CS-E4800 Artificial Intelligence: Exercises

Jussi Rintanen
Department of Computer Science
Aalto University
Helsinki, Finland

February 5, 2026

Contents

1 Introduction

This document contains exercises for the Aalto University course *CS-E4800 Artificial Intelligence*. The course has obligatory weekly exercises done on the course's website, to *evaluate* the understanding of each week's study material.

To test your command of the course material *before* proceeding to the on-line exercises, *prepare* for that evaluation, read through the course material carefully, and confirm your understanding by using the questions in the present document.

Model answers to all of the exercises in this document are given in the end of each section.

Many thanks for all those who have pointed out errors and inconsistencies in the document, including Mojtaba Elahi, Kasper Kivimäki and Aleksi Lankinen.

2 Exercises: Propositional Logic

2.1 Boolean Functions

Exercise 2.1

Consider atomic propositions x_1, x_2, x_3 such that x_i denotes that "employee i is working". Give formulas for the following requirements.

1. At least one of the employees 1, 2 and 3 is working.
2. At least two of the employees 1, 2, and 3 are working.
3. Not all of the employees are working.

Exercise 2.2

You have to organize the party, but the relationships between your friends are a bit complicated, and you have to be very careful about who to invite, to avoid embarrassing situations, or worse.

Formalize the following constraints as formulas in the propositional logic. Use the initial of each potentials guest's names as the name of an atomic proposition.

1. Both Betty and Cathy are your friends, but Cathy is much closer, so you cannot invite Betty without inviting also Cathy.
2. Dimitri, Fatima and Ginjiro are a bit noisy, so you want to invite at most two of them.
3. Dimitri and Betty are not in good terms, so you cannot invite both.
4. No point inviting Cathy unless Eva is also invited, as the former is shy and would not come if Eva is not there.
5. You have to invite both or neither of Andy and Eva, as they are a couple.
6. Andy and Fatima were married before, so you should not invite both of them without also inviting Fatima's new husband Ginjiro.
7. You want somebody to play piano in your party, and only Betty and Fatima can do it, but due to their rivalry, the one who is not playing, should not be there at all.

If you have additionally the constraint that a party is not a proper party unless there are at least five guests, what are your options?

Exercise 2.3

The following is an *incrementer circuit* that computes the function $f(x) = x+1$ for 4-bit integers. So 0000 is incremented to 0001, 0010 is incremented to 0011, 0011 is incremented to 0100, and so on.

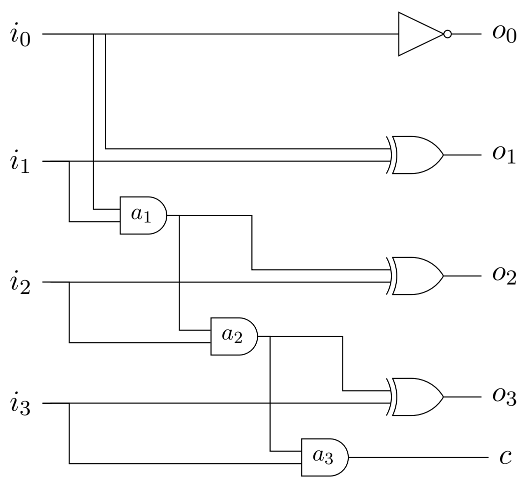
We denote the input bits by i_0, i_1, i_2, i_3 and the output bits by o_0, o_1, o_2, o_3. The bit 0 is the *least significant bit* and the bit 3 is the *most significant bit*. The bit 0 represents 1, the bit 1 represents 2, the bit 2 represents 4, and the bit 3 represents 8, so that these four bits can represent all values from 0 to 15. Additionally we have the *carry* bit c, which is essentially the output bit of value $2^4 = 16$.

An output bit is 1 if and only if exactly one of the following holds.

- The corresponding input bit is 1.
- All preceding input bits are 1.

So, for example, output o_2 has value 1 if either $i_2 = 1$, or $i_0 = i_1 = 1$, but not both. A special case is o_0, where the above condition means that $o_0 = 1$ iff $i_0 = 0$.

The *carry bit* c has value 1 if the result of the incrementation does not fit in 4 bits, that is, the input bits are 1111 and the incrementation overflows, with the carry bit set, and output bits with value 0000.



1. For output bit o_0 the value computed by the incrementer circuit is represented by the formula $\neg i_0$. Express similarly the value of each of the rest of the outputs o_1, o_2, o_3, c as a propositional formula that only has occurrences of the atomic propositions i_0, i_1, i_2, i_3 that describe the values of the input bits. Use the connectives $\oplus$ for *exclusive or* (*xor*) and $\wedge$ for *conjunction* (*and*).
2. The combined size of the formulas for the outputs expressed in terms of the inputs i_0, i_1, i_2, i_3 is *quadratic* $O(k^2)$ in the number of inputs and in the size of the circuit. A logic representation for the incrementer circuit that more closely reflects the actual size of the circuit, in terms of the number of logic gates, represents the values of the outputs of at least some of the gates explicitly.

   Express the output values of the AND gates a_1, a_2, a_3 in terms of the values of the inputs and the outputs of the preceding AND gates, and then express the values of the outputs in terms of the inputs and the outputs of the relevant AND gates. Now use atomic propositions $i_0, i_1, i_2, i_3, a_1, a_2, a_3, o_0, o_1, o_2, o_3, c$, and express the output values of the circuit as equivalences ... $\leftrightarrow o_0$, ..., ... $\leftrightarrow o_3$, ... $\leftrightarrow c$, and similarly express the output values of the AND gates as equivalences ... $\leftrightarrow a_1$, ... $\leftrightarrow a_2$, ... $\leftrightarrow a_3$.

2.2 Reasoning with Equivalences

Exercise 2.4

Show that the following equivalences hold, by using the equivalences in Table 1 (see the Appendix). Start from the formula on the left-hand-side of $\equiv$, and derive the formula on the right-hand-side by applying a chain of one or more of the already-known equivalences.

1. $(a \wedge (b \wedge c)) \equiv ((c \wedge a) \wedge b)$
2. $\top \vee \bot \vee \top \equiv \top$

Exercise 2.5

Simplify the following formulas. For example, eliminate the logical constants $\top$ and $\bot$, if possible, or otherwise make the formulas simpler. Use the equivalences given in Table 1.

1. $\top \wedge (\bot \vee a)$
2. $(b \wedge \top) \vee \bot$
3. $\bot \vee (c \wedge \bot)$
4. $a \wedge (\neg a \vee b)$
5. $a \wedge (\neg a \vee b) \wedge (\neg b \vee c)$

2.3 Satisfiability

Questions about satisfiability of a given formula can always be answered by going through all valuations, and seeing if the formula is true in at least one of them. When the number of atomic propositions is high, this is too much work, and one can try to reason about the possibility of a valuation that makes the formula true.

Exercise 2.6

Are the following formulas satisfiable?

1. $A \land B \land C$
2. $A \land (B \lor \neg A)$
3. $(A \rightarrow B) \land (\neg A \rightarrow B)$
4. $(A \lor B) \land (A \rightarrow \neg C) \land (B \rightarrow \neg C) \land C$

Exercise 2.7

Do the following hold? Give a counterexample or prove it.

1. If a set $\Sigma = \{\phi_1, \ldots, \phi_n\}$ is satisfiable, then any of its subsets $\Sigma_0 \subseteq \Sigma$ is also satisfiable.
2. If the formulas ϕ_1 and ϕ_2 are satisfiable, then also $\phi_1 \land \phi_2$ is satisfiable.
3. If the formulas ϕ_1 and ϕ_2 are satisfiable, then also $\phi_1 \lor \phi_2$ is satisfiable.
4. If at least one of the formulas ϕ_1 and ϕ_2 is satisfiable, then also $\phi_1 \lor \phi_2$ is satisfiable.

Exercise 2.8

Are the following formulas satisfiable?

1. $(A \lor B) \land (\neg C \lor \neg D) \land (E \lor F) \land (\neg G \lor \neg H) \land (I \lor J)$
2. $(A \land B) \lor (\neg A \land \neg B) \lor (B \land \neg C) \lor (\neg A \land \neg C) \lor (\neg A \land C)$

Exercise 2.9

Take the solution to part 2 of Exercise 2.3 and denote it by ϕ.

1. Is the formula $\phi \land i_0 \land o_0$ satisfiable?
2. Is the formula $\phi \land i_1 \land o_1$ satisfiable?

2.4 Logical Consequence and Equivalence

Questions of logical equivalence can be answered by looking at all possible valuations (e.g. by building the truth-table), or by applying known equivalences such as those given in Table 1. Equivalences may also help in determining whether some logical consequence holds, if used for rewriting the formula to a form where things are more obviously visible.

Reasoning with implications can sometimes be helped by remembering that $\alpha \rightarrow \beta$ is equivalent to $\neg\alpha \lor \beta$.

Exercise 2.10

Do the following hold?

1. $A \land (\neg B \rightarrow \neg A) \models B$
2. $A \land \phi \models A$ for any formula ϕ
3. $A \lor \phi \models A$ for any formula ϕ
4. $A \land (A \rightarrow B) \land (B \rightarrow C) \land (C \rightarrow D) \land (D \rightarrow E) \models E$
5. $(A \rightarrow B) \land (B \rightarrow C) \land (C \rightarrow D) \land (D \rightarrow E) \land \neg E \models \neg B$
6. $(A \lor B) \land (A \rightarrow C) \land (B \rightarrow C) \models C$

Exercise 2.11

If $\alpha \models \beta$ holds, does also $\neg\beta \models \neg\alpha$ hold as well? Give a counterexample, or prove that it is so.

Exercise 2.12

Do the following hold?

1. $A \to A \equiv A$
2. $A \to (A \to A) \equiv \top$
3. $B \to (A \to A) \equiv \top$
4. $A \to (B \to C) \equiv (A \wedge B) \to C$
5. $(A \to A) \to A \equiv A$

2.5 Model-Counting

Model-counting means *counting* the number of satisfying valuations of a formula. For example, the formula $A \vee B$ is satisfied by three of the four valuations over A and B, all except $A = 0, B = 0$. Note that a formula is *satisfiable* if and only if its model-count is > 0.

Exercise 2.13

What are the model-counts of the following formulas?

1. $A \wedge \neg A$
2. $A \to B$
3. $(A \vee \neg A) \vee (B \wedge (B \to C) \wedge (C \to D) \wedge \neg D)$
4. $(A \vee B) \wedge (C \vee D) \wedge (E \vee F) \wedge (G \vee H)$
5. $\neg((A \vee B) \wedge (C \vee D) \wedge (E \vee F) \wedge (G \vee H))$
6. $A \wedge (A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to E) \wedge (E \to F)$

2.6 Solutions

Answer of exercise 2.1

1. $x_1 \vee x_2 \vee x_3$
2. $(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$
3. $\neg(x_1 \wedge x_2 \wedge x_3)$, which is logically equivalent to $\neg x_1 \vee \neg x_2 \vee \neg x_3$ (“Not all of them” equals “At least one is not”)

Answer of exercise 2.2

1. $B \to C$
2. $\neg D \vee \neg F \vee \neg G$, or, equivalently, $\neg(D \wedge F \wedge G)$ (A general scheme for choosing at most N out of a set of M would be to go through all cardinality $k = |M| - |N|$ subsets $\{e_1, \ldots, e_k\}$, and form a long disjunction, with every possible $\neg e_1 \wedge \cdots \wedge \neg e_k$ as disjuncts.)
3. $\neg(D \wedge B)$
4. $C \to E$
5. $A \leftrightarrow E$
6. $(A \wedge F) \to G$
7. $B \leftrightarrow \neg F$

The only proper parties are ABCEG and ACEGF.

Answer of exercise 2.3

1.
$$\begin{aligned}
o_0 &: \neg i_0 \\
o_1 &: i_1 \oplus i_0 \\
o_2 &: i_2 \oplus (i_0 \wedge i_1) \\
o_3 &: i_3 \oplus (i_0 \wedge i_1 \wedge i_2) \\
c &: \quad i_0 \wedge i_1 \wedge i_2 \wedge i_3
\end{aligned}$$

2.
$$\begin{aligned}
&a_1 \leftrightarrow (i_0 \wedge i_1)\\
&a_2 \leftrightarrow (a_1 \wedge i_2)\\
&a_3 \leftrightarrow (a_2 \wedge i_3)\\
&o_0 \leftrightarrow \neg i_0\\
&o_1 \leftrightarrow (i_1 \oplus i_0)\\
&o_2 \leftrightarrow (i_2 \oplus a_1)\\
&o_3 \leftrightarrow (i_3 \oplus a_2)\\
&c \leftrightarrow a_3
\end{aligned}$$

Answer of exercise 2.4

1.
$$\begin{aligned}
a \wedge (b \wedge c) &\equiv a \wedge (c \wedge b) \text{ commutativity}\wedge\\
&\equiv (a \wedge c) \wedge b \text{ distributivity } \wedge\\
&\equiv (c \wedge a) \wedge b \text{ commutativity}\wedge
\end{aligned}$$

2.
$$\begin{aligned}
\top \vee (\bot \vee \top) &\equiv \top \vee \top && \text{elimination}\bot \vee \text{ or elimination}\top \vee \text{ (either one)}\\
&\equiv \top && \text{elimination}\top \vee \text{ or idempotence}\wedge
\end{aligned}$$

Answer of exercise 2.5

1. a
2. b
3. $\bot$
4. $a \wedge b$
5. $a \wedge b \wedge c$

Answer of exercise 2.6

1. yes: Take any valuation that assigns *true* to A, B and C.
2. yes: Take any valuation that assigns *true* to A and B.
3. yes: Take any valuation that assigns *true* to B.
4. no: There is no valuation that makes the formula true. Since at least one of A and B is true (due to $A \vee B$), the left-hand side of at least one of the implications $A \rightarrow \neg C$ and $B \rightarrow \neg C$ must be true, and to make these implications true, $\neg C$ must be *true* and hence C must be *false*. But then the last conjunct C requires C to be *true*. Which is a contradiction.

Answer of exercise 2.7

1. This is true. Proof: Since Σ is satisfiable, there is a valuation v such that $v \models \phi$ for every $\phi \in \Sigma$. Clearly, $v \models \phi$ holds for all $\phi \in \Sigma_0$, because $\Sigma_0 \subseteq \Sigma$.
2. This is not true. As a counterexample to the claim, choose $\phi_1 = A$ and $\phi_2 = \neg A$. Clearly, $A \wedge \neg A$ is not satisfiable.
3. This is true. Since ϕ_1 and ϕ_2 are satisfiable, there are valuations v_1 and v_2 such that $v_1 \models \phi_1$ and $v_2 \models \phi_2$. Both $v_1 \models \phi_1 \vee \phi_2$ and $v_2 \models \phi_1 \vee \phi_2$, so to show the satisfiability of $\phi_1 \vee \phi_2$, we could pick either valuation.
4. This is true. The argument is essentially the same: if there is a valuation v such that $v \models \phi_1$, then by the truth-definition of $\vee$ also $v \models \phi_1 \vee \phi_2$. Similarly if there is v such that $v \models \phi_2$.

Answer of exercise 2.8

1. The formula is satisfiable. While the conjunction of satisfiable formulas is in general does not have to be satisfiable, in this case the conjuncts are independent of each other, as they share no atomic propositions. It suffices to find a valuation for each conjunct, and then combine them: $A \vee B$ is satisfied by $A = 1, B = 1$, $\neg C \vee \neg D$ is satisfied by $C = 0, D = 0$, and so on, leading to a valuation that satisfies all conjuncts and hence the whole formula.

2. The formula is satisfiable, because it is a disjunction with at least one satisfiable disjunct (actually, every disjunct is satisfiable). Any valuation that makes at least one of the disjuncts *true* makes the whole formula *true*.

Answer of exercise 2.9

1. No. There is no 4-bit binary number xyz1 that would increment to a binary number of the form uvw1. So the formula is unsatisfiable.
2. Yes. You get a satisfying valuation from any input number of the form xy10, which will be incremented to xy11. One such valuation is the following.

i_0	i_1	i_2	i_3	a_1	a_2	a_3	o_0	o_1	o_2	o_3	c
0	1	0	0	0	0	0	1	1	0	0	0

Another such valuation is the following.

i_0	i_1	i_2	i_3	a_1	a_2	a_3	o_0	o_1	o_2	o_3	c
0	1	1	1	0	0	0	1	1	1	1	0

Answer of exercise 2.10

1. yes
2. yes
3. no
4. yes
5. yes
6. yes

Answer of exercise 2.11

The answer is yes. Proof: We assume that $\alpha \models \beta$, and will now show that $\neg\beta \models \neg\alpha$ holds as well. For $\neg\beta \models \neg\alpha$ to hold, it must be $v \models \neg\alpha$ for any valuation v such that $v \models \neg\beta$. Let v be any valuation such that $v \models \neg\beta$. Hence $v \not\models \beta$. If it was the case that $v \models \alpha$, then by $\alpha \models \beta$ it should be the case that also $v \models \beta$. Hence it cannot be the case that $v \models \alpha$. Hence $v \models \neg\alpha$, and we have confirmed that $\neg\beta \models \neg\alpha$. Q.E.D.

Answer of exercise 2.12

1. no
2. yes
3. yes
4. yes
5. yes

Answer of exercise 2.13

1. This is an unsatisfiable formula. The model-count is **0**.
2. There are 4 valuations, and the formula is *false* in only one. The model-count is **3**.
3. The first disjunct is valid (true in every valuation) and the second disjunct is unsatisfiable. The whole formula is valid. The model-count is $2^4 = \mathbf{16}$ because the formula is true in every valuation of A, B, C, D.
4. The conjuncts of the formula are logically disjoint (not sharing any atomic propositions), and each conjunct is satisfied by 3 valuations over the two atomic propositions. Hence the model-count is $3 \times 3 \times 3 \times 3 = 3^4 = \mathbf{81}$.
5. This is the negation of the preceding formula. It is *false* in 81 valuations. There is a total of $2^8 = 256$ valuations, so this formula is *true* in $256 - 81 = 175$ valuations. The model count is **175**.
6. The formula has only one satisfying valuation, in which all atomic propositions are *true*. So the model-count is **1**.

3 Exercises: State-Space Search

In many applications of state space search methods the number of states is too high for all states to be enumerate explicitly. Instead, state spaces are described implicitly so that a state-space search algorithm such as A* can generate – on demand – all the *successor states* of any given state.

In general, there are two main possibilities, one *procedural* (expressed in terms of program code) and one *declarative*:

1. Implement, as a program, the mapping from a given state to its successor states.
2. Describe the successor generation in terms of *actions* or *transition rules*, consisting of
   - a *pre-condition*, which describes if the action is *applicable* in a given state, and
   - the *effects*, which describe the *changes* to the current state the action causes.

A system's behavior can hence be described by indicating

- what is the starting state of the system (the *initial state*), and
- how the successors of any given state are generated.

A state is *reachable* from the initial state of the system, if there is a *sequence* of n actions $a_1, \dots, a_n$, generating a state sequence $s_0, \dots, s_n$, so that s_0 is the initial state, and each a_i maps s_{i-1} to s_i (for each $i \in \{1, \dots, n\}$).

An example of a system would be for example some puzzle or game, in which the initial state of the system can be changed by the player(s) performing some actions that change the state of the game. The solution of a puzzle like the 15-puzzle is a sequence of actions that reaches the goal state from the initial state. Hence the *reachability problem* is a core problem about systems with a changing state.

Other, more real-world like, applications include different types of validation problems, for example proving that some program or control system does not exhibit faulty behaviors. Think about avionics software used in airplanes, control systems of a nuclear power plant, or a CPU. All of these can be modeled, and are actually modelled as the same type of system models we are using in our toy examples.

Exercise 3.1

Consider two integer-valued state variables x and y and the initial state $(0, 0)$, respectively indicating that $x = 0$ and $y = 0$. How many states are *reachable* from this initial state with the following actions (remember to include the initial state itself, which is always reachable with the empty (length 0) action sequence.

1. There is one action only, with pre-condition: $0 \leq x \leq 9$ and effect: $x := x + 1$.
2. There are two actions:
   - pre-condition $0 \leq x \leq 9$ and effect: $x := x + 1$
   - pre-condition $0 \leq y \leq 9$ and effect: $y := y + 1$
3. There are four actions:
   - pre-condition $0 \leq x \leq 9$ and effect: $x := x + 1$.
   - pre-condition $0 \leq y \leq 9$ and effect: $y := y + 1$.
   - pre-condition $1 \leq x \leq 10$ and effect: $x := x - 1$.
   - pre-condition $1 \leq y \leq 10$ and effect: $y := y - 1$.
4. There is only one action:
   - pre-condition $x \geq 0$ and effect: simultaneously assigning $x := y + 1$ and $y := x$

3.1 A*

Exercise 3.2

Simulate the A* algorithm with the graph in Figure 1, with A as the unique initial state, and G_1, G_2 and G_3 as goal states, and then provide the requested information.

Hint: After expanding a state (determining its successors), calculate the f-values for each of the new states in *OPEN* set. The f-value is the sum of the g-value (the weight/cost of the path through which the state was reached from A) and the h-value (the heuristic lower-bound estimate of the remaining cost to a goal state.) The next state to be expanded is always an unexpanded state (in *OPEN*) with the lowest f-value. Remember that the algorithm does not terminate when first encountering a goal state, but only when there are no states in *OPEN* with an f-value lower than the cost of the best

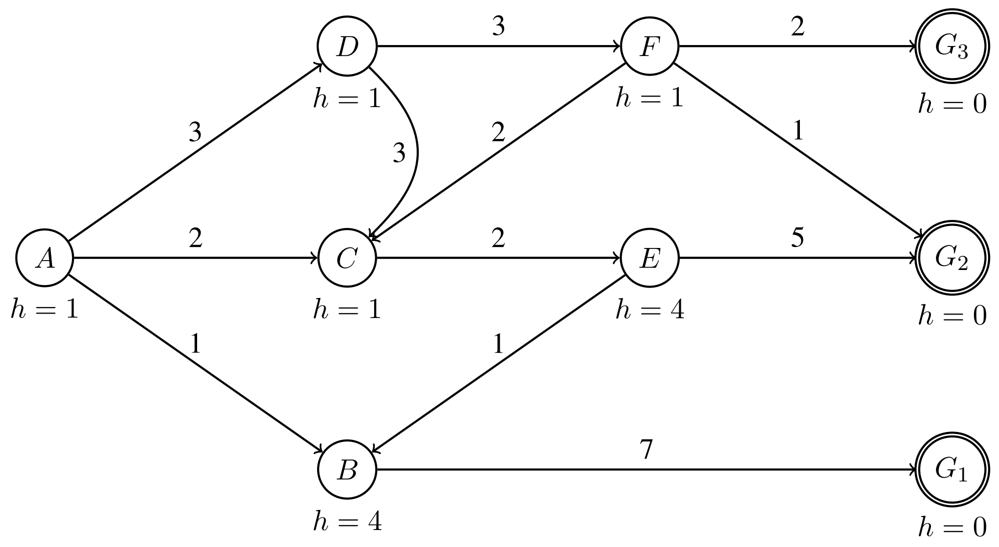


Figure 1: A map annotated with h-values and arc weights

solution path found so far.

1. Determine the f-values for all states in the system.
2. Which states are *not* expanded during the execution of the algorithm?
3. Is h monotonous?

3.2 Lower Bounds / Heuristics

Exercise 3.3

Consider a transportation problem with multiple packages to be delivered from different source locations to different target locations on a road network.

The cost is defined as the sum of the weights/lengths of the edges (road segments) traversed by the delivery vehicles when transporting the packages from their source locations to their target locations. We assume that every vehicle can hold multiple packages, or even all of them at the same time.

Clearly, the straight-line distance is a lower bound on the actual cost of transporting one package. Let $D(l, l')$ be the straight-line distance between locations l and l', and let $s(p)$ and $t(p)$ be the source and target locations of for package $p \in P$, where P is the set of all packages.

Are the following *admissible heuristics* (cost lower bounds) when transporting all packages in P?

1. $\max_{p \in P} D(s(p), t(p))$
2. $\sum_{p \in P} D(s(p), t(p))$ assuming that there is only one vehicle
3. $\sum_{p \in P} D(s(p), t(p))$ assuming that every package is delivered with a different vehicle
4. $\sum_{p \in P} D(s(p), t(p))$ assuming that there is never more than one package inside any vehicle
5. $\max_{p \in P} D(s_v, s(p)) + \max_{p \in P} D(s(p), t(p))$ assuming that there is only one vehicle and its initial location is s_v

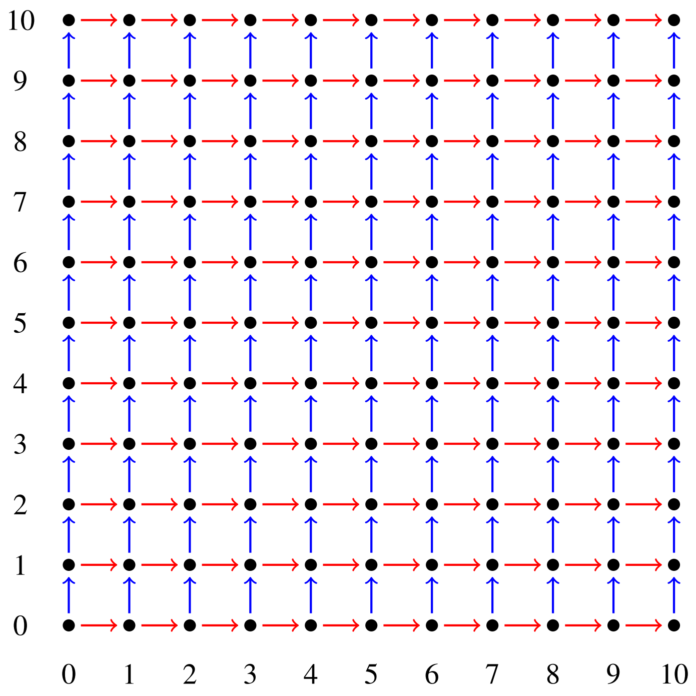
3.3 Solutions

Answer of exercise 3.1

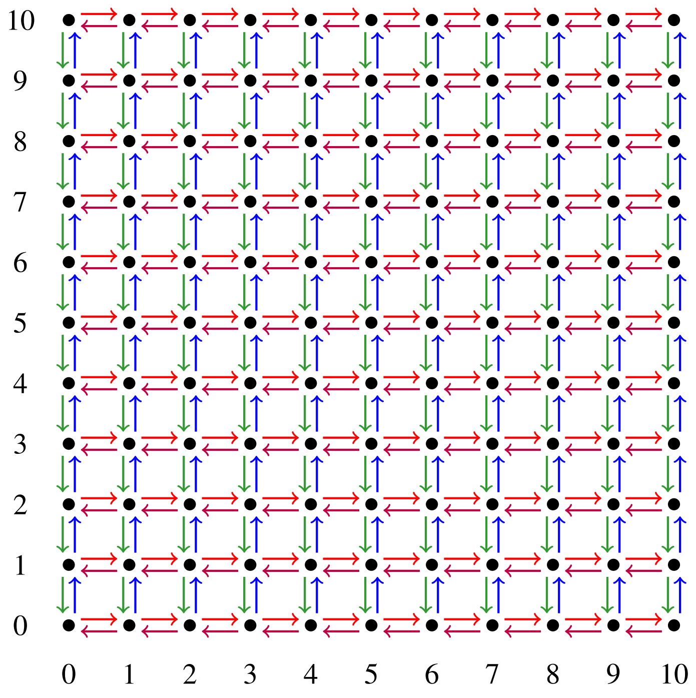
1. The value of y cannot be changed, and the possible values of x are all integers from 0 to 10. Hence the reachable states are $(0, 0), (1, 0), (2, 0), \ldots, (10, 0)$. The number of reachable states is **11**.

$$\begin{tikzcd} 0 & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \arrow[r] & \bullet \\ & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{tikzcd}$$

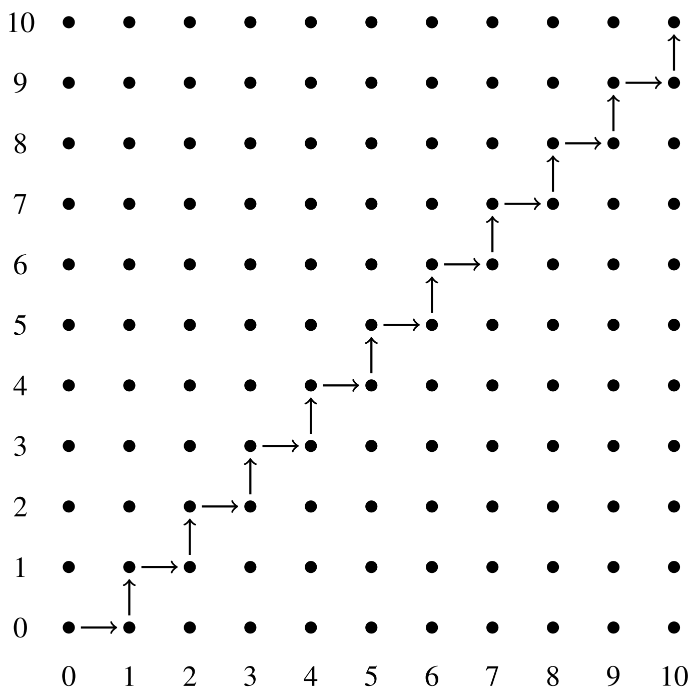
2. Now we can move from $(0,0)$ to any (x,y) with $0 \le x \le 10$ and $0 \le y \le 10$ (but we could not get back to the initial state $(0,0)$, as we cannot decrease the values of x and y. The reachable part of the state space is $\{0,\ldots,10\} \times \{0,\ldots,10\}$, and hence there are $11 \times 11 = \mathbf{121}$ reachable states, induced by all combinations of possible values of x and y. The reachable state space is shown below, with the $x := x+1$ action depicted in red, and the $y := y+1$ action depicted in blue.



3. The set of reachable states is the same, $11 \times 11 = \mathbf{121}$. However, the system's behavior is different as we can move from any reachable state to any other reachable state, because of the decrementing actions.



4. The state sequence generated by the single action is $(0,0),(1,0),(1,1),(2,1),(2,2),(3,2),(3,3),\ldots$, and there are **infinitely many** reachable states. Below we show only a part of the state space.



Answer of exercise 3.2

1. From A to F the values are 1, 5, 3, 4, 8, 7.
2. State E is never expanded.
3. Yes it is.

Answer of exercise 3.3

1. Yes. No matter how many vehicles are used, for every $p \in P$ some vehicle must travel from $s(p)$ to $t(p)$, and $D(s(p), t(p))$ is a lower bound on that cost.
2. No. With one vehicle only, that vehicle has to visit – for every package $p \in P$ – the package's source location and later its target location, which might suggest the sum as a lower bound. However, parts of these trips may be shared, with multiple packages in the hold of the vehicle. In the extreme case, all packages could have the same source and target locations s_{all} and t_{all}. Hence $\sum_{p \in P} D(s(p), t(p))$ could potentially overestimate the total cost by a factor of $|P|$. An admissible bound in this case would be simply $D(s_{\text{all}}, t_{\text{all}})$.
3. Yes. All routes for every $p \in P$ from $s(p)$ to $t(p)$ must be driven separately.
4. Yes. This is exactly the same as the previous.
5. No. The vehicle has to move from s_v to the source locations of every package, so $\max_{p \in P} D(s_v, s(p))$ is an admissible lower bound. But, part of the trip that maximizes $D(s_v, s(p))$ (for some $p \in P$) may overlap the trip that maximizes $D(s(p'), t(p'))$ for some $p' \in P$, so we cannot simply take the sum of these two without risking getting an estimate that is longer than the actual distance. We would obtain an admissible heuristic by replacing either of the two $\max$ by $\min$.

4 Exercises: Heuristics (Lower Bounds) for State-Space Search

Exercise 4.1

Pattern databases with one object moving in an empty 2D grid are can be obtained directly from Manhattan distances.

If the PDB is constructed for two objects (e.g. two tiles in the 8-puzzle), then summing the Manhattan distances for the two objects is not (in general) giving the true cost for moving the two objects.

Give an example that demonstrates this.

Exercise 4.2

The target configuration of the 8-puzzle restricted to the tiles 1, 2 and 3 is the following.

1	2	3
.	.	.
.	.	.

Construct part of the PDB for tiles 1, 2 and 3 by determining the cost of getting from the following configurations to the target configuration.

1.

3	1	2
.	.	.
.	.	.

2.

2	1	.
3	.	.
.	.	.

3.

.	.	.
.	.	.
3	2	1

In which cases does the cost in the PDB coincide with the sum of the Manhattan distances for the tiles?

Exercise 4.3

We construct a PDB heuristic for a problem in which tiles are moving in a grid. The number of tiles can be high, but the PDB only considers a subset of the tiles, thereby *underestimating* the number of moves required to reach a given goal state. However, the number of moves for the subset are considered *exactly*, so the only place where approximation takes place is in ignoring some of the tiles.

We want the tiles to be in the following state in the goal state.

.	3	2
.	4	1
.	.	.

We construct two pattern databases (PDB), one for the subset $\{1, 2\}$ and the other for the subset $\{3, 4\}$. The pattern databases are constructed by determining the distances of *all* states with tiles 1 and 2 to the state

.	.	2
.	.	1
.	.	.

and the distances of all states with tiles 3 and 4 to the state

.	3	.
.	4	.
.	.	.

Notice that these 2-element goal states exactly match the goal state for the 4-tile problem, only ignoring some of the tiles.

1. What cost lower bound is obtained for the following state if we *take the maximum* of the two estimates obtained with the 2-element subsets?

.	.	.
2	1	.
3	4	.

2. What cost lower bound is obtained if we *sum* the two estimates?
3. Which one is correct, summing or taking the maximum?

Exercise 4.4

Construct two PDBs for a tile puzzle of size 3×3 and a state in that tile puzzle so that the *sum of the PDB estimates* for that state is not a lower bound on the actual cost.

Exercise 4.5

Estimate the feasibility of constructing PDBs of different sizes for 8-Puzzle and 15-Puzzle, by considering the number of states for N tiles placed in a grid of $M \times M$ cells for $M = 3$ and $M = 4$.

4.1 Solutions

Answer of exercise 4.1

The simplest example is the distance between the following two states.

1	.	.
2	.	.
.	.	.

2	.	.
1	.	.
.	.	.

According to Manhattan distances, each object could be moved in the right position by only one move, so the cost would be two. However, for each object the other object blocks this direct move. The actual cost is 4: move one object to the side; move the other object to its target location; move the first object to the right row; move the first object to the right column which is now the target location.

Answer of exercise 4.2

1. 6
2. 7
3. 10

In the last case all of the tiles can be moved along a shortest possible path without other tiles blocking its route, so in this case the sum of the Manhattan distances gives the same cost.

Answer of exercise 4.3

1. First determine the distance for tiles 1 and 2 to the respective goal state, moving tiles 1 and 2 from the middle row to the two upper right-hand corner. This is one move right for 1, and one move up and two moves right for 2, which is 4 moves total for 1 and 2.
   Similarly, the distance for 3 and 4 is four moves: move 4 one step up, and move 3 twice up and once right.
   If we take the maximum of the distances, we get 4.
2. Taking the sum of the distances is 4+4=8, which in this case is a far better (= tighter) lower bound estimate for the number of moves required to move all four tiles. Actually, the smallest number of moves is this same 8, so the sum in this case works very well.

3. As has been pointed out in the lecture, the *maximum* of any two lower bounds (admissible heuristics) is still a lower bound, so taking the maximum OK.
   In this puzzle, taking the sum is in general correct, but it is correct when the two PDBs are for two *disjoint* sets of tiles, not sharing any tile.

Answer of exercise 4.4

As pointed out earlier, summing two lower bounds may become too high an estimate for the lowest cost solution when the PDBs share a tile. When a tile is shared, the moves for the shared tile get counted twice, and this may then lead to obtaining estimates that exceed the actual cost.

Here is a very simple example, a tile movement problem with three tiles, and two PDBs that share one tile. The goal state is

3	2	1
.	.	.
.	.	.

and the current state is

.	.	.
3	2	1
.	.	.

Clearly, the optimal solution is just moving each tile one step up.

If we construct PDBs for the subsets $\{1, 2\}$ and $\{2, 3\}$, these PDBs respectively give cost lower bounds 2 and 2, moving two tiles in both cases. This will yield the lower bound 4, which is too high because the moving of tile 2 is counted twice.

Answer of exercise 4.5

The number of states in a PDB for a MAPP problem in grids of size $M \times M$ and N objects is the same as the number of ways N objects can be placed in M^2 locations, which is by basic combinatorics $\frac{(M^2)!}{(M^2-N)!}$.

Similarly, PDBs for the 8-Puzzle and the 15-Puzzle are about placing some N tiles in a grid of size 3×3 or 4×4. We get the following table for the number of entries in a PDB for these two puzzles.

PDB	states with grid size	
tiles	3×3	4×4
1	9	16
2	72	240
3	504	3360
4	3024	43680
5	15120	524160
6	60480	5765760
7	181440	57657600
8	362880	518918400
9	-	4151347200
10	-	29059430400
11	-	174356582400
12	-	871782912000
13	-	3487131648000
14	-	10461394944000
15	-	20922789888000

The number of states in 8-Puzzle is small, and even exhaustive breadth-first with all tiles is feasible with an efficient implementation. (The Python implementations used in our course are far from optimized in terms of speed!)

For the 15-Puzzle it is feasible to use a PDB of maybe up to size 5, 6 or 7, but beyond that the number of states becomes too high, and will lead to excessive memory use and long runtimes. Here adding one tile to the pattern multiplies the PDB size by about 10, so a more feasible way of improving the heuristics would probably be using two PDBs of the same size instead, and aggregate the estimates obtained with them.

It is not clear what size PDB leads to fastest solution with A*: a bigger PDB may take too long time to construct and not speed up the A$*$ search enough to compensate for the long construction time.

Note that these puzzles are *very easy* to solve if it is not required that an optimal (least cost) solution is found. Similarly to Rubik's cube, there are simple schemes that solve the puzzles without search.

5 Exercises: Reasoning in Propositional Logic

5.1 Normal Forms

Exercise 5.1

Transform the following formulas to Negation Normal Form (NNF).

1. $a \rightarrow (b \rightarrow c)$
2. $\neg(a \wedge (\neg b \vee c))$
3. $\neg((a \vee \neg b) \wedge \neg(c \wedge b))$

Exercise 5.2

Transform the following formulas to Conjunctive Normal Form (CNF).

1. $a \wedge (b \vee \neg c \vee d) \wedge \neg b$
2. $a \vee (b \wedge \neg c \wedge \neg e)$
3. $(a \wedge b) \vee (c \wedge d)$

5.2 Simplifications

It is often useful to simplify formulas, to make either manual or automated reasoning with them more efficient. Many of the equivalences listed in the course material can help in this.

Exercise 5.3

Simplify the following formulas as much as possible, that is, find a logically equivalent but simpler formula, with fewer occurences of atomic propositions and/or connectives.

1. $\neg(\neg a \vee b)$
2. $a \wedge (\neg a \vee b \vee c)$
3. $a \rightarrow (a \rightarrow a)$
4. $a \wedge (b \vee \neg a) \wedge (\neg b \vee c)$

5.3 Logical Equivalences

Exercise 5.4

Prove that the following logical equivalences hold by establishing a chain of logically equivalent formulas with the left-most formula as the first one and the rightmost formulas as the last one in the chain.

1. $\neg a \rightarrow (b \vee c) \equiv \neg c \rightarrow (a \vee b)$
2. $a \rightarrow (b \wedge c) \equiv (\neg b \rightarrow \neg a) \wedge (\neg c \rightarrow \neg a)$

5.4 Resolution and Unit Resolution

Exercise 5.5

Which unit clauses (literals) are inferred by unit resolution from the following clause sets?

1. $\{\{a, \neg b\}, \{b, \neg c\}, \{c, \neg a\}\}$
2. $\{\{a\}, \{\neg a, \neg b\}, \{b, \neg c\}, \{c, d\}$
3. $\{\{d, \neg e\}, \{\neg d, \neg e\}, \{e\}\}$

If unit resolution derives a complementary pair of literals (and then the empty clause), for example a and $\neg a$, then the clause is unsatisfiable. But the converse does not hold: there are many unsatisfiable clause sets from which no empty clause can be derived. Hence unit resolution is *incomplete* as an inference rule, and stronger reasoning methods are needed in general.

Unit resolution is part of many complete algorithms for solving the satisfiability problem, including the Davis-Putnam-Logemann-Loveland procedure and the more recent Conflict-Driven Clause Learning (CDCL) algorithm.

Exercise 5.6

Show that the formula $(a \leftrightarrow b) \wedge (a \leftrightarrow \neg b)$ is unsatisfiable by using the resolution rule.

Exercise 5.7

How many different clauses can you derive from the clauses $\{\{a_1, a_2, \ldots, a_{10}\}, \{\neg a_1, b_1\}, \{\neg a_2, b_2\}, \ldots, \{\neg a_{10}, b_{10}\}\}$ by using the resolution rule?

5.5 Logical vs Arithmetic Reasoning

Exercise 5.8

Propositional formulas in the clausal form can be translated into integer inequalities, and solutions of those integer inequalities will be solutions to the satisfiability problem for those formulas.

Consider atomic propositions x_1, x_2, x_3, x_4. These are Boolean variables, with possible values *true* and *false*, or, equivalently, 0 and 1. Let us view these now as *integer* variables, with the constraints $0 \leq x_1 \leq 1$, $0 \leq x_2 \leq 1$, $0 \leq x_3 \leq 1$, and $0 \leq x_4 \leq 1$.

The requirement that the atomic formula x_1 is true can now be represented as the integer inequality $x_1 \geq 1$, which together with $0 \leq x_1 \leq 1$ mean that $x_1 = 1$.

Represent the following propositional formulas as integer inequalities of the form $c_1 \cdot x_1 + c_2 \cdot x_2 + c_3 \cdot x_3 + c_4 \cdot x_4 \geq c$ so that the formula is *true* if and only if the integer inequality holds. Here c_i for $1 \leq i \leq 4$ and c are integer coefficients.

1. $\neg x_2$
2. $x_1 \vee \neg x_2$
3. $x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4$

Any clause set (obtained by turning some propositional formula to CNF) can be translated into 0-1 integer inequalities this way. The clause set is satisfiable if and only if the integer inequalities have a solution. Note that it is critical here that the variables have integer values only: with real-valued variables some of the solutions could be real-valued, and would not correspond to solutions of the satisfiability problem.

The propositional satisfiability problem SAT can be viewed as a special case of 0-1 Integer Programming, which in turn is a special case of the general Integer Programming problem. All these three are NP-hard. The SAT problem has no optimization component like 0-1 IP and IP do, but there is the MAX-SAT problem that combines logical reasoning with an optimization problem. Integer Programming and SAT or MAX-SAT algorithms are often alternative and complementary ways of solving the same combinatorial problems, with different strengths and weaknesses. Note that the Linear Programming problem is solvable in polynomial time, and hence much easier than SAT and IP.

5.6 State-Space Search with Logic

Exercise 5.9

Represent the following system's behavior as a propositional formula. The system consists of 3 lamps, which are initially all turned off. There are three actions, each for turning on one of the lamps. Use the atomic propositions $x_1@t, x_2@t, x_3@t$ for $t \in \{0, 1\}$ to indicate whether a lamp is on or off in the formula which we call $\Phi@0$.

Then give a satisfying valuation for the formula $S = \neg x_1@0 \wedge \neg x_2@0 \wedge \neg x_3@0 \wedge \Phi@0 \wedge \Phi@1 \wedge \Phi@2 \wedge x_1@2 \wedge x_2@2 \wedge x_3@2$, where the formulas $\Phi@1$ and $\Phi@2$ are obtained from $\Phi@0$ by replacing the atoms $x_1@0, x_2@0, x_3@0, x_1@1, x_2@1, x_3@1$ respectively by $x_1@1, x_2@1, x_3@1, x_1@2, x_2@2, x_3@2$ and $x_1@2, x_2@2, x_3@2, x_1@3, x_2@3, x_3@3$.

Exercise 5.10

Why are the conjuncts $\neg x_1@0 \wedge \neg x_2@0 \wedge \neg x_3@0$ needed in the formula

$$S = \neg x_1@0 \wedge \neg x_2@0 \wedge \neg x_3@0 \wedge \Phi@0 \wedge \Phi@1 \wedge \Phi@2 \wedge x_1@2 \wedge x_2@2 \wedge x_3@2?$$

Aren't the lamps off in the beginning anyway? Explain.

Exercise 5.11

Also, is it necessary to say explicitly by the equivalences $x_i@(t+1) \leftrightarrow x_i@t$ that a lamp's state does not change when some other lamp is turned on? Isn't this so anyway? It seems we could use a much simpler formula $\Phi'@0 = x_1@1 \vee x_2@1 \vee x_3@1$ to describe the system's behavior. Explain.

Exercise 5.12

Give a formula that maps any bit-vector $b_0b_1b_2b_4$ to its mirror image. Use the atomic propositions $b_0@0, b_1@0, b_2@0, b_4@0$ to indicate the components of the input bit-vector and $b_0@1, b_1@1, b_2@1, b_4@1$ for the output bit-vector.

Exercise 5.13

Consider the state variables x_0, x_1, x_2 and the following formula

$$\begin{pmatrix} (x_0@0 \vee x_1@0) \leftrightarrow x_0@1) \\ \wedge(x_0@0 \wedge x_1@0) \leftrightarrow x_1@1) \\ \wedge(x_2@0 \leftrightarrow x_2@1) \end{pmatrix} \bigvee \begin{pmatrix} (x_0@0 \leftrightarrow x_0@1) \\ \wedge(x_1@0 \vee x_2@0) \leftrightarrow x_1@1) \\ \wedge(x_1@0 \wedge x_2@0) \leftrightarrow x_2@1) \end{pmatrix}$$

that represents the possible transitions from a state to its successor.

Is the state $x_0 = 1, x_1 = 1, x_2 = 0$ reachable from the state $x_0 = 0, x_1 = 1, x_2 = 1$ by two steps?

Note: Unlike in some other exercises here, by 110 we denote the valuation $x_0 = 1, x_1 = 1, x_2 = 0$ and not $x_0 = 0, x_1 = 1, x_2 = 1$, which should not be too confusing as we do not view the bits $x_0x_1x_2$ as representing an integer.

Exercise 5.14

Consider the following valuation that was returned by a SAT solver for a scheduling problem that has the tasks in three independent jobs ordered as $A < B < G$ and $C < D < H$ and $E < F < J < K$.

	t					
atom	0	1	2	3	4	5
$A@t$	1	0	0	0	0	0
$B@t$	0	0	1	0	0	0
$C@t$	1	0	0	0	0	0
$D@t$	0	0	0	1	0	0
$E@t$	0	1	0	0	0	0
$F@t$	0	0	1	1	0	0
$G@t$	0	1	0	0	0	0
$H@t$	0	0	0	0	1	0
$J@t$	0	0	0	0	1	0
$K@t$	0	0	0	0	0	0

The tasks in the set $\{A, B, C, D, E\}$ use resource 1, and the remaining tasks use resource 2. Tasks that use the same resource cannot be simultaneous.

The software engineer who implemented the scheduling system has some small typos in the implementation, and some of the constraints on the atomic propositions representing the tasks are missing or incorrect.

1. Draw a table in which the schedule for the three jobs respectively consisting of the tasks A, B, G and C, D, H and E, G, J is more understandable.
2. List all issues with the schedule that make it incorrect.
3. Which formulas seem to be missing? These formulas are the ones that prevent the issues you have identified.

5.7 Solutions

Answer of exercise 5.1

1. $a \rightarrow (b \rightarrow c) \equiv \neg a \vee (\neg b \vee c) \equiv \neg a \vee \neg b \vee c$
2. $\neg(a \wedge (\neg b \vee c)) \equiv \neg a \vee \neg(\neg b \vee c) \equiv \neg a \vee (\neg\neg b \wedge \neg c) \equiv \neg a \vee (b \wedge \neg c)$
3. $\neg((a \vee \neg b) \wedge \neg(c \wedge b)) \equiv \neg(a \vee \neg b) \vee \neg\neg(c \wedge b) \equiv (\neg a \wedge \neg\neg b) \vee (c \wedge b) \equiv (\neg a \wedge b) \vee (c \wedge b)$

Answer of exercise 5.2

1. The formula is already in CNF. No need to do anything.
2. $(a \vee b) \wedge (a \vee \neg c) \wedge (a \vee \neg e)$ by two applications of the distributivity rule $\phi_1 \vee (\phi_2 \wedge \phi_3) \equiv (\phi_1 \vee \phi_2) \wedge (\phi_1 \vee \phi_3)$, first obtaining $(a \vee b) \wedge (a \vee (\neg c \wedge \neg e))$, and then applying the same rule to the rightmost disjunction.
3. $$\begin{array}{rll} (a \wedge b) \vee (c \wedge d) & \equiv ((a \wedge b) \vee c) \wedge ((a \wedge b) \vee d) & \text{distributivity} \\ & \equiv (c \vee (a \wedge b)) \wedge (d \vee (a \wedge b)) & \text{commutativity of } \vee \\ & \equiv ((c \vee a) \wedge (c \vee b)) \wedge ((d \vee a) \wedge (d \vee b)) & \text{distributivity } \vee \wedge \\ & \equiv (c \vee a) \wedge (c \vee b) \wedge (d \vee a) \wedge (d \vee b) & \text{removal of parentheses} \end{array}$$

   Notice that the same could have been obtained more directly simply by picking every possible combination of literals from $\{a, b\}$ and from $\{c, d\}$ and forming their disjunction, and then forming a long conjunction from those disjunctions. This also holds more generally, with $(\phi_1 \wedge \phi_1') \vee (\phi_2 \wedge \phi_2') \vee \cdots \vee (\phi_n \wedge \phi_n')$ resulting in a conjunction of 2^n disjunctions that correspond to elements of $\{\phi_1, \phi_1'\} \times \cdots \times \{\phi_n, \phi_n'\}$. This generalizes directly to conjunctions of any length, not only those with just two conjuncts.

Answer of exercise 5.3

1. From $\neg(\neg a \vee b)$ one gets $\neg\neg a \wedge \neg b$ by one of the *De Morgan* rules ($\neg(\phi_1 \vee \phi_2) \equiv \neg\phi_1 \wedge \neg\phi_2$), and then $a \wedge \neg b$ by eliminating the double negation with the equivalence $\neg\neg\phi \equiv \phi$.
2. $a \wedge (b \vee c)$, obtained by distributivity $\wedge\vee$, followed by replacing $a \vee \neg a$ by $\perp$, and then by replacing $\perp \vee (b \vee c)$ by $b \vee c$. One can also think of this as follows: for the formula to be true, a must be true, and if a is true, then $\neg a$ is false, and hence the second conjunct is $\perp \vee b \vee c$, which can be simplified to $b \vee c$. The reasoning here is similar to what is obtained with the (unit) resolution rule.
3. $\top$, obtained by observing that $a \rightarrow a \equiv \neg a \vee a \equiv \top$ and $a \rightarrow \top \equiv \top$
4. $a \wedge b \wedge c$, obtained by similar reasoning as in (2) above.

Answer of exercise 5.4

1. $$\begin{array}{rll} \neg a \rightarrow (b \vee c) & \equiv \neg\neg a \vee (b \vee c) & \text{Definition of implication} \\ & \equiv a \vee (b \vee c) & \text{Double negation} \\ & \equiv (a \vee b) \vee c & \text{Associativity of } \vee \\ & \equiv c \vee (a \vee b) & \text{Commutatitivy of } \vee \\ & \equiv \neg\neg c \vee (a \vee b) & \text{Double negation} \\ & \equiv \neg c \rightarrow (a \vee b) & \text{Definition of implication} \end{array}$$
2. $$\begin{array}{rll} a \rightarrow (b \wedge c) & \equiv \neg a \vee (b \wedge c) & \text{Definition of implication} \\ & \equiv (\neg a \vee b) \wedge (\neg a \vee c) & \text{Distributivity} \\ & \equiv (b \vee \neg a) \wedge (c \vee \neg a) & \text{Commutativity of } \vee \\ & \equiv (\neg\neg b \vee \neg a) \wedge (\neg\neg c \vee \neg a) & \text{Double negation} \\ & \equiv (\neg b \rightarrow \neg a) \wedge (\neg c \rightarrow \neg a) & \text{Definition of implication} \end{array}$$

Answer of exercise 5.5

1. There are no unit clauses, so the unit resolution rule is not applicable.
2. The literals $\neg b$, $\neg c$ and d are inferred.
3. The literals d, $\neg d$ and $\neg e$ are inferred. Since we now have the unit clauses e and $\neg e$, we can also infer the empty clause $\emptyset$, indicating that the clause set is unsatisfiable.

Answer of exercise 5.6

1. First transform the formula to CNF:

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

2. Then write it in clause form as $\{\{\neg a, b\}, \{\neg b, a\}, \{\neg a, \neg b\}, \{b, a\}\}$ (not obligatory, of course).
3. Finally, apply the resolution rule until you derive the empty clause.

$$\begin{tikzcd}
\{\neg a, b\} \arrow[dr] & \{\neg b, a\} \arrow[dr] & \{\neg a, \neg b\} \arrow[dl] & \{b, a\} \arrow[dl] \\
 & \{\neg a\} \arrow[d] & \{a\} \arrow[dl] & \\
 & \emptyset & &
\end{tikzcd}$$

Answer of exercise 5.7

Every possible resolution step is between a 10-literal clause consisting of positive literals only (no negation $\neg$ symbols) and one of the 2-literal clauses $\neg a_i \vee b_i$, leading to a new clause that replaces the literal a_i in the 10-literal clause by b_i. There are $2^{10} = 1024$ clauses, each with exactly 10 literals, that can be obtained from the original 10-literal clause. No other clauses can be derived.

Answer of exercise 5.8

1. $-1 \cdot x_2 \geq 0$ (which can be obtained from $(1 - x_2) \geq 1$ when viewing negation as subtraction from 1)
2. $x_1 + (-1) \cdot x_2 \geq 0$ (obtained from $x_1 + (1 - x_2) \geq 1$)
3. $x_1 + (-1 \cdot x_2) + (-1 \cdot x_3) + x_4 \geq -1$ (obtained from $x_1 + (1 - x_2) + (1 - x_3) + x_4 \geq 1$)

Answer of exercise 5.9

Turning the first lamp on (represented by x_1) is represented by the formula

$$x_1@1 \wedge (x_2@1 \leftrightarrow x_2@0) \wedge (x_3@1 \leftrightarrow x_3@0)$$

indicating that the first lamp is on after the action has been taken, and the state of the lamps 2 and 3 remains unchanged.

Turning on lamps 2 and 3 is analogous.

The choice between the three possible action is by disjunction: either turn on lamp 1, turn on lamp 2, or turn on lamp 3. Hence the formula $\Phi@0$ is as follows:

$$\begin{aligned}\Phi@0 = {} & (x_1@1 \wedge (x_2@1 \leftrightarrow x_2@0) \wedge (x_3@1 \leftrightarrow x_3@0))\\ & \vee (x_2@1 \wedge (x_1@1 \leftrightarrow x_1@0) \wedge (x_3@1 \leftrightarrow x_3@0))\\ & \vee (x_3@1 \wedge (x_1@1 \leftrightarrow x_1@0) \wedge (x_2@1 \leftrightarrow x_2@0))\end{aligned}$$

A valuation that satisfies the formula S is

t	$x_1@t$	$x_2@t$	$x_3@t$
0	0	0	0
1	0	1	0
2	0	1	1
3	1	1	1

This is just one of the $3! = 6$ satisfying valuations for the formula S: the lamps could be turned on just as well in any other order.

Answer of exercise 5.10

If these conjuncts are not included, the formula has satisfying valuations that do not correspond to the scenario as described. In particular, there are several valuations that correspond to situations in which one or more lamps are not initially turned off. You can easily verify that e.g. the following valuation satisfies S if the $\neg x_i@0$ conjuncts are left out.

t	$x_1@t$	$x_2@t$	$x_3@t$
0	0	1	1
1	1	1	1
2	1	1	1
3	1	1	1

Answer of exercise 5.11

This is similar to the previous exercise. If the part of the formula about turning on lamp 1 does not include $(x_2@1 \leftrightarrow x_2@0) \wedge (x_3@1 \leftrightarrow x_3@0)$, then there will be satisfying valuations of the formula in which lamps 2 and 3 do *not* remain unchanged when lamp 1 is turned on.

For example, it would be possible to get all three lamps on just by turning on one of them, with the other two magically turning on simultaneously. The following would be a satisfying valuation for $\neg x_1@0 \wedge \neg x_2@0 \wedge \neg x_3@0 \wedge \Phi'@0 \wedge x_1@1 \wedge x_2@1 \wedge x_3@1$, with all lamps turning on by taking only one of the actions.

t	$x_1@t$	$x_2@t$	$x_3@t$
0	0	0	0
1	1	1	1

Answer of exercise 5.12

So we have to map 0000 to 0000, 0001 to 1000, 0010 to 0100, and so on. We express these bit-vectors in terms of state variables b_0, b_1, b_2, b_3, and to talk about change, we have the atomic propositions $b_0@0, b_1@0, b_2@0, b_3@0$ to refer to the *current* (old) values of the state variables, and $b_0@1, b_1@1, b_2@1, b_3@1$ to refer to their *next* (new) values.

Our formula simply copies the value of b_3 to b_0, b_2 to b_1, b_1 to b_2, and b_0 to b_3, simultaneously.

This is easy to express with equivalences between each state variable's new values and another state variable's old value:

$$\Phi = (b_0@1 \leftrightarrow b_3@0) \wedge (b_1@1 \leftrightarrow b_2@0) \wedge (b_2@1 \leftrightarrow b_1@0) \wedge (b_3@1 \leftrightarrow b_0@0)$$

The truth-table of this formula looks as follows:

$b0_{@}0$	$b_1@0$	$b_2@0$	$b_3@0$	$b_0@1$	$b_1@1$	$b_2@1$	$b_3@1$	Φ
0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	0	1
0	0	1	0	0	1	0	0	1
0	0	1	1	1	1	0	0	1
0	1	0	0	0	0	1	0	1
0	1	0	1	1	0	1	0	1
0	1	1	0	0	1	1	0	1
0	1	1	1	1	1	1	0	1
1	0	0	0	0	0	0	1	1
1	0	0	1	1	0	0	1	1
1	0	1	0	0	1	0	1	1
1	0	1	1	1	1	0	1	1
1	1	0	0	0	0	1	1	1
1	1	0	1	1	0	1	1	1
1	1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1

Rows that are not listed explicitly have 0 in the last column.

Answer of exercise 5.13

We first see what the transition formula means. First notice that the formula consists of two disjuncts, and each of them *uniquely* determines the successor state for every state. This is because both disjuncts are a conjunction of equivalences, and there is an equivalence for every state variable with the @1 tag, and the other side makes reference to only something with the @0 tag (in other words: the new value of every state variable is a (Boolean) function of the old values). So we can look at what the successor state is for each state, w.r.t. each disjunct.

The first disjunct corresponds to the following. (This is the rows of the truth-table on which the first disjunct evaluates to *true*.)

$x_0@0$	$x_1@0$	$x_2@0$	$x_0@1$	$x_1@1$	$x_2@1$
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

The second disjunct corresponds to the following.

$x_0@0$	$x_1@0$	$x_2@0$	$x_0@1$	$x_1@1$	$x_2@1$
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Now we can see that the first part *orders the values of x_0 and x_1 to descending order*, changing $x_0 = 0, x_1 = 1$ to $x_0 = 1, x_1 = 0$ if possible. The second part similarly orders x_1, x_2.

At this point we can probably see the solution: ordering the first two turns 011 to 101, and after that ordering the last two turns 101 to 110.

So, yes, the target state is reachable in two steps.

Below we go through the same thing more formally.

We can form the truth-table for the whole formula, and this is – in terms of relations – simply the union of the above two relations.

$x_0@0$	$x_1@0$	$x_2@0$	$x_0@1$	$x_1@1$	$x_2@1$
0	0	0	0	0	0
0	0	1	0	0	1
0	0	1	0	1	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	1	0	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

The whole formula then means: order x_0, x_1, or, order x_1, x_2. (Of course, ordering something that is already ordered is always allowed, so it is not necessary to change anything.)

Next we want to look at two consecutive attempts to do these orderings. This is by making another copy of the original formula, and replacing $x_i@0$ and $x_i@1$ respectively by $x_i@1$ and $x_i@2$, and conjoining it with the original formula.

Its truth-table has the following rows with *true* in the last column, obtained by a relational join of the previous table with itself (once the columns have been similarly renamed.)

x_0@0	x_1@0	x_2@0	x_0@1	x_1@1	x_2@1	x_0@2	x_1@2	x_2@2
0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	1	0
0	0	1	0	1	0	1	0	0
0	0	1	0	1	0	0	1	0
0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	1	0	0
0	1	0	1	0	0	1	0	0
0	1	1	0	1	1	0	1	1
0	1	1	0	1	1	1	0	1
0	1	1	1	0	1	1	0	1
0	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**
1	0	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0	1
1	0	1	1	0	1	1	1	0
1	1	0	1	1	0	1	1	0
1	1	1	1	1	1	1	1	1

The row corresponding to the two transitions that take 011 to 110 through 101 is highlighted in the table.

Answer of exercise 5.14

1.

	time					
job	0	1	2	3	4	5
1	A	G	B			
2	C			D	H	
3		E	F	F	J	

2. (a) Ordering of $B < G$ in the first job is incorrect.
   (b) Tasks A and C use the same resource at the same time.
   (c) Task F occurs twice in the schedule.
   (d) Tasks H and J use the same resource at the same time.
   (e) Task K is missing in the schedule.
3. (a) The formula $\neg(B@2 \wedge G@1)$ is missing and should be added.
   (b) The formula $\neg(A@0 \wedge C@0)$ is missing and should be added.
   (c) The formula $\neg(F@2 \wedge F@3)$ is missing and should be added.
   (d) The formula $\neg(H@4 \wedge J@4)$ is missing and should be added.
   (e) The formula $K@0 \vee K@1 \vee K@2 \vee K@3 \vee K@4 \vee K@5$ is missing and should be added.

   Of course, each of these missing formulas is a symptom of some more general problem, so the fix to each of these issues will most likely add many other formulas as well.

6 Exercises: Predicate Logic

6.1 Model Theory

Exercise 6.1

For each of the following requirements, construct a structure $\mathcal{S}$ that fulfills it. In each case the structure has to have exactly those predicate, constant and function symbols that occur in the formula.[1]

1. $\mathcal{S} \models P(a)$
2. $\mathcal{S} \models \exists x.P(x)$
3. $\mathcal{S} \models \forall x.P(x)$
4. $\mathcal{S} \models \forall x.P(a, x)$
5. $\mathcal{S} \models (\exists x.P(x)) \wedge (\exists x.\neg P(x))$
6. $\mathcal{S} \models a = b$
7. $\mathcal{S} \models \neg(a = b)$

Exercise 6.2

For each of the following requirements, argue why there *cannot* be a structure $\mathcal{S}$ that fulfills it.

1. $\mathcal{S} \models P(a) \wedge \neg P(a)$
2. $\mathcal{S} \models P(a) \wedge (P(a) \rightarrow P(b)) \wedge \neg P(b)$
3. $\mathcal{S} \models (\forall x.P(x)) \wedge (\exists x.\neg P(x))$
4. $\mathcal{S} \models (\forall x.P(x)) \wedge (\forall x.\neg P(x))$
5. $\mathcal{S} \models P(a) \wedge \neg P(b) \wedge (a = b)$

Exercise 6.3

Argue why the following logical consequences hold.

1. $\forall x.P(x) \models P(a)$
2. $\{\forall x.(P(x) \rightarrow Q(x)), P(a)\} \models Q(a)$
3. $\{P(a), (a = b)\} \models P(b)$
4. $\{P(a), \neg P(b)\} \models \neg(a = b)$

Exercise 6.4

Construct a structure that shows that the formulas $\forall x.\exists y.P(x, y)$ and $\exists y.\forall x.P(x, y)$ are not equivalent.

6.2 Knowledge Representation

Exercise 6.5

Formalize the following sentences in the predicate logic.

1. Every student knows somebody.
2. Somebody knows every student.
3. If somebody knows every student, then every student knows somebody.
4. All students are clever.
5. Some students are clever.

Exercise 6.6

Explain why:

1. $\forall x \exists y(\mathit{student}(x) \wedge \mathit{likes}(x, y) \rightarrow \mathit{happy}(x))$ doesn't correctly formalize "Every student who likes somebody is happy."

[1]None of the formulas have free variables, so constant symbols can be distinguished from variables because the latter are bound by a quantifier and the former are not.

2. Not both of $\exists x(mammal(x) \to marineAnimal(x))$ and $\exists x(mammal(x) \wedge marineAnimal(x))$ are correct formalizations "Some mammals are marine animals."

Exercise 6.7

Translate the following predicate logic formulas to English.

1. $\forall x.\exists y.hasMother(x, y)$
2. $\exists y.\forall x.hasMother(x, y)$
3. $\forall x.(student(x) \to clever(x))$
4. $\neg\exists x.(student(x) \wedge \neg clever(x))$

6.3 Solutions

Answer of exercise 6.1

There are (infinitely) many correct answers. We give one of the simplest and smallest in each case.

1. $\mathcal{S} \models P(a)$: The universe of $\mathcal{S}$ is $U = \{1\}$, $[\![a]\!]^{\mathcal{S}} = 1$ and $[\![P]\!]^{\mathcal{S}} = \{1\}$. It is enough that the interpretation of P includes the element that is the interpretation of a.
2. $\mathcal{S} \models \exists x.P(x)$: The universe of $\mathcal{S}$ is $U = \{1\}$ and $[\![P]\!]^{\mathcal{S}} = \{1\}$. It is enough that the interpretation of P is non-empty.
3. $\mathcal{S} \models \forall x.P(x)$: The universe of $\mathcal{S}$ is $U = \{1\}$ and $[\![P]\!]^{\mathcal{S}} = \{1\}$. It is enough that the interpretation of P equals U, so that P holds for all elements of the universe.
4. $\mathcal{S} \models \forall x.P(a, x)$: The universe of $\mathcal{S}$ is $U = \{1, 2\}$, $[\![a]\!]^{\mathcal{P}} = 1$ and $[\![P]\!]^{\mathcal{S}} = \{(1, 1), (1, 2)\}$. There has to be a pair $(1, x)$ for every $x \in U$. A still simpler solution would be with $U = \{1\}$ and $[\![P]\!]^{\mathcal{S}} = \{(1, 1)\}$
5. $\mathcal{S} \models (\exists x.P(x)) \wedge (\exists x.\neg P(x))$: The universe of $\mathcal{S}$ is $U = \{1, 2\}$ and $[\![P]\!]^{\mathcal{S}} = \{1\}$. Here we need at least two elements so that $P(x)$ can be true for one and $\neg P(x)$ can be true for the other. So the interpretation of P has to be non-empty and cannot include all elements of the universe.
6. $\mathcal{S} \models a = b$: We could pick any universe U (universe is always non-empty), pick any element $x \in U$, and have $[\![a]\!]^{\mathcal{S}} = [\![b]\!]^{\mathcal{S}} = x$.
7. $\mathcal{S} \models \neg(a = b)$: Now we need at least two elements in the universe so that a and b can have different interpretations i.e. $[\![a]\!]^{\mathcal{S}} \neq [\![a]\!]^{\mathcal{S}}$, for example $U = \{1, 2\}$ and $[\![a]\!]^{\mathcal{S}} = 1$ and $[\![b]\!]^{\mathcal{S}} = 2$.

Answer of exercise 6.2

1. $\mathcal{S} \models P(a) \wedge \neg P(a)$: No matter what the interpretation of a is, not both $P(a)$ and $\neg P(a)$ can be true. If the first is true then the second if false, and if the first is false then the second is true. This formula is essentially a formula in the propositional logic because there are no quantifiers: we can pick the truth-values for all atomic formulas independently (and in this case there is only one atomic formula $P(a)$).
2. $\mathcal{S} \models P(a) \wedge (P(a) \to P(b)) \wedge \neg P(b)$: This, too, is essentially a propositional formula. The formula cannot be true, because $P(a)$ would have to be true (by definition of the truth of conjunctions), and hence because of $P(a) \to P(b)$ also $P(b)$ would have to be true. But this contradicts with the requirement that also the last conjunct has to be true. So the formula is unsatisfiable as it cannot be made true.
3. $\mathcal{S} \models (\forall x.P(x)) \wedge (\exists x.\neg P(x))$: The first conjunct requires that the universe U equals $[\![P]\!]^{\mathcal{S}}$, and the second conjunct requires that there is at least one $x \in U$ so that $x \notin [\![P]\!]^{\mathcal{S}}$. These requirements are contradictory, so no such $\mathcal{S}$ exists.
4. $\mathcal{S} \models (\forall x.P(x)) \wedge (\forall x.\neg P(x))$: The first conjunct requires that the universe U equals $[\![P]\!]^{\mathcal{S}}$, and the second conjunct requires that $[\![P]\!]^{\mathcal{S}}$ equals the empty set. This is contradictory, because the universe must always be non-empty. Hence no such $\mathcal{S}$ exists.
5. $\mathcal{S} \models P(a) \wedge \neg P(b) \wedge (a = b)$: If it was the case that a and b refer to the same element as required by the last conjunct, then the truth-values of $P(a)$ and $P(b)$ would necessarily be the same. This contradicts the requirement that $P(a)$ should be true and $P(b)$ should be false.

Answer of exercise 6.3

Argue why the following logical consequences hold.

1. $\forall x.P(x) \models P(a)$: Assume that for a given structure $\mathcal{S}$ we have $\mathcal{S} \models \forall x.P(x)$. This means that $[\![P]\!]^{\mathcal{S}} = U$ for the universe U of $\mathcal{S}$. Since $[\![a]\!]^{\mathcal{S}} \in U$, it must also be that $\mathcal{S} \models P(a)$. Since this holds for any $\mathcal{S}$ in which $\forall x.P(x)$ is true, $P(a)$ is a logical consequence of $\forall x.P(x)$.
2. $\{\forall x.(P(x) \rightarrow Q(x)), P(a)\} \models Q(a)$: Assume that $\mathcal{S}$ is a structure such that $\mathcal{S} \models (\forall x.(P(x) \rightarrow Q(x))) \wedge P(a)$. Since $\mathcal{S} \models P(a)$, we have $[\![a]\!]^{\mathcal{S}} \in [\![P]\!]^{\mathcal{S}}$. Since $\mathcal{S} \models \forall x.(P(x) \rightarrow Q(x))$, it must be that $[\![a]\!]^{\mathcal{S}} \in [\![Q]\!]^{\mathcal{S}}$. Hence $\mathcal{S} \models Q(a)$.
3. $\{P(a), (a = b)\} \models P(b)$: Assume $\mathcal{S}$ is any structure such that $\mathcal{S} \models P(a) \wedge (a = b)$. Hence $[\![a]\!]^{\mathcal{S}} \in [\![P]\!]^{\mathcal{S}}$ and $[\![a]\!]^{\mathcal{S}} = [\![b]\!]^{\mathcal{S}}$. A direct consequence of this is $[\![b]\!]^{\mathcal{S}} \in [\![P]\!]^{\mathcal{S}}$. Hence $\mathcal{S} \models P(b)$.
4. $\{P(a), \neg P(b)\} \models \neg(a = b)$: Take any structure $\mathcal{S}$ for which $\mathcal{S} \models P(a)$ and $\mathcal{S} \models P(b)$. Hence $[\![a]\!]^{\mathcal{S}} \in [\![P]\!]^{\mathcal{S}}$ and $[\![a]\!]^{\mathcal{S}} \notin [\![P]\!]^{\mathcal{S}}$. Hence necessarily $[\![a]\!]^{\mathcal{S}} \neq [\![b]\!]^{\mathcal{S}}$. This means that it must be that $\mathcal{S} \models \neg(a = b)$.

Answer of exercise 6.4

The non-equivalence of $\forall x.\exists y.P(x, y)$ and $\exists y.\forall x.P(x, y)$ is the same as the standard example "Everybody has somebody who is that person's mother" and "Somebody is everybody's mother". One structure for demonstrating this difference is the following.

$$U = \{1, 2, 3\}$$
$$[\![P]\!]^{\mathcal{S}} = \{(1, 2), (2, 3), (3, 1)\}$$

Now, for every $x \in U$, there is some $y \in U$ such that $(x, y) \in [\![P]\!]^{\mathcal{S}}$. In all cases it is a different y. So the first formula $\forall x.\exists y.P(x, y)$ is true in $\mathcal{S}$. But, $\exists y.\forall x.P(x, y)$ is false in $\mathcal{S}$, as there is no one single y so that $(x, y) \in [\![P]\!]^{\mathcal{S}}$ for every $x \in U$. Hence the formulas are not equivalent.

Answer of exercise 6.5

1. $\forall x.(\mathit{student}(x) \rightarrow \exists y.\mathit{knows}(x, y))$
2. $\exists x \forall y.(\mathit{student}(y) \rightarrow \mathit{knows}(x, y))$
3. $(\exists x \forall y(\mathit{student}(y) \rightarrow \mathit{knows}(x, y))) \rightarrow \forall x(\mathit{student}(x) \rightarrow \exists y.\mathit{knows}(x, y))$
   Note that there are no variables shared by the two sides of the implication, and that the outermost connective is the implication connective. The "somebody" does not have to be one person on both sides of the implication, and, actually, on the right-hand side, each student could know a different "somebody".
4. $\forall x.(\mathit{student}(x) \rightarrow \mathit{clever}(x))$
   "All students are clever" is the same as "everybody is either clever or not a student", which leads to the logically equivalent formula $\forall x.(\neg \mathit{student}(x) \vee \mathit{clever}(x))$
5. $\exists x.(\mathit{student}(x) \wedge \mathit{clever}(x))$
   Notice the difference to the previous case. This is perhaps easiest understood by thinking what these sentences say in terms of sets. "All students are clever" says that the set of students is a subset of the set of clever people. "Some students are clever" says that the set of students intersects the set of clever people.

Answer of exercise 6.6

1. Probably the simplest way to understand what the formula says is to think of it saying that "For all x there is at least one y so that the implication $\mathit{student}(x) \wedge \mathit{likes}(x, y) \rightarrow \mathit{happy}(x)$ holds".
   This implication trivially holds if $\mathit{likes}(x, y)$ is false. So if for every student x there is at least one y that x does not like, the formula holds. The student x would have to like *everybody* (all y) before the formula forces x to be happy. This is not at all what is claimed by the sentence "Every student who likes somebody is happy". Somehow the "likes somebody" has turned to "likes everybody". Why?
   We can analyze this question more formally by rewriting it as follows.

$$\begin{aligned}
\forall x \exists y(\mathit{student}(x) \wedge \mathit{likes}(x, y) \rightarrow \mathit{happy}(x)) &\equiv \forall x \exists y(\neg(\mathit{student}(x) \wedge \mathit{likes}(x, y)) \vee \mathit{happy}(x)) \\
&\equiv \forall x \exists y(\neg \mathit{student}(x) \vee \neg \mathit{likes}(x, y) \vee \mathit{happy}(x)) \\
&\equiv \forall x(\neg \mathit{student}(x) \vee (\exists y.\neg \mathit{likes}(x, y)) \vee \mathit{happy}(x)) \\
&\equiv \forall x(\neg(\neg \mathit{student}(x) \vee (\exists y.\neg \mathit{likes}(x, y))) \rightarrow \mathit{happy}(x)) \\
&\equiv \forall x(\mathit{student}(x) \wedge \neg(\exists y.\neg \mathit{likes}(x, y)) \rightarrow \mathit{happy}(x)) \\
&\equiv \forall x(\mathit{student}(x) \wedge (\forall y.\mathit{likes}(x, y)) \rightarrow \mathit{happy}(x))
\end{aligned}$$

which means that "every student who likes everybody is happy". This is quite a bit different from "every student who likes somebody is happy."

Another way to rewrite the formula is as follows.

$$\begin{aligned}\forall x\exists y(\mathit{student}(x) \wedge \mathit{likes}(x,y) \rightarrow \mathit{happy}(x)) &\equiv \forall x\exists y(\neg(\mathit{student}(x) \wedge \mathit{likes}(x,y)) \vee \mathit{happy}(x))\\ &\equiv \forall x\exists y(\neg\mathit{student}(x) \vee \neg\mathit{likes}(x,y) \vee \mathit{happy}(x))\\ &\equiv \forall x(\neg\mathit{student}(x) \vee (\exists y.\neg\mathit{likes}(x,y)) \vee \mathit{happy}(x))\\ &\equiv \forall x(\mathit{student}(x) \rightarrow ((\exists y.\neg\mathit{likes}(x,y)) \vee \mathit{happy}(x)))\end{aligned}$$

This is "Every student either dislikes somebody or is happy", which of course means the same thing as "Every student who likes everybody is happy."

The problematic thing in formalizing "Every student who likes somebody is happy" as $\forall x\exists y(\mathit{student}(x)\wedge\mathit{likes}(x,y) \rightarrow \mathit{happy}(x))$ is that $\mathit{likes}(x,y)$ is on the right-hand size of the implication, and therefore implicitly negated (as ϕ in the implication $\phi \rightarrow \psi$ really is negated as it is equivalent to $\neg\phi \vee \psi$). Moving a subformula between the left and right sides of an implication means that the formula is (implicitly) negated, which changes universal $\forall$ quantification to $\exists$ existential quantification, or vice versa.

The correct way to formalize the sentence is

$$\forall x.(\mathit{student}(x) \wedge (\exists y.\mathit{likes}(x,y)) \rightarrow \mathit{happy}(x)).$$

You could rewrite this to

$$\forall x.(\mathit{student}(x) \rightarrow (\forall y.\neg\mathit{likes}(x,y) \vee \mathit{happy}(x))),$$

meaning that "every student either likes nobody or is happy", which is again the same as "every student who likes somebody is happy".

You could also rewrite the formula to move the quantifier outside the implication.

$$\begin{aligned}\forall x.(\mathit{student}(x) \wedge (\exists y.\mathit{likes}(x,y)) \rightarrow \mathit{happy}(x)) &\equiv \forall x.(\neg\mathit{student}(x) \vee \neg(\exists y.\mathit{likes}(x,y)) \vee \mathit{happy}(x))\\ &\equiv \forall x.(\neg\mathit{student}(x) \vee (\forall y.\neg\mathit{likes}(x,y)) \vee \mathit{happy}(x))\\ &\equiv \forall x.\forall y.(\neg\mathit{student}(x) \vee \neg\mathit{likes}(x,y) \vee \mathit{happy}(x))\\ &\equiv \forall x.\forall y.(\neg(\mathit{student}(x) \wedge \mathit{likes}(x,y)) \vee \mathit{happy}(x))\\ &\equiv \forall x.\forall y.(\mathit{student}(x) \wedge \mathit{likes}(x,y) \rightarrow \mathit{happy}(x))\end{aligned}$$

So, for all x and y, if x likes y, then x is happy. It is sufficient for there to be one y liked by x, so this correctly represents "every student who likes somebody is happy."

When formalizing natural language sentences, in general it is a good idea to put the quantifier as far inside the formula as possible, to make its scope as small as possible, which tends to make the context of the quantification more clear.

2. The second formula is a correct formalization of the sentence: there is at least one entity that is both a mammal and a marine animal.

   The first formula can be expressed logically equivalently as $\exists x(\neg\mathit{mammal}(x) \vee \mathit{marineAnimal}(x))$, that is, "there is at least one entity that is either not a mammal or is a marine animal." This sentence does not even require that there are any mammals or marine animals. A similar (in the real world true) formula would be $\exists x.(\mathit{vampire}(x) \rightarrow \mathit{human}(x))$, meaning, "There is some entity for which it holds that if it is a vampire, then it is a human being." This is true simply because no vampires exist: We can pick any value c for x, and $\mathit{vampire}(c) \rightarrow \mathit{human}(c)$ holds simply because $\mathit{vampire}(c)$ is false.

Answer of exercise 6.7

1. Everybody has a mother.
2. Somebody is everybody's mother.
3. All students are clever.
4. There are no students who are not clever. (This is logically the same as the "All students are clever.")

double negation	$\neg\neg\alpha$	$\equiv$	α
associativity $\vee$	$\alpha \vee (\beta \vee \gamma)$	$\equiv$	$(\alpha \vee \beta) \vee \gamma$
associativity $\wedge$	$\alpha \wedge (\beta \wedge \gamma)$	$\equiv$	$(\alpha \wedge \beta) \wedge \gamma$
commutativity $\vee$	$\alpha \vee \beta$	$\equiv$	$\beta \vee \alpha$
commutativity $\wedge$	$\alpha \wedge \beta$	$\equiv$	$\beta \wedge \alpha$
distributivity $\wedge$ $\vee$	$\alpha \wedge (\beta \vee \gamma)$	$\equiv$	$(\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
distributivity $\vee$ $\wedge$	$\alpha \vee (\beta \wedge \gamma)$	$\equiv$	$(\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
idempotence $\vee$	$\alpha \vee \alpha$	$\equiv$	α
idempotence $\wedge$	$\alpha \wedge \alpha$	$\equiv$	α
absorption 1	$\alpha \wedge (\alpha \vee \beta)$	$\equiv$	α
absorption 2	$\alpha \vee (\alpha \wedge \beta)$	$\equiv$	α
De Morgan's law 1	$\neg(\alpha \vee \beta)$	$\equiv$	$(\neg\alpha) \wedge (\neg\beta)$
De Morgan's law 2	$\neg(\alpha \wedge \beta)$	$\equiv$	$(\neg\alpha) \vee (\neg\beta)$
contraposition	$\alpha \rightarrow \beta$	$\equiv$	$\neg\beta \rightarrow \neg\alpha$
negation $\top$	$\neg\top$	$\equiv$	$\bot$
negation $\bot$	$\neg\bot$	$\equiv$	$\top$
constant $\bot$	$\alpha \wedge \neg\alpha$	$\equiv$	$\bot$
constant $\top$	$\alpha \vee \neg\alpha$	$\equiv$	$\top$
elimination $\top$ $\vee$	$\top \vee \alpha$	$\equiv$	$\top$
elimination $\top$ $\wedge$	$\top \wedge \alpha$	$\equiv$	α
elimination $\bot$ $\vee$	$\bot \vee \alpha$	$\equiv$	α
elimination $\bot$ $\wedge$	$\bot \wedge \alpha$	$\equiv$	$\bot$
elimination $\bot$ $\rightarrow$	$\bot \rightarrow \alpha$	$\equiv$	$\top$
elimination $\bot$ $\rightarrow$	$\alpha \rightarrow \bot$	$\equiv$	$\neg\alpha$
elimination $\top$ $\rightarrow$	$\top \rightarrow \alpha$	$\equiv$	α
elimination $\top$ $\rightarrow$	$\alpha \rightarrow \top$	$\equiv$	$\top$
commutativity $\leftrightarrow$	$\alpha \leftrightarrow \beta$	$\equiv$	$\beta \leftrightarrow \alpha$
elimination $\top$ $\leftrightarrow$	$\top \leftrightarrow \alpha$	$\equiv$	α
elimination $\bot$ $\leftrightarrow$	$\bot \leftrightarrow \alpha$	$\equiv$	$\neg\alpha$
unit resolution	$\alpha \wedge (\neg\alpha \vee \beta)$	$\equiv$	$\alpha \wedge \beta$

Table 1: Propositional Equivalences

A Logical Equivalences

Logical equivalences that are sometimes useful about reasoning about logical formulas are listed in Table 1.

B Solution Methods for Basic Reasoning Tasks in Logic

Below we list some of the basic approaches to solve reasoning tasks about the propositional logic and the predicate logic.

problem	solution
Show that formula ϕ is satisfiable.	Give a valuation/structure v such that $v \models \phi$.
Show that a propositional formula ϕ is not satisfiable.	Construct truth-table; Column for ϕ contains 0 only.
Show that a predicate logic formula ϕ is not satisfiable.	Argue that there can be no structure S such that $S \models \phi$.
Show that a propositional formula ϕ is valid.	Construct truth-table; Column for ϕ contains 1 only.
Show that a predicate logic formula ϕ is valid.	Argue that there can be no structure S such that $S \models \neg\phi$.
Show that logical consequence $\phi \models \psi$ does not hold.	Construct a valuation/structure v such that $v \models \phi$ and $v \not\models \psi$.
Show that logical consequence $\phi \models \psi$ holds.	Same as showing that $\phi \rightarrow \psi$ is valid.
How many valuations satisfy propositional formula ϕ?	Build truth table (if the table is not too big)
Show that formulas ϕ_1 and ϕ_2 are equivalent.	Use logical equivalences to rewrite ϕ_1 step by step to ϕ_2.

In the propositional logic, most reasoning tasks can be reduced to constructing the truth-table. However, in easy cases constructing the truth-table is unnecessary, and for example if you just need to show that a formula ϕ is satisfiable, it is sufficient to just give one valuation v such that $v \models \phi$.

Note that manually constructing a truth-table when the number of atomic propositions is 5 or more quickly becomes too tedious, as the number of rows starts getting too high. In those cases it is best to try some other approach than truth-tables.

For the predicate logic, there is no (finite) counterpart of constructing the truth-table. To show that a formula ϕ in the predicate logic is satisfiable, just give a structure S so that $S \models \phi$. If you do not directly see how that kind of structure would like, try simple structures first, with one or two or three elements, to see what is needed to make the formula true.