or not, depends on whether the present is part of the future (only then is R reflexive).
- R is not symmetric, since there is no symmetry between the past and the future.
- Assuming that time does not halt, R should be serial: each point in time, x, has some future point, y.
- R is transitive: if y is in the future of x and z in the future of y, then z is in the future of x as well.
- R is not Euclidean; for example, if x is the year 2005 and $y \stackrel{\text{def}}{=} 2031$, $z \stackrel{\text{def}}{=} 2012$, then y and z are in the future of x, but z is not a future of y.
- Assuming that time goes on without end, R is not functional, for each point in time will have many distinct future points in time.
- Assuming the ordinary notion of time, namely that future points are either identical, or one occurs prior to the other, the relation R is linear.
- Whether R is total, or not, depends on the answer to the question whether the present is part of the future (only then is R total).
- R is not an equivalence relation since it is not symmetric.

16(b). A frame $\mathcal{F} = (W, R)$ satisfies $\Box\bot$ iff each world $w \in W$ satisfies this formula. (Why don't we have to consider labelling functions here?) But $w \Vdash \Box\bot$ holds iff there is no w' with $R(w, w')$, for $\bot$ is satisfied by no world whatsoever. Thus, $\Box\bot$ corresponds to R being the empty relation (= no two worlds are related).

16(c). $\Diamond\Box\phi \to \Box\Diamond\phi$ corresponds to *weakly directed*: if $R(x, y)$ and $R(x, z)$, then there exists a point u such that $R(y, u)$ and $R(z, u)$. The proof of this is in similar style to Theorem 5.13.

17. We seek a formula scheme ϕ which corresponds to *density*: for all $x, z \in W$, there exists some $z \in W$ with $R(x, y)$ and $R(y, z)$. We claim that

$$\Diamond\phi \to \Diamond\Diamond\phi$$

is such a formula scheme.

(i) Let $\mathcal{F} = (W, R)$ be a frame which satisfies $\Diamond\phi \to \Diamond\Diamond\phi$. Suppose that $R(x, z)$ holds. Let L be any labelling function with respect to which we have $z \Vdash p$, but $w \nVdash \phi$ for any other world. Then $x \Vdash \Diamond\phi$ as $R(x, z)$. Since the frame satisfies $\Diamond\phi \to \Diamond\Diamond\phi$ we obtain that $x \Vdash \Diamond\Diamond\phi$ holds. Thus, there is some $y \in W$ with $R(x, y)$ and $y \Vdash \Diamond\phi$. But the latter can only mean $R(y, z)$ due to the choice of our labelling function L. Thus, R is dense.

(ii) Let $\mathcal{F} = (W, R)$ be a frame such that R is dense. Let $x \in W$ be given. We need to show that $x \Vdash \Diamond\phi \to \Diamond\Diamond\phi$ holds for any choice of labelling function L. This is certainly the case if $x \nVdash \Diamond\phi$. But if $x \Vdash \Diamond\phi$, then there has to exist a world, z, such that $R(x, z)$ and $z \Vdash \phi$. By density of R, $R(x, z)$ implies the existence of some world y with $R(x, y)$ and $R(y, z)$. Since $z \Vdash \phi$, the relation $R(y, z)$ implies $y \Vdash \Diamond\phi$. But the latter guarantees $x \Vdash \Diamond\Diamond\phi$ as $R(x, y)$ holds.

EXERCISES 5.4 (p.353)

1(a). We prove the validity of $\Box(p \to q) \vdash_{\mathrm{K}} \Box p \to \Box q$ by

$\Box(p \to q)$	prem
$\Box p$	ass
p	$\Box$e 2
$p \to q$	$\Box$e 1
q	$\to$e 4, 3
$\Box q$	$\Box$i 3–5
$\Box p \to \Box q$	$\to$i 2–6

1(c). We prove the validity of $\vdash_{\mathrm{K}} \Box(p \to q) \wedge \Box(q \to r) \to \Box(p \to r)$ by

$\Box(p \to q) \wedge \Box(q \to r)$	ass
$\Box(p \to q)$	$\wedge$e$_1$ 1
$\Box(q \to r)$	$\wedge$e$_2$ 1
p	ass
$p \to q$	$\Box$e 2
q	$\to$e 6, 5
$q \to r$	$\Box$e 3
r	$\to$e 8, 7
$p \to r$	$\to$i 5–9
$\Box(p \to r)$	$\Box$i 4–10
$\Box(p \to q) \wedge \Box(q \to r) \to \Box(p \to r)$	$\to$i 1–11

We left line 4 empty since our macros for producing boxes otherwise causes the two boxes to overlap; we will use this "hack" in subsequent examples as well.

1(f). We prove the validity of $\Diamond(p \to q) \vdash_{\mathrm{K}} \Box p \to \Diamond q$ by

$\neg\Box\neg(p \to q)$	prem
$\Box p$	ass
$\Box\neg q$	ass
$p \to q$	ass
p	$\Box$e 2
q	$\to$e 5, 6
$\neg q$	$\Box$e 3
$\bot$	$\neg$e 7, 8
$\neg(p \to q)$	$\to$i 5–9
$\Box\neg(p \to q)$	$\Box$i 4–10
$\bot$	$\neg$e 11, 1
$\neg\Box\neg q$	$\neg$i 3–12
$\Box p \to \neg\Box\neg q$	$\to$i 2–13

2(c). We prove the validity of $\vdash_{\mathrm{KT45}} \Diamond\Box p \leftrightarrow \Box p$ by proving $\vdash_{\mathrm{KT45}} \Diamond\Box p \to \Box p$ and $\vdash_{\mathrm{KT45}} \Box p \to \Diamond\Box p$ separately:

(i) We prove the validity of $\vdash_{\mathrm{KT45}} \Diamond\Box p \to \Box p$ by

$\neg\Box\neg\Box p$	ass
$\Box\neg\Box\neg\Box p$	**5** 1
$\neg\Box p$	ass
$\Box\neg\Box p$	**5** 4
$\neg\Box\neg\Box p$	$\Box$e 2
$\bot$	$\neg$e 5, 6
$\Box p$	RAA 4–7
p	**T** 8
$\Box p$	$\Box$i 3–9
$\neg\Box\neg\Box p \to \Box p$	$\to$e 1–10

Notice that this half of the equivalence did not need axiom **4**, so we actually proved the stronger $\vdash_{\mathrm{KT5}} \Diamond\Box p \to \Box p$.

(ii) We prove the validity of $\vdash_{\mathrm{KT45}} \Box p \to \Diamond\Box p$ by

$\Box p$	ass
$\Box\neg\Box p$	ass
$\neg\Box p$	**T** 2
$\bot$	$\neg$e 1, 3
$\neg\Box\neg\Box p$	$\to$i 2–4
$\Box p \to \neg\Box\neg\Box p$	$\to$i 1–5

Since this proof also never required the axiom **4**, we conclude that the equivalence between $\Box p$ and $\Diamond\Box p$ holds already with respect to KT5.

EXERCISES 5.5 (p.354)

3(c). $K_1 p \vee K_2 q$.

3(g). $K_1 K_2 p$.

3(k). $(K_1 p \wedge K_1 \neg q) \vee (K_2 p \wedge K_2 \neg q) \vee \cdots \vee (K_n p \wedge K_n \neg q)$, where the agents are numbered from 1 to n.

4(d). By definition of E, the relation $x \Vdash E(p \vee q)$ holds iff $x \Vdash K_1(p \vee q) \wedge K_2(p \vee q) \wedge K_3(p \vee q)$ is true. For all $i = 1, 2, 3$, we have that $R_i(x_3, w)$ implies $w \in \{x_1, x_2\}$. But since $x_1 \Vdash p \vee q$ and $x_2 \Vdash p \vee q$, we infer that $x \Vdash E(p \vee q)$ indeed holds.

4(i). By definition of C, we have $x_6 \Vdash C\neg q$ iff $x_6 \Vdash E^k \neg q$ holds for each $k = 1, 2, \ldots$, where E^k is $EE \ldots E$ (k times).

(i) Now $x_6 \Vdash E^1 \neg q$ simply means $x_6 \Vdash K_1\neg q \wedge K_2\neg q \wedge K_3\neg q$. The latter holds since, for $i = 1, 2, 3$, the relation $R_i(x_6, w)$ implies that w equals x_5; and we do have $x_5 \Vdash \neg q$.

(ii) We have $x_6 \Vdash E^2\neg q$ iff we have $x_6 \Vdash K_1 E\neg q \wedge K_2 E\neg q \wedge K_3 E\neg q$. As in (i), we see that this amounts to checking whether x_5 satisfies a certain formula, but this time it is $E\neg q$ and not $\neg q$. However, $x_5 \Vdash E\neg q$ fails to hold. For example, we have $R_2(x_5, x_4)$, but $x_4 \nVdash \neg q$ since $x_4 \Vdash q$.

Therefore, we infer $x_6 \nVdash E^2\neg q$, and from that $x_6 \nVdash C\neg q$ follows.

6. • Let x be any world in any model of KT45. To show $x \Vdash C_G\phi \to C_G C_g \phi$, it suffices to assume $x \Vdash C_G\phi$ and show $x \Vdash C_G C_G \phi$. The latter is shown if we can prove that $x \Vdash E_G^l C_G \phi$ for any $l \leq 1$.

We make use of the second part of Theorem 5.26. Thus, if y is any world which is G-reachable from x within l steps, we have to argue that $y \Vdash C_G\phi$, i.e. $y \Vdash E_G^k\phi$ for all $k \geq 1$. Fix any $k \geq 1$. If z is G-reachable from y within k steps, we are done if $z \Vdash \phi$. But since y is G-reachable from x within l steps, we conclude that z is G-reachable from x within $l + k$ steps. But then our assumption that $x \Vdash C_G\phi$ implies $x \Vdash E_G^{l+k}\phi$ which renders $z \Vdash \phi$. Thus, $C_G\phi \rightarrow C_GC_g\phi$ is valid in KT45.

- To show that $\neg C_G\phi \rightarrow C_G\neg C_G\phi$ is valid in KT45, it suffices to consider an arbitrary world x in an arbitrary model of KT45 such that $x \Vdash \neg C_G\phi$. We then only have to show that $x \Vdash C_G\neg C_G\phi$ follows. We make crucial use of the second part of Theorem 5.26, and apply *proof by contradiction*: Assume that $x \nVdash C_G\neg C_G\phi$. By Theorem 5.26, there exists a world y which is G-reachable by x such that $y \nVdash \neg C_G\phi$, i.e. $y \Vdash C_G\phi$. Since G is non-empty (what does $\neg C_G\phi \rightarrow C_G\neg C_G\phi$ mean if G is empty, and why it is then valid?), we know that the relation

$$\bigcup_{i \in G} R_i \tag{5.1}$$

is an equivalence relation (see the discussion in the textbook after Theorem 5.26). But then x and y are related via this equivalence relation and one satisfies $C_G\phi$, whereas the other one does not. This is a contradiction. (Why?)

10(c). Again, we prove this via showing two separate implications.

(i) We prove $Cp \rightarrow K_iC_p$ by

Cp	ass
CCp	$C\mathbf{4}$ 1
ECp	CE 2
K_iCp	EK_i 3
$Cp \rightarrow K_iCp$	$\rightarrow$i 1–4

(ii) This follows simply by the rule KT.

10(e). The formula scheme $\neg\phi \rightarrow K_i\neg K_i\phi$ means

"If ϕ is false, then agent i knows that he does not know ϕ."

Here is a proof:

$\neg\phi$	ass
$K_i\phi$	ass
ϕ	KT 2
$\bot$	$\neg$e 3, 1
$\neg K_i\phi$	$\neg$i 2–4
$K_i\neg K_i\phi$	$K5$ 5
$\neg\phi \rightarrow K_i\neg K_i\phi$	$\rightarrow$i 1–6

10(g)

- The proof strategy for showing the validity of $\neg K_1\neg K_1\phi \rightarrow K_1\phi$ is to use proof by contradiction with $\neg K_1\phi$ as an assumption which then renders $K_1\neg K_1\phi$ by $K4$ applied to agent 1 and so $\bot$ can be inferred.
- The other direction is reasoned in a similar way: use proof by contradiction and assume $K_1\neg K_1\phi$ which is demoted to $\neg K_1\phi$ by KT applied to agent 1 and so $\bot$ can be inferred.

6

Binary decision diagrams

EXERCISES 6.1 (p.398)

3. If we identify p and q with x and y, respectively, then there are *infinitely many* boolean formulas $f(x, y)$ in terms of $\cdot$, $+$, -, 0 and 1 such that f has a truth table corresponding to the one for $p \to q$. For example,

$$f(x, y) \stackrel{\text{def}}{=} \overline{x} + y$$

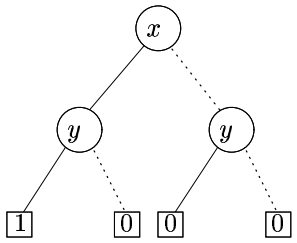
is one such formula and

$$f(x, y) \stackrel{\text{def}}{=} z + \overline{x \cdot \overline{y}} + 0 + \overline{z}$$

is yet another.

EXERCISES 6.2 (p.398)

1. If we swap the dashed and solid lines in the binary decision tree of Figure 6.2, we obtain the binary decision tree



A truth table corresponding to this binary decision tree is

x	y	$f(x,y)$
1	1	1
0	1	0
1	0	0
0	0	0

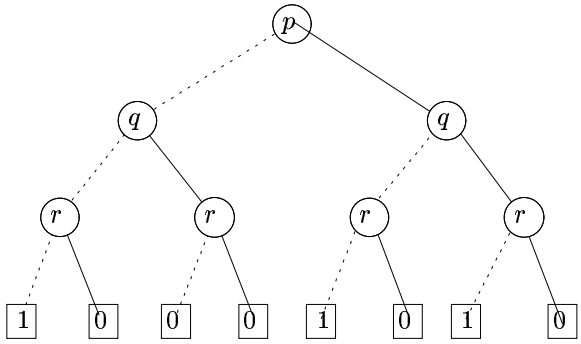
A formula with this truth table is $f(x,y) \stackrel{\text{def}}{=} x \cdot y$, which corresponds to "logical and".

2. The solution depends on

(i) whether we actually impose an ordering on p, q and r as we construct the binary decision tree;

(ii) and, if so, what ordering we would actually choose.

We choose an ordering, namely the obvious one of "p before q before r". The resulting binary decision tree is



Note that the truth values of ϕ, read downwards in the table, correspond to the leaf values of the binary decision tree, if read from the left to the right.

EXERCISES 6.3 (p.399)

1. Let B be any BDD. We appeal to Definition 6.3 which also is meant to apply to BDDs.

(i) In that Definition we see that, in order to evaluate the BDD B as a boolean function, we take its argument which uniquely determines a path along B and the result is the value of the leaf node it reaches. So if we redirect edges to identical leaf nodes, this cannot change the value computed in this manner.

(ii) Suppose that both outgoing edges of node n point to the same node m. Then any evaluation path of B which passes through node n has to pass through node m, independent of the value of n's label. Thus, we may savely redirect incoming edges of n to m without changing the leaf nodes that one could reach on such paths

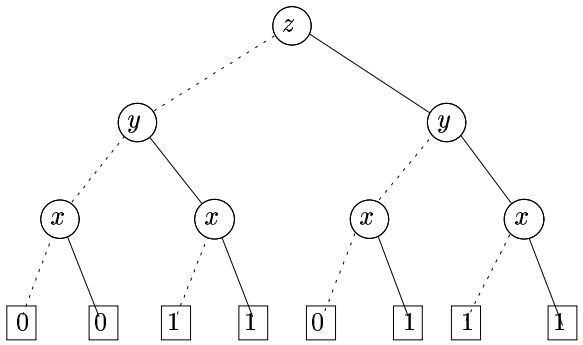
(iii) Structurally identical BDDs clearly induce the same boolean function. So if we redirect all incoming edges of the root node of one BDD to the root node of an identical one, this cannot change the value of leaf nodes that are reached by given arguments.

2(a).

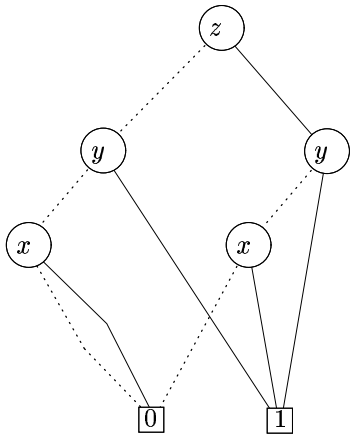
x	y	z	$f(x,y)$
1	1	1	1
1	1	0	1
1	0	1	1
0	1	1	1
1	0	0	0
0	1	0	1
0	0	1	0
0	0	0	0

EXERCISES 6.4 (p.400)

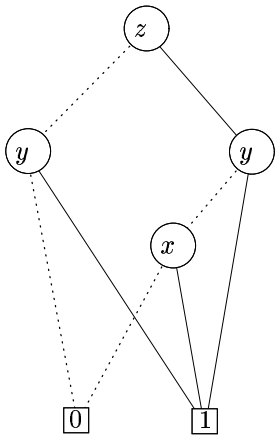
1(a). First, we use the truth table of item 2(a) in Exercises 6.3 to construct the binary decision tree for it:



Second, we apply the reductions **C1-C3** to this tree until a reduced BDD is found. Two x-nodes perform a redundant test on 1 and are themselves redundant non-terminals. Thus, these two nodes may be replaced with a leaf node for 1. We also remove duplicate terminal and obtain

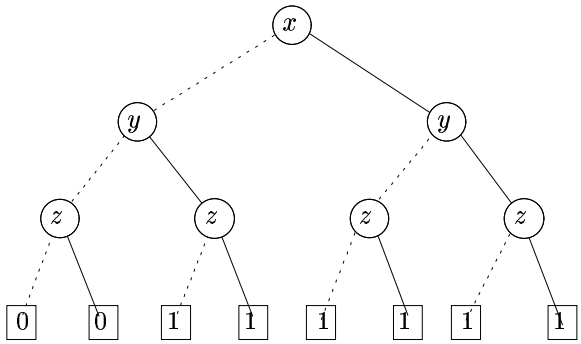


Next, we remove the remaining redundant test for one x-node and get

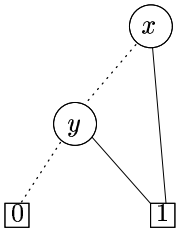


This BDD is reduced.

3(b). A binary decision tree for $f(x, y) \stackrel{\text{def}}{=} x + y$ in the ordering $[x, y, z]$ is

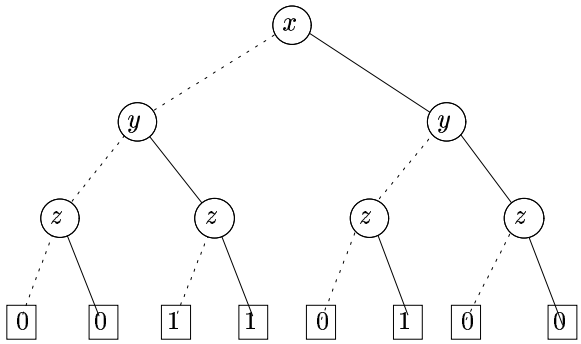


We immediately see that all z-nodes are redundant tests, so they may all be eliminated. Further, the rightmost y-node is a redundant non-terminal, so we get

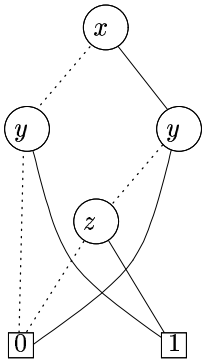


which is reduced.

3(d). The binary decision tree for $f(x,y,z) \stackrel{\text{def}}{=} (x \oplus y) \cdot (\overline{x} + z)$ in the ordering $[x,y,z]$ is



Three z-nodes are redundant tests and we may share two which are identical. In conjunction with the removal of duplicate terminals, this gives us



which is reduced.

EXERCISES 6.5 (p.401)

6(a).

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	1
0	1	0	0	0
1	0	0	0	0
1	1	0	0	1
0	0	0	1	0
0	1	0	1	1
1	0	0	1	1
1	1	0	1	0
0	0	1	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	0
0	0	1	1	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

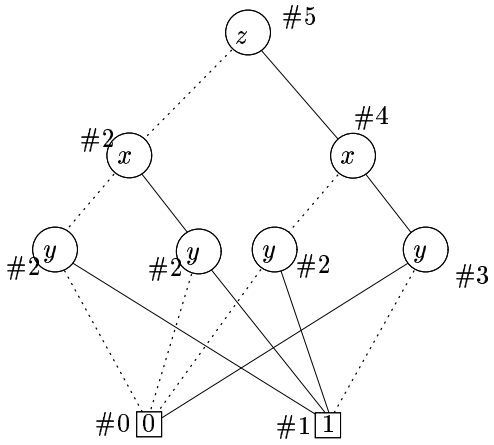
6(b). We proceed as in Chapter 1 (see the discussion after Proposition 1.45):

$$(x_1 + \overline{x}_2 + x_3 + x_4) \cdot (\overline{x}_1 + x_2 + x_3 + x_4) \cdot$$
$$(x_1 + x_2 + x_3 + \overline{x}_4) \cdot (\overline{x}_1 + \overline{x}_2 + x_3 + \overline{x}_4) \cdot$$
$$(x_1 + x_2 + \overline{x}_3 + x_4) \cdot (\overline{x}_1 + \overline{x}_2 + \overline{x}_3 + x_4) \cdot$$
$$(x_1 + \overline{x}_2 + \overline{x}_3 + \overline{x}_4) \cdot (\overline{x}_1 + x_2 + \overline{x}_3 + \overline{x}_4).$$

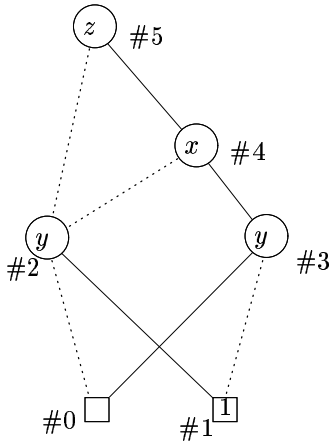
6(c). In general, if we consider the even-bit parity function in n variables ($n = 4$ in item 6(a) and 6(b)), then the OBDD corresponding to Figure 6.11 has $2n + 1$ nodes. The corresponding conjunctive normalform, however, has 2^{n-1} many conjuncts, unacceptable for even moderate values of n.

EXERCISES 6.6 (p.402)

1(c). The BDD



is the result of applying the labels to the BDD and



is the resulting reduced OBDD with unique labels.

EXERCISES 6.7 (p.402)

5. The expression $f \to (g, h)$ is a boolean function in three arguments (f, g and h) which is equivalent to g if f evaluates to 1; otherwise, it is equivalent to h.

(a) Using $+$, $\oplus$ and , we may define this as

$$f \cdot g + \overline{f} \cdot h.$$

For if f evaluates to 1, the expression is equivalent to $1 \cdot g + \overline{1} \cdot h$ which in turn is equivalent to g. Conversely, if f evaluates to 0, then we get $0 \cdot g + \overline{0} \cdot h$ which is eqivalent to h.

(b) We set $f_n \stackrel{\text{def}}{=} x \to (f_{\text{lo}(n)}, f_{\text{hi}(n)})$.

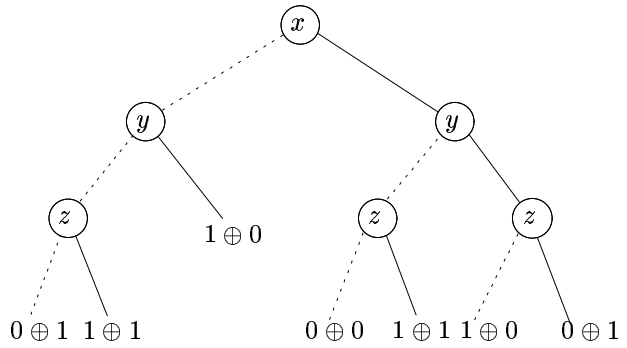
(c) We use mathematical induction on the maximal number, l_n, of edges we need to reach a leaf node from node n.

- If $l_n = 1$, then the outcoming edges of n point to leaf nodes. So for f_n being $x \to (f_{\mathrm{lo}(n)}, f_{\mathrm{hi}(n)})$, we have $f_{\mathrm{lo}(n)}, f_{\mathrm{hi}(n)} \in \{0, 1\}$. Thus, f_n is independent of any y, different from x, which occurs before x in the ordering.
- We assume that the claim holds for all nodes m with $l_m \leq k$. Let n be a node such that $l_n = k{+}1$. Then $l_{\mathrm{lo}(n)}, l_{\mathrm{hi}(n)} \leq k$ follow. By our induction hypothesis, $f_{\mathrm{lo}(n)}$ and $f_{\mathrm{hi}(n)}$ are independent from any y, different from x, which occurs before x in the ordering. But then f_n is also independent of all such y as f_n equals $x \to (f_{\mathrm{lo}(n)}, f_{\mathrm{hi}(n)})$.

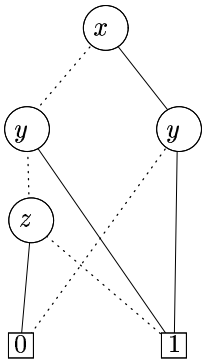
10(a).(i) Let the boolean formula $(\overline{f} \oplus g) + c$ be valid. If c evaluates to 0, then that formula is equivalent to $\overline{f} \oplus g$ and since the former is valid, we conclude that $\overline{f} \oplus g$ is valid as well. By the definition of $\oplus$ is means that $\overline{f}$ and g can never evaluate to the same value for a given argument. Thus, f and g always evaluate to the same value, i.e. they are equivalent.

(ii) Suppose that f and g are equivalent on all arguments for which c evaluates to 0. If $(\overline{f} \oplus g) + c$ is not valid, then there is some argument for which is evaluates to 0. But then c evaluates to 0 as well. By our assumption, f and g then evaluate to the same value, but this contradicts the fact that $\overline{f} \oplus g$ evaluates to 0.

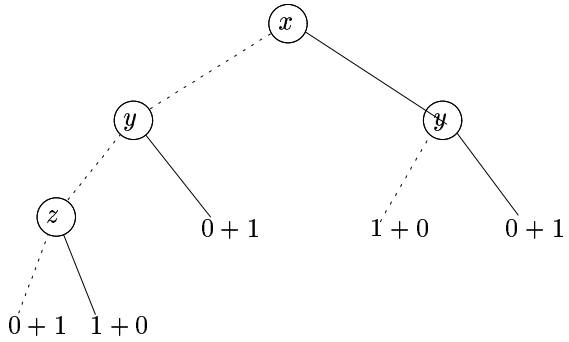
10(b). We first apply $\oplus$ to $B_{\overline{f}}$ [note: the BDD for $\overline{f}$] and B_g, yielding:



Reducing, we obtain:



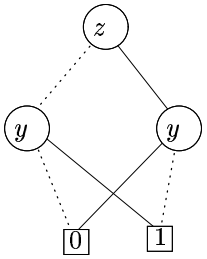
Now apply + to B_c and this BDD, resulting in:



We find that this reduces to the constant BDD 1.

EXERCISES 6.8 (p.404)

1(b). The reduced OBDD for $f[1/x]$ is



1(d). The reduced OBDD for $f[0/z]$ is the one which represents the function y, since we remove the top z-node and keep the left, reduced, subOBDD.

2(a). The expression $f \equiv \overline{x} \cdot f[\overline{g}/x] + x \cdot f[g/x]$ does it.

2(b). Replacing a node n, labelled with x, with the BDD for g will almost always be inconsistent with the chosen global variable ordering.

2(c). The expression $g \to (f[1/x], f[0/x])$ is equivalent to $f[g/x]$ and can be implemented via the operator described in item 5(a) of Exercises 6.7.

EXERCISES 6.9 (p.405)

3(a).(i) Let ϕ be satisfiable. Then there is a valuation ρ such that ϕ evaluates to 1 with respect to ρ. Since

$$\exists x.\phi \stackrel{\text{def}}{=} \phi[0/x] + \phi[1/x]$$

we infer that $\exists x.\phi$ evaluates to 1 as well under ρ, for the latter has to assign 0 or 1 to x.

(ii) Let $\exists x.\phi$ be satisfiable. Then there exists a valuation ρ' such that $\phi[0/x] + \phi[1/x]$ evaluates to 1 with respect to ρ'.

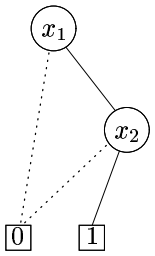
Case 1: If $\phi[0/x]$ computes to 1 under ρ', then ϕ computes to 1 under $\rho'[x \mapsto 0]$, where $\rho'[x \mapsto 0]$ behaves like ρ', but that it assigns 0 to x.

Case 2: If $\phi[1/x]$ computes to 1 under ρ', then ϕ computes to 1 under $\rho'[x \mapsto 1]$, where $\rho'[x \mapsto 1]$ behaves like ρ', but that it assigns 1 to x.

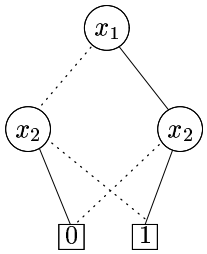
Since at least one of these cases has to apply (why?), this concludes the argument.

EXERCISES 6.11 (p.407)

1.(i) We identify the subset of states $\{s_0, s_1\}$ with the boolean function $x_1 \cdot x_2 + x_1 \cdot \overline{x}_2$ whose reduced OBDD with the ordering $[x_1, x_2]$ is



(ii) The set $\{s_0, s_2\}$ corresponds to $x_1 \cdot x_2 + \overline{x}_1 \cdot \overline{x}_2$ whose reduced OBDD with the ordering $[x_1, x_2]$ is



EXERCISES 6.13 (p.408)

1(a). (Bonus) The transition function of this circuit is

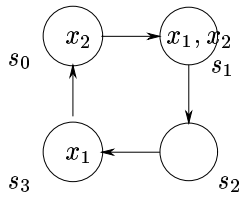
$$(x_1' \leftrightarrow \overline{x}_1) \cdot (x_2' \leftrightarrow x_1 \oplus x_2).$$

Notice the this describes a *determinstic* system. If we start this off in the state 01, the system evolves as in

x_1	x_2
0	1
1	1
0	0
1	0
0	1

at which point it finished a full cycle. Thus, this implements a counter "modulo 4".

1(b). (Bonus) The CTL model is given by



EXERCISES 6.14 (p.409)

2.(i) If $m = 0$, then $\nu_0 Z.Z \stackrel{\text{def}}{=} 1$ ensures $\rho \vDash \nu_0 Z.Z$ by the definition of $\vDash$.

(ii) If $\rho \vDash \nu_m Z.Z$, then we also get $\rho \vDash \nu_{m+1} Z.Z$, for $\nu_{m+1} Z.Z$ is defined to be $Z[\nu_m Z.Z/Z]$ which is just $\nu_m Z.Z$ again.

(Can you argue that $\nu_m Z.Z$ equals 1 for all $m \geq 0$?)

EXERCISES 6.15 (p.410)

8(a). The function $f^{\mathrm{EX}\,(x_1 \wedge \neg x_2}$ is defined as $\exists \hat{x}'.(f^{\rightarrow} \cdot f^{x_1 \wedge \neg x_2}[\hat{x} := \hat{x}']$ which equals $\exists \hat{x}'.((x_1' \leftrightarrow \overline{x}_1) \cdot (x_2' \leftrightarrow x_1 \oplus x_2) \cdot (x_1' \cdot \overline{x}_2'))$.

8(c). The function $f^{\mathrm{AG}\,(\mathrm{AF}\,\neg x_1 \wedge \neg x_2)}$ should compute 1 exactly for those states s which satisfy the CTL formula $\mathrm{AG}\,(\mathrm{AF}\,\neg x_1 \wedge \neg x_2)$. The latter formula says, at state s, that $\neg x_1 \wedge \neg x_2$ is true infinitely often on all computation paths that begin in s. But this $\neg x_1 \wedge \neg x_2$ is true exactly at state s_2, this formula is satisfied by all states, for each infinite path has to pass through s_2 infinitely often. Thus, we may choose as boolean formula without foxed points simply 1.

EXERCISES 6.16 (p.411)

1. (Bonus) In Equation (6.27), the function $f^{\mathrm{E}_C \mathrm{G}\phi}$ is defined in terms of `checkEX` and `checkEU`, but these two patterns are defined in a way which does not depend in fair (see the Equations (6.25) and (6.26)).