CS-E4800 Artificial Intelligence: Exercises

Jussi Rintanen
Department of Computer Science
Aalto University
Helsinki, Finland

April 8, 2024

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. Ari 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.

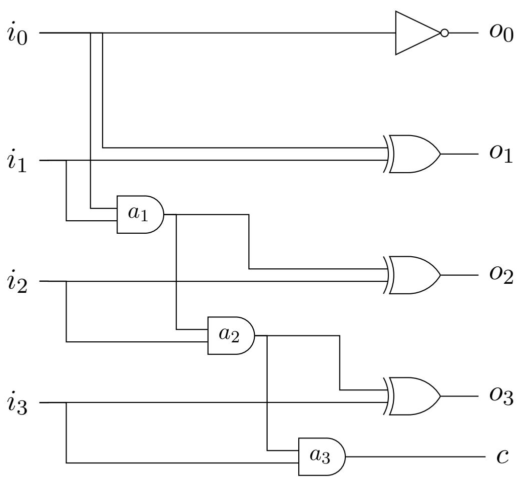
We denote the input bits by i_0, i_1, i_2, i_3 and the output bits by o_0, o_1, o_2, o_3. The bit 0 is the *least significant bit* and the bit 3 is the *most significant bit*. The bit 0 represents 1, the bit 1 represents 2, the bit 2 represents 4, and the bit 3 represents 8, so that these four bits can 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. 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. $\top \vee \bot \vee \top \equiv \top$
3.
4.

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. $A \wedge B \wedge C$
2. $A \wedge (B \vee \neg A)$
3. $(A \to B) \wedge (\neg A \to B)$
4. $(A \vee B) \wedge (A \to \neg C) \wedge (B \to \neg C) \wedge C$

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 $\phi_1 \wedge \phi_2$ is satisfiable.
3. If the formulas ϕ_1 and ϕ_2 are satisfiable, then also $\phi_1 \vee \phi_2$ is satisfiable.
4. If at least one of the formulas ϕ_1 and ϕ_2 is satisfiable, then also $\phi_1 \vee \phi_2$ is satisfiable.

Exercise 2.8

Are the following formulas satisfiable?

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

Exercise 2.9

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

1. Is the formula $\phi \wedge i_0 \wedge o_0$ satisfiable?
2. Is the formula $\phi \wedge i_1 \wedge o_1$ 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 $\alpha \to \beta$ is equivalent to $\neg\alpha \vee \beta$.

Exercise 2.10

Do the following hold?

1. $A \wedge (\neg B \to \neg A) \models B$
2. $A \wedge \phi \models A$ for any formula ϕ
3. $A \vee \phi \models A$ for any formula ϕ
4. $A \wedge (A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to E) \models E$
5. $(A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to E) \wedge \neg E \models \neg B$
6. $(A \vee B) \wedge (A \to C) \wedge (B \to C) \models C$

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. $A \wedge (A \to B) \wedge (B \to C) \wedge (C \to D) \wedge (D \to E) \wedge (E \to F)$

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.

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

3.
4.

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
2. yes
3. yes
4. no

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 or typically *not* 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.
2. The formula is satisfiable, because it is a disjunction with at least one satisfiable disjunct (actually, every disjunct is satisfiable). Any valuation that satisfies one of the disjuncts satisfies the whole formula.

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 a 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. 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*, 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 $a_1, \ldots, a_n$, generating a state sequence $s_0, \ldots, s_n$, so that s_0 is the initial state, and each a_i maps s_{i-1} to s_i (for each $i \in \{1, \ldots, n\}$).

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.

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 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?

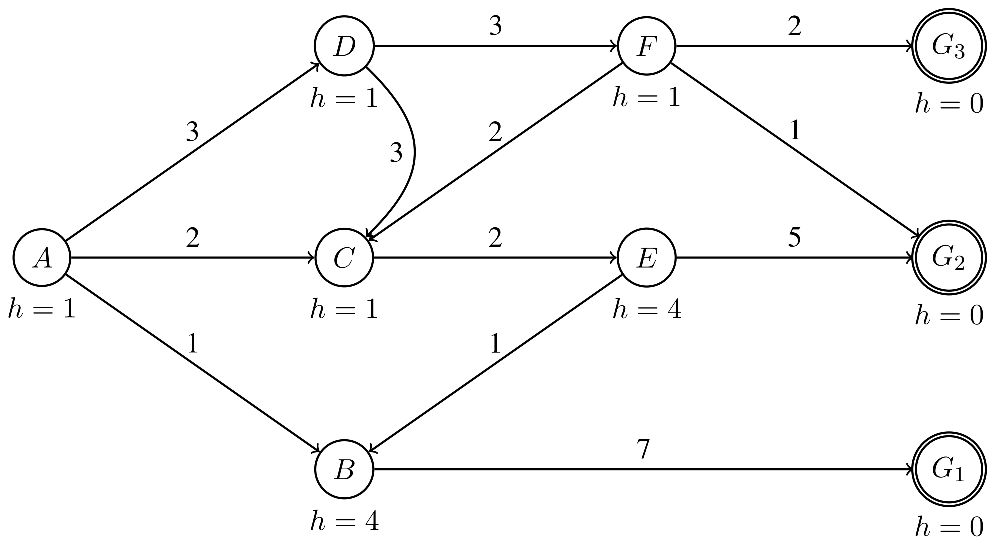


Figure 1: A map annotated with h-values and arc weights

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.

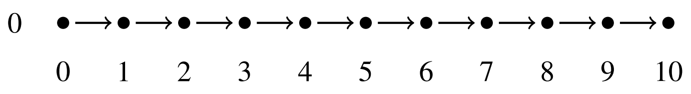
Are the following *admissible heuristics* (cost lower bounds) when transporting all packages in P?

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

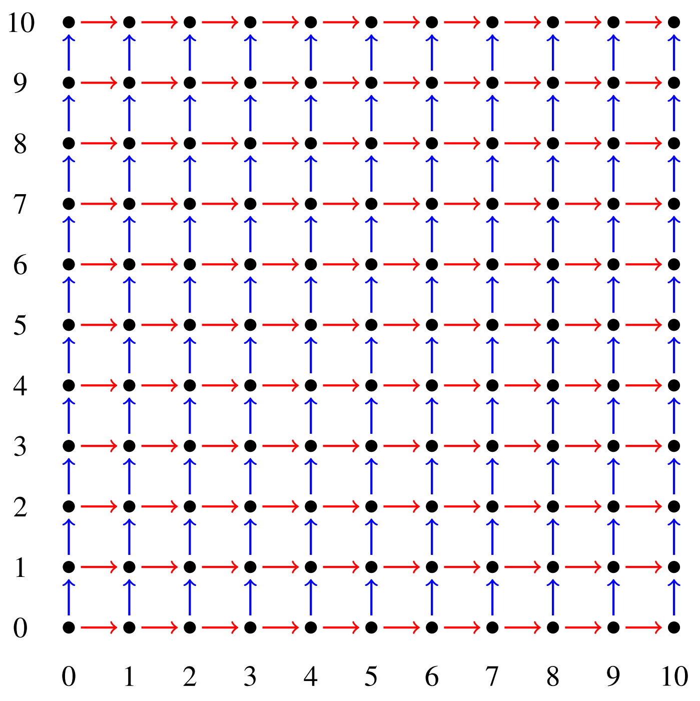
3.3 Solutions

Answer of exercise 3.1

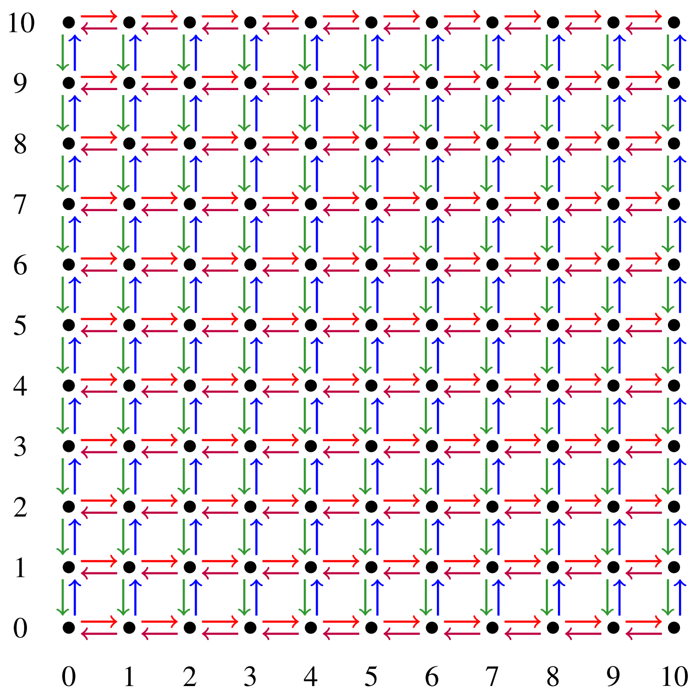
1. The value of y cannot be changed, and the possible values of x are all integers from 0 to 10. Hence the reachable states are $(0, 0), (1, 0), (2, 0), \ldots, (10, 0)$. The number of reachable states is **11**.



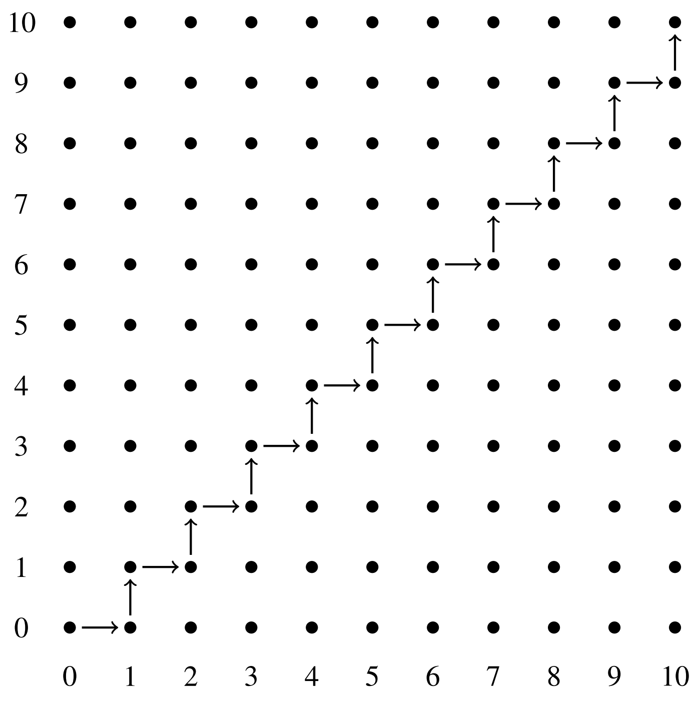
2. Now we can move from $(0, 0)$ to any (x, y) with $0 \le x \le 10$ and $0 \le y \le 10$ (but we could not get back to the initial state $(0, 0)$, as we cannot decrease the values of x and y. The reachable part of the state space is $\{0, \ldots, 10\} \times \{0, \ldots, 10\}$, and hence there are $11 \times 11 = \mathbf{121}$ 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, $11 \times 11 =$ **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. $a \to (b \to c) \equiv \neg a \vee (\neg b \vee c) \equiv \neg a \vee \neg b \vee c$
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. $$\begin{aligned}(a \wedge b) \vee (c \wedge d) &\equiv ((a \wedge b) \vee c) \wedge ((a \wedge b) \vee d) && \text{distributivity}\\ &\equiv (c \vee (a \wedge b)) \wedge (d \vee (a \wedge b)) && \text{commutativity of } \vee\\ &\equiv ((c \vee a) \wedge (c \vee b)) \wedge ((d \vee a) \wedge (d \vee b)) && \text{distributivity } \vee\wedge\\ &\equiv (c \vee a) \wedge (c \vee b) \wedge (d \vee a) \wedge (d \vee b) && \text{removal of parentheses}\end{aligned}$$

   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 $a \to a \equiv \neg a \vee a \equiv \top$ and $a \to \top \equiv \top$
4. $a \wedge b \wedge c$, obtained by similar reasoning as in (2) above.

Answer of exercise 5.4

1. $$\begin{aligned}\neg a \to (b \vee c) &\equiv \neg\neg a \vee (b \vee c) && \text{Definition of implication}\\ &\equiv a \vee (b \vee c) && \text{Double negation}\\ &\equiv (a \vee b) \vee c && \text{Associativity of } \vee\\ &\equiv c \vee (a \vee b) && \text{Commutatitivy of } \vee\\ &\equiv \neg\neg c \vee (a \vee b) && \text{Double negation}\\ &\equiv \neg c \to (a \vee b) && \text{Definition of implication}\end{aligned}$$

2. $$\begin{aligned}a \to (b \wedge c) &\equiv \neg a \vee (b \wedge c) && \text{Definition of implication}\\ &\equiv (\neg a \vee b) \wedge (\neg a \vee c) && \text{Distributivity}\\ &\equiv (b \vee \neg a) \wedge (c \vee \neg a) && \text{Commutativity of } \vee\\ &\equiv (\neg\neg b \vee \neg a) \wedge (\neg\neg c \vee \neg a) && \text{Double negation}\\ &\equiv (\neg b \to \neg a) \wedge (\neg c \to \neg a) && \text{Definition of implication}\end{aligned}$$

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.

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

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

Exercise 6.3

Argue why the following logical consequences hold.

1. $\forall x.P(x) \models P(a)$
2. $\{\forall x.(P(x) \to Q(x)), P(a)\} \models Q(a)$
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.

Exercise 6.6

Explain why:

1. $\forall x \exists y(\mathit{student}(x) \land \mathit{likes}(x, y) \to \mathit{happy}(x))$ doesn't correctly formalize "Every student who likes somebody is happy."
2. Not both of $\exists x(\mathit{mammal}(x) \to \mathit{marineAnimal}(x))$ and $\exists x(\mathit{mammal}(x) \land \mathit{marineAnimal}(x))$ are correct formalizations "Some mammals are marine animals."

[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.

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. $\forall x.(student(x) \rightarrow clever(x))$
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. $\mathcal{S} \models P(a) \wedge (P(a) \rightarrow P(b)) \wedge \neg P(b)$: 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 $P(a) \rightarrow P(b)$ 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. $\{\forall x.(P(x) \to Q(x)), P(a)\} \models Q(a)$: Assume that $\mathcal{S}$ is a structure such that $\mathcal{S} \models (\forall x.(P(x) \to Q(x))) \wedge P(a)$. Since $\mathcal{S} \models P(a)$, we have $[\![a]\!]^{\mathcal{S}} \in [\![P]\!]^{\mathcal{S}}$. Since $\mathcal{S} \models \forall x.(P(x) \to Q(x))$, 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. $\forall x.(\mathit{student}(x) \to \exists y.\mathit{knows}(x, y))$

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 $\mathit{student}(x) \wedge \mathit{likes}(x, y) \to \mathit{happy}(x)$ 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.

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

   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.

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

   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 things in formalizing "Every student who likes somebody is happy" as $\forall x\exists y(\mathit{student}(x) \wedge \mathit{likes}(x, y) \to \mathit{happy}(x))$ is that $\mathit{likes}(x, y)$ is on the right-hand size of the implication, and therefore implicitly negated (as ϕ in the implication $\phi \to \psi$ 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

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

You could rewrite this to

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

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.

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

So, for all x and y, if x likes y, then x is happy. It is sufficient 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 $\exists x(\neg mammal(x) \vee marineAnimal(x))$, 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 $\exists x.(vampire(x) \rightarrow human(x))$, 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 $vampire(c) \rightarrow human(c)$ holds simply because $vampire(c)$ 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.")

7 Exercises: Predicate Logic and Applications

7.1 Predicate Logic for Databases

Exercise 7.1

The following are the column names for the relations that represent the predicates of the same name.

predicate/relation	column names
U	object
parentOf	parent, child
female	person

Translate the following formulas into relational algebra.

- $\exists y.\text{parentOf}(x, y)$
- $\exists y.(\text{parentOf}(x, y) \vee \text{female}(x))$
- $\forall y.(\text{parentOf}(x, y) \rightarrow \text{female}(y))$

Exercise 7.2

Consider the query $P(x, y) \wedge P(y, z)$ that tries to identify all value combinations for x, y, z for which the query formula is true. Let there are 1000 objects in the database's universe which occur in the database tables for the predicate P. Let there be 1000 rows in the database table for P.

1. One way of answering the query is to loop over all value combinations for x, y, and z, and then see, if that value combination is in the table for P. How many combinations will you be testing this way?
2. The other way is to reduce the query to relational algebra, and compute the value of the relational algebra expression. How many value combinations (rows in the intermediate results of query evaluation) will you be considering (in the best case, and in the worst case)?

7.2 Predicate Logic for Natural Language Semantics

Exercise 7.3

Which variables are free in the following λ-expressions?

1. $\lambda x.x$
2. $\lambda x.xy$
3. $(\lambda x.xy)(\lambda y.xy)$
4. $(\lambda x.xy)(\lambda x.\lambda y.z(\lambda z.xy))$

Exercise 7.4

Reduce the following λ-expressions as far as possible. Is an infinite sequence of reductions possible?

1. $(\lambda x.x)(\lambda y.y)z$
2. $(\lambda x.xx)(\lambda x.xx)$
3. $(\lambda x.\lambda y.xy)(\lambda y.yy)$

Exercise 7.5

Assign a type to the following λ-expressions that include values of type $\mathbb{N}$ and functions on natural numbers.

1. $\lambda x.x + 1$
2. $\lambda x.\lambda y.x + y + 1$
3. $\lambda f.\lambda x.x + f(4)$
4. $\lambda f.\lambda x.1 + f(\lambda y.2 + x(5))$

Exercise 7.6

Show that the following sentences are syntactically correct according to the grammar given in the lecture, by constructing a parse tree for them.

1. John owns a dog.
2. John owns a dog that eats grass.

Here “John” is a name, “owns” and “eats” are transitive verbs, and “dog” is a predicate.

Exercise 7.7

Derive the meanings for the sentences in the previous exercise by using the Montague grammar given in the course material.

7.3 Solutions

Answer of exercise 7.1

- $\pi_x(\rho_{x/\text{parent},y/\text{child}}(\text{parentOf}))$ (The query means: “All x such that x is somebody’s parent.”)
- $\pi_x(\rho_{x/\text{parent},y/\text{child}}(\text{parentOf}) \cup (\rho_{y/\text{object}}(\text{U}) \bowtie \rho_{x/\text{person}}(\text{female})))$ (This means: “All x who are either parents of female.”)
- We rewrite the formula to eliminate $\rightarrow$ and to make the relational union operation applicable.

$\forall y.(\text{parentOf}(x,y) \rightarrow \text{female}(y))$	The formula
$\forall y.(\neg\text{parentOf}(x,y) \vee \text{female}(y))$	Elimination of $\rightarrow$
$\forall y.(\neg\text{parentOf}(x,y) \vee (U(x) \wedge \text{female}(y)))$	Same free variables in both disjuncts

  Finally, we translate this to the relational algebra.

  $((\rho_{x/\text{object}}(\text{U}) \bowtie \rho_{y/\text{object}}(\text{U})) - (\rho_{x/\text{parent},y/\text{child}}(\text{parentOf}))) \cup (\rho_{x/\text{object}}(\text{U}) \bowtie \rho_{y/\text{person}}(\text{female}))) \div \rho_{y/\text{object}}(U)$

  The query means: “All x such that all children of x are female.”

Answer of exercise 7.2

1. This is $1000 \times 1000 \times 1000 = 10^9$. This is still feasible, but may still take quite a bit time (several seconds), depending on how the database tables are accessed. Note that this one billion is completely independent of the properties of the table P.
2. The query $P(x,y) \wedge P(y,z)$ translates to a relational algebra query that involves renaming the columns of P in two different ways, and then performing a natural join. The result of the natural join can have at most 1000×1000 rows in the worst case, assuming that for every value in the second column matches every value in the first column. So this one million is the worst case for this query. In the best case there are few matches only, and the time it takes to perform the natural join is close to linear in the size of the table even without any kind of smart indexing of the values in the tables (which could make the runtime sub-linear in the table size.)

So, in summary, relational algebra may substantially reduce the amount of work done in query evaluation, in comparison to the most brute force way of processing the query.

Answer of exercise 7.3

The free variables are as follows. By free(E) we denote the free variables in E.

1. $\text{free}(\lambda x.x) = \{\}$
2. $\text{free}(\lambda x.xy) = \{y\}$
3. $\text{free}((\lambda x.xy)(\lambda y.xy)) = \{x,y\}$
4. $\text{free}((\lambda x.xy)(\lambda x.\lambda y.z(\lambda z.xy))) = \{y,z\}$

Clearly, the notion of “free variable” in the λ-calculus is similar to the same notion in the predicate logic: each variable is bound in the scope of a quantifier, and in the λ-calculus it is the λ symbol that is the quantifier.

Answer of exercise 7.4

1. $(\lambda x.x)(\lambda y.y)z \rightsquigarrow (\lambda y.y)z \rightsquigarrow z$
2. $(\lambda x.xx)(\lambda x.xx) \rightsquigarrow (\lambda x.xx)(\lambda x.xx) \rightsquigarrow \cdots$ (infinite reduction sequence)

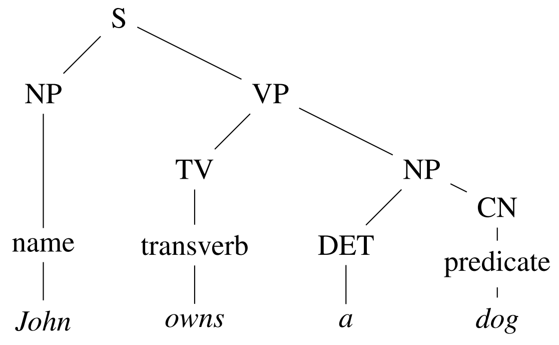
3. $(\lambda x.\lambda y.xy)(\lambda y.yy) \rightsquigarrow \lambda y.(\lambda y.yy)y \rightsquigarrow \lambda y.yy$

Answer of exercise 7.5

1. $\lambda x.x + 1$ is of type $\mathbb{N} \to \mathbb{N}$
2. $\lambda x.\lambda y.x + y + 1$ is of type $\mathbb{N} \to (\mathbb{N} \to \mathbb{N})$
3. $\lambda f.\lambda x.x + f(4)$ is of type $(\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$
4. $\lambda f.\lambda x.1 + f(\lambda y.2 + x(5) + y)$ is harder to type. We find the typing step by step, by inferring the types of some of the sub-expressions first.
   - x is of type $\mathbb{N} \to \mathbb{N}$ as it clearly maps numbers to numbers.
   - f is of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ as its input is a function that maps a number y to a number.
   - Hence the type of the whole expression is $((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N})$.

Answer of exercise 7.6

1. The word "John", "owns" and "dog" are first recognized respectively as *name*, *transverb* and *predicate*, and parsed as *NP*, *TV* and *CN*. After that, "a dog" is parsed by the rule *NP : DET CN*, and then "owns a dog" is parsed by the rule *VP : TV NP*, and finally "John owns a dog" is parsed by the rule *S : NP VP*. The parse tree is depicted below.



2.

Answer of exercise 7.7

1. For "John owns a dog", all four of these words have a meaning directly associated with them in one of the grammer rules.
   - For "John" this is $\lambda P.(P \text{ John})$, where *John* is a term in the predicate logic, consisting of the constant symbol *John*.
   - For "owns", a transitive verb, this is $\lambda y.\lambda x.\text{owns}(x, y)$, where *owns* is a 2-place predicate symbol.
   - For "a" this is $\lambda P.\lambda Q.\exists x((P\ x) \wedge (Q\ x))$.
   - For "dog" this is $\lambda x.\text{dog}(x)$, applying to common nouns that correspond to unary predicate symbols in the predicate logic.

   Now we form meanings for the more complex expressions "a dog", "owns a dog" and, finally "John owns a dog". These are as follows.
   - "a dog": The relevant rule is $NP : DET\ CN$, which applies the λ-expression for "a" to the one for "dog", resulting in $\lambda Q.\exists x(\text{dog}(x) \wedge (Q\ x))$.
   - "own a dog": The relevant rule is $VP : TV\ NP$, which is associated with the meaning $\lambda x.(NP\ (\lambda y.(TV\ y\ x)))$. So we have

"owns a dog"	VP : TV NP	$\lambda x.(NP\ (\lambda y.(TV\ y\ x)))$
"owns"	TV	$\lambda y.\lambda x.\text{owns}(x, y)$
"a dog"	NP	$\lambda Q.\exists x(\text{dog}(x) \wedge (Q\ x))$

   Here we plug in for NP and TV the meanings of "a dog" and "owns", respectively. This produces

$$\lambda x.((\lambda Q.\exists x(\text{dog}(x) \wedge (Q\ x)))\ (\lambda y.((\lambda y.\lambda x.\text{owns}(x, y))\ y\ x)))$$

   We perform β-reductions for $\lambda y.\lambda x$, obtaining

$$\lambda x.((\lambda Q.\exists x(\text{dog}(x) \wedge (Q\ x)))\ (\lambda y.\text{owns}(x, y))).$$

To avoid confusion with the x in λx and the x in $\exists x$, we rename the latter x to z.

$$\lambda x.((\lambda Q.\exists z(\text{dog}(z) \wedge (Q\ z)))\ (\lambda y.\text{owns}(x, y))).$$

We perform β-reduction for λQ, obtaining

$$\lambda x.(\exists z(\text{dog}(z) \wedge ((\lambda y.\text{owns}(x, y)))\ z))$$

We perform β-reduction for λy, obtaining

$$\lambda x.(\exists z(\text{dog}(z) \wedge \text{owns}(x, z)))$$

- Finally, we apply $\lambda P.(P\ \text{John})$ to the previous λ-expressions, obtaining

$$\exists z(\text{dog}(z) \wedge \text{owns}(\text{John}, z))$$

2.

8 Exercises: Bayesian Nets and Probabilistic Reasoning

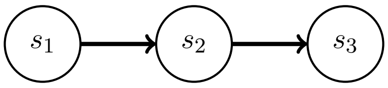
Exercise 8.1

Show that each of the following is probabilistically consistent by giving a probability distribution in which the statements are true.

1. $P(A|B) = 0.9$, $P(B|A) = 0.1$
2. $P(A|B) = 0.9$, $P(B|C) = 0.9$, $P(A|C) = 0.1$

Exercise 8.2

Consider Boolean random variables s_1, s_2, and s_3 denoting that it *snows* in Kaisaniemi the first day, the second day and the third day of observation, respectively. Below there is a Bayesian network modeling the dependencies of these variables:



$P(s_1)$
0.226

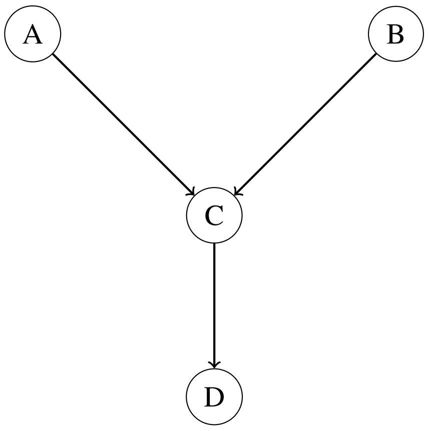
s_1	$P(s_2|s_1)$
F	0.018
T	0.951

s_2	$P(s_3|s_2)$
F	0.018
T	0.951

What is the probability $P(s_1|s_3)$?

Exercise 8.3

Answer the questions based on this Bayesian net.



$P(A)$
0.3

$P(B)$
0.9

C	$P(D|C)$
0	0.9
1	0.8

A	B	$P(C|A,B)$
0	0	0.9
0	1	0.8
1	0	0.0
1	1	0.2

Calculate the following probabilities?

1. $P(A|B)$
2. $P(C|A,B)$
3. $P(\neg C|\neg A,\neg B)$
4. $P(A|C)$

8.1 Solutions

Answer of exercise 8.1

1. To show that $P(A|B) = 0.9$, $P(B|A) = 0.1$ is consistent, we give a probability distribution by enumerating all valuations fo the variables and associating a probability with each valuation. The probabilities have to sum up to 1.0. We obtain the probabilities as follows from the requirements that $\frac{P(A,B)}{P(B)} = 0.9$ and $\frac{P(A,B)}{P(A)} = 0.1$.

$$\frac{p_{A,B}}{p_{A,B}+p_{\neg A,B}} = 0.9$$
$$\frac{p_{A,B}}{p_{A,B}+p_{A,\neg B}} = 0.1$$

These do not uniquely determine the probabilities, so you have to fix some first, and the solve the rest.

One solution is as follows. Other solutions could be obtained by starting with a lower probability for $P(A,B)$, which would have left more probability for $P(\neg A, \neg B)$.

A	B	$P(A,B)$
0	0	0.09
0	1	0.01
1	0	0.81
1	1	0.09

2. $P(A|B) = 0.9$, $P(B|C) = 0.9$, $P(A|C) = 0.1$
This translates to the following requirements.

$$\frac{P(A,B)}{P(B)} = 0.9 \qquad \frac{P(B,C)}{P(C)} = 0.9 \qquad \frac{P(A,C)}{P(C)} = 0.1$$

Let's introduce a variable for the probabilities for each of the valuations.

A	B	C	$P(A,B,C)$
0	0	0	p_{000}
0	0	1	p_{001}
0	1	0	p_{010}
0	1	1	p_{011}
1	0	0	p_{100}
1	0	1	p_{101}
1	1	0	p_{110}
1	1	1	p_{111}

With the above, we can start assigning probabilities to individual valuations.

requirement	valuation's probabilities
$\frac{P(A,B)}{P(B)} = 0.9$	$\frac{p_{110}+p_{111}}{p_{010}+p_{011}+p_{110}+p_{111}} = 0.9$
$\frac{P(B,C)}{P(C)} = 0.9$	$\frac{p_{011}+p_{111}}{p_{001}+p_{011}+p_{101}+p_{111}} = 0.9$
$\frac{P(A,C)}{P(C)} = 0.1$	$\frac{p_{101}+p_{111}}{p_{001}+p_{011}+p_{101}+p_{111}} = 0.1$

If interpreted as sets, we could think of this as follows.
(a) Most elements of B are in A.
(b) Most elements of C are in B.
(c) Few elements of C are in A.
We could think of this as C being a very atypical subset of B, and it has to be a small set, so we associate with it a low probability.
The important thing at this stage is that the probabilities p_{001}, p_{011}, p_{101} and p_{111} are small enough so that $P(A|B) = 0.9$ is still achievable, even when $P(A|C) = 0.1$ and $P(B|C) = 0.9$. We choose these probabilities so that $P(C) = 0.1$ (or some other quite small probability), and then it must be that $P(A,C) = 0.01$ and $P(B,C) = 0.09$. This is as follows.

$$\begin{aligned} p_{001} &= 0.005 \\ p_{011} &= 0.085 \\ p_{101} &= 0.005 \\ p_{111} &= 0.005 \end{aligned}$$

For all valuations we now have the following.

A	B	C	$P(A,B,C)$
0	0	0	p_{000}
0	0	1	0.005
0	1	0	p_{010}
0	1	1	0.085
1	0	0	p_{100}
1	0	1	0.005
1	1	0	p_{110}
1	1	1	0.005

What is left to do is choosing the remaining probabilities so that $\frac{P(A,B)}{P(B)} = 0.9$, by

$$\frac{p_{110} + 0.005}{p_{010} + 0.085 + p_{110} + 0.005} = 0.9. \qquad (1)$$

The choice of p_{110} and p_{010} also depends on p_{000} and p_{100}, as the sum of these four must be exactly 0.9 (which is what remains for $P(\neg C)$, as we already have $P(C) = 0.1$).
We arbitrarily pick $p_{110} = 0.8$, and from (1) solve $p_{010} = 0.005$. Note that a too small value for p_{110} would yield a negative value for p_{010} given (1). Now the probabilities look as follows.

A	B	C	$P(A,B,C)$
0	0	0	p_{000}
0	0	1	0.005
0	1	0	0.005
0	1	1	0.085
1	0	0	p_{100}
1	0	1	0.005
1	1	0	0.800
1	1	1	0.005

The probabilities excluding $p_{000} + p_{100}$ sum up to 0.905, and the probabilities p_{000} and p_{100} can be chosen arbitrarily as long as $p_{000} + p_{100} + 0.905 = 1.0$. One solution is simply $p_{000} = 0$ and $p_{100} = 0.095$.

Answer of exercise 8.2

We calculate the condition probability on the basis of the definition of conditional probabilities as $P(s_1|s_3) = \frac{P(s_1,s_3)}{P(s_3)}$. For this calculation, we have to make all parent variables explicit. This is

$$\begin{aligned} P(s_3) &= P(s_1, s_2, s_3) + P(\neg s_1, s_2, s_3) + P(s_1, \neg s_2, s_3) + P(\neg s_1, \neg s_2, s_3) \\ P(s_1, s_3) &= P(s_1, s_2, s_3) + P(s_1, \neg s_2, s_3) \end{aligned}$$

Next, we express each of the probabilities on the right-hand side by using the chain rule, in which the joint probability of two or more variables is expressed as a product of the conditional probability of each in terms of their parents. From this we get the following.

$$\begin{aligned} P(s_3) &= P(s_1)P(s_2|s_1)P(s_3|s_2)+P(\neg s_1)P(s_2|\neg s_1)P(s_3|s_2)+P(s_1)P(\neg s_2|s_1)P(s_3|\neg s_2)+P(\neg s_1)P(\neg s_2|\neg s_1)P(s_3|\neg s_2) \\ P(s_1, s_3) &= P(s_1)P(s_2|s_1)P(s_3|s_2) + P(s_1)P(\neg s_2|s_1)P(s_3|\neg s_2) \end{aligned}$$

Now all the probabilities here are something directly expressed in the CPTs of the variables. So the final step is looking up those numbers and calculating the result.

$$\begin{aligned} P(s_3) &= 0.226{\cdot}0.951{\cdot}0.951{+}(1{-}0.226){\cdot}0.018{\cdot}0.951{+}0.226{\cdot}(1{-}0.951){\cdot}0.018{+}(1{-}0.226){\cdot}(1{-}0.018){\cdot}0.018 \\ P(s_1, s_3) &= 0.226 \cdot 0.951 \cdot 0.951 + 0.226 \cdot (1 - 0.951) \cdot 0.018 \end{aligned}$$

The division $\frac{P(s_1,s_3)}{P(s_3)}$ yields the answer 0.88368162.

Answer of exercise 8.3

1. $P(A|B) = P(A) = 0.3$ because A is conditionally independent of B.
2. $P(C|A, B) = 0.2$ which can be directly read from the CPT for C.
3. $P(\neg C|\neg A, \neg B) = 0.9$ which can be directly read from the CPT for C: $P(C|\neg A \wedge \neg B) = 0.9$, so $P(\neg C|\neg A \wedge \neg B) = 1.0 - 0.9 = 0.1$.
4. This is calculated as $P(A|C) = \frac{P(A,C)}{P(C)}$
   First we calculate $P(A, C)$. We look at all valuations that are compatible with A, C.
   There are two possible values for B: either B is true, or it is false. There is also the hidden variable D, but since it is below the variables of interest (none of A, B or C depends on it), it can be ignored. Hence we are interested in the probabilities $P(A, C, B)$ and $P(A, C, \neg B)$.
   These probabilities are calculated with the chain rule, with each variable's probability expressed in terms of its parents.

$$\begin{aligned} P(A, C, B) &= P(A) \cdot P(C|A, B) \cdot P(B) \\ P(A, C, \neg B) &= P(A) \cdot P(C|A, \neg B) \cdot P(\neg B) \end{aligned}$$

Each of the probabilities on the right-hand side can be directly read from the CPTs for the Bayes network.

$$\begin{aligned} P(A,C,B) &= P(A) \cdot P(C|A,B) \cdot P(B) = 0.3 \cdot 0.2 \cdot 0.9 = 0.054 \\ P(A,C,\neg B) &= P(A) \cdot P(C|A,\neg B) \cdot P(\neg B) = 0.3 \cdot 0.0 \cdot 0.1 = 0.0 \end{aligned}$$

Hence $P(A,C) = P(A,C,B) + P(A,C,\neg B) = 0.054$.
It remains to calculate $P(C)$. Now there are all four valuations of A and B to be considered hidden (D can again be ignored), and we calculate

$$P(C) = P(A,B,C) + P(\neg A,B,C) + P(A,\neg B,C) + P(\neg A,\neg B,C).$$

Expressed in terms of each variable's parents, this is as follows.

$$\begin{aligned} P(C) = & \; P(A) \cdot P(B) \cdot P(C|A,B) \\ & + P(\neg A) \cdot P(B) \cdot P(C|\neg A,B) \\ & + P(A) \cdot P(\neg B) \cdot P(C|A,\neg B) \\ & + P(\neg A) \cdot P(\neg B) \cdot P(C|\neg A,\neg B) \end{aligned}$$

Again, all these probabilities can be directly read from the CPTs.

$$\begin{aligned} P(C) = & \; 0.3 \cdot 0.9 \cdot 0.2 \\ & + 0.7 \cdot 0.9 \cdot 0.8 \\ & + 0.3 \cdot 0.1 \cdot 0.0 \\ & + 0.7 \cdot 0.1 \cdot 0.9 = 0.621 \end{aligned}$$

Finally, we obtain the sought probability as

$$P(A|C) = \frac{P(A,C)}{P(C)} = \frac{0.054}{0.621} = 0.08695652.$$

Note that we could have got off with a little bit less work by – instead of calculating $P(C)$ from scratch – calculating $P(\neg A, C)$, and then obtaining $P(C)$ as $P(A,C) + P(\neg A, C)$.

9 Exercises: Game-Tree Search

No exercises on this topic.

10 Exercises: Decision-Making Under Uncertainty

10.1 Markov Chains

Exercise 10.1

Given a finite system that consists of a number of discrete states, with known probabilities for all possible transitions from a state to another (this is a *Markov chain*!), an important problem is to determine the probability of the system being in a given state.

If the graph formed by the states and the possible transitions between them is connected (there is a path from every node to every other node), the system satisfies a condition of *periodicity* (which we do not discuss here in more detail), and the system has been running for a long time (number of steps much higher than the number of states), then the probability of each state at any given time point is independent of the initial state of the system.

These *steady-state* probabilities can be determined by solving a set of linear equations, with the probability of each state expressed in terms of the probabilities of its predecessor states and the probabilities of the transitions from those predecessor states to the given state.

$$\begin{tikzcd} C \arrow[d, bend right] & \\ A \arrow[u, "0.6"'] \arrow[r, "0.4"'] & B \arrow[ul] \end{tikzcd}$$

Determine the steady state probabilities of states A, B and C in the Markov chain above. (If there is a unique transition out of a state, its probability is not indicated in the graph, as it is necessarily 1.0.)

10.2 The Bellman Equation

Exercise 10.2

Show that the general form of the Bellman equation

$$v(s) = \max_{a \in A} \sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma v(s')]$$

reduces to the somewhat simpler form when the reward is independent of the successor state s':

$$v(s) = \max_{a \in A} \left(R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') v(s') \right).$$

10.3 Solutions

Answer of exercise 10.1

We denote the probabilities of the states A, B, and C by the variables p_A, p_B, and p_C. We have one equation for the probability for each state, and the requirement that the probabilities of all states sum up to 1.0.

$$p_A = p_C \tag{2}$$
$$p_B = 0.4 \cdot p_A \tag{3}$$
$$p_C = 0.6 \cdot p_A + p_B \tag{4}$$
$$p_A + p_B + p_C = 1.0 \tag{5}$$

To solve the unknowns, we first simplify with $p_A = p_C$.

$$p_B = 0.4 \cdot p_A \tag{6}$$
$$p_A = 0.6 \cdot p_A + p_B \tag{7}$$
$$2p_A + p_B = 1.0 \tag{8}$$

Then we substitute p_B in the last equation and solve it.

$$2p_A + 0.4 \cdot p_A = 1.0 \tag{9}$$

This yields $p_A = \frac{1}{2.4} = \frac{5}{12} \sim 0.41666667$. We have $p_C = p_A$ immediately, and solve p_B from

$$2p_A + p_B = 1.0 \tag{10}$$

Hence $p_B = \frac{1}{6} \sim 0.16666667$.

Answer of exercise 10.2

$$\begin{aligned}
\max_{a\in A} \sum_{s'\in S} P(s,a,s')[R(s,a,s') + \gamma v(s')] &= \max_{a\in A} \sum_{s'\in S} P(s,a,s')[R(s,a) + \gamma v(s')] && (11)\\
&= \max_{a\in A} \sum_{s'\in S} \big(P(s,a,s')R(s,a) + P(s,a,s')\gamma v(s')\big) && (12)\\
&= \max_{a\in A} \left(\sum_{s'\in S} P(s,a,s')R(s,a) + \sum_{s'\in S} P(s,a,s')\gamma v(s')\right) && (13)\\
&= \max_{a\in A} \left(R(s,a) + \sum_{s'\in S} P(s,a,s')\gamma v(s')\right) && (14)\\
&= \max_{a\in A} \left(R(s,a) + \gamma \sum_{s'\in S} P(s,a,s') v(s')\right) && (15)
\end{aligned}$$

The justification for (11) is that the reward is independent of s'. In (14) we have $\sum_{s'\in S} P(s,a,s')R(s,a) = R(s,a)$ because $\sum_{s'\in S} P(s,a,s') = 1$ for any s and a.

10.4 Partial Observability: Discrete Belief States

Exercise 10.3

Let $R = \{(a,b),(b,a),(c,d),(c,a),(d,c),(d,b),(e,c)\}$. Answer the following questions concerning belief states over the state space $\{a,b,c,d,e\}$.

1. What is the successor of the belief state $\{a,b\}$ when an action corresponding to the transition relation R is taken?
2. What is the successor of the belief state $\{a,c,d\}$ with respect to the same action?
3. Consider belief state $\{b,c\}$. Find a *predecessor* of this belief state.
4. Let there be two possible observations, the observation 1 indicating states $\{a,b,c\}$ and the observation 2 indicating states $\{d,e\}$. What are the possible new belief states that follow the belief state $\{b,c,d\}$ when one of the two observations is made?

10.5 Partial Observability: Probabilistic Belief States

Exercise 10.4

Consider the belief state $B = (0.3, 0.5, 0.2)$, indicating the belief probabilities of the states s_0, s_1, s_2, and an action with the transition probabilities indicated by the following matrix. The entry (s_i, s_j) indicates the probability of going from state s_i to state s_j.

$$M = \left(\begin{array}{c|ccc} & s_0 & s_1 & s_2 \\ \hline s_0 & 0.7 & 0.0 & 0.3 \\ s_1 & 0.1 & 0.2 & 0.7 \\ s_2 & 0.0 & 0.0 & 1.0 \end{array}\right)$$

1. Determine the new belief state after this action has been taken.
2. What matrix operation does this computation correspond to?
3. What if you multiply M by itself as MM? M^3? M^4?

Matrices with the property that each row sums to 1 and all elements are non-negative, are called *stochastic* matrices. They are used in analyzing Markov chains, Markov decision processes, and partially observable Markov decision processes.

Exercise 10.5

Consider the belief state $B = (0.3, 0.5, 0.2)$ (for states s_0, s_1, s_2) and the probabilities of some given observation:

$$P(O|s_0) = 0.9$$
$$P(O|s_1) = 0.1$$
$$P(O|s_2) = 0.3$$

What is the new belief state when this observation has been made?

10.6 Solutions

Answer of exercise 10.3

1. The successor belief state w.r.t. a transition relation is computed by using the *image* operation. We identify the successors of the states a and b by using R. For a there is only one successor state, b, and for b there is similarly only one successor state, a. The image of B is $\{s'|s \in B, sRs'\} = \{a,b\}$.
2. The successor of $B = \{a,c,d\}$ w.r.t R is $B' = \{a,b,c,d\}$.
3. One possible predecessor belief state of $B = \{b,c\}$ is $B' = \{a,e\}$ because $\text{img}_R(B) = B'$, but this is not the only possibility, as also $\text{img}_R(\{d\}) = B'$. The strong pre-image operation gives, for any B', a maximal belief state $B = \text{spreimg}_R(B')$ such that $B' = \text{img}_R(B)$. In this case $\text{spreimg}_R(\{b,c\}) = \{a,e,d\}$.
4. If we make observation 1 and we knew that the current state is one of $\{b,c,d\}$, then clearly this observation limits the possible current states to $\{b,c,d\} \cap \{a,b,c\} = \{b,c\}$. Similarly observation 2 limits the possible current states to $\{b,c,d\} \cap \{d,e\} = \{d\}$. Hence the possible new belief states after making an observation are $B_1 = \{b,c\}$ and $B_2 = \{d\}$.

Answer of exercise 10.4

1. The new belief state is (p_0, p_1, p_2) where p'_0, p'_1, p'_2 are the new probabilies for the states s_0, s_1, s_2, obtained by

$$p'_i = \sum_{j=0}^{2} p_j \cdot P(s_j, a, s_i)$$

where $P(s_j, a, s_i)$ is given by the entry (s_j, s_i) in the matrix for action a. This is $(0.3 \cdot 0.7 + 0.5 \cdot 0.1 + 0.2 \cdot 0.0, 0.3 \cdot 0.0 + 0.5 \cdot 0.2 + 0.2 \cdot 0.0, 0.3 \cdot 0.3 + 0.5 \cdot 0.7 + 0.2 \cdot 1.0) = (0.26, 0.1, 0.64)$

2. This is the matrix product

$$BM = (0.3, 0.5, 0.2) \begin{pmatrix} 0.7 & 0.0 & 0.3 \\ 0.1 & 0.2 & 0.7 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}$$

3. The element (i, j) of MM indicates the probability of moving from state s_i to s_j by taking the action consecutively two times. The calculation of this matrix product can be thought of as determining for every (i, j), all the possible ways of getting from s_i to s_j through some intermediate state s_k. In general, the n-fold multiplication M^n of M by itself gives the probabilities for going between states by taking the action n times consecutively. The special case M^0 is the identity matrix I that maps every belief state to itself, not changing it.

Answer of exercise 10.5

First we calculate the probability that the given observation is made in the given belief state. This is

$$\sum_{s \in \{s_0, s_1, s_2\}} P(O|s)B(s) = 0.9 \cdot 0.3 + 0.1 \cdot 0.5 + 0.3 \cdot 0.2 = 0.38$$

This is the denominator in the calculations of the new probabilities of the states by the Bayes rule, in which we divide the probability of making the observation in a given state (when the latter probability is given by the belief state) by the overall probability of making the observation in the belief state.

$$\begin{aligned} P(s_0) &= \tfrac{0.3 \cdot 0.9}{0.38} = 0.7105263158 \\ P(s_1) &= \tfrac{0.5 \cdot 0.1}{0.38} = 0.1315789474 \\ P(s_2) &= \tfrac{0.2 \cdot 0.3}{0.38} = 0.1578947368 \end{aligned}$$

So the belief state changes from $(0.3, 0.5, 0.2)$ to $(0.7105, 0.1316, 0.1579)$.

11 Exercises: Multi-Agent Decision-Making and Game Theory

11.1 Iterated Strict Dominance

Some games can be solved by repeatedly identifying strategies that would never need to be used by a rational player, and eliminating them from consideration. If for a player there is only one possible strategy left after this process, then that strategy is the best strategy for that player.[2]

The reason for a strategy never being used is that it is worse than some other strategy, no matter which strategy the opponent has chosen. Such a strategy is *strictly dominated* by another strategy.[3]

The number of iterations in this process can be $N - 1 + M - 1$ for a 2-player normal form game of size $N \times M$, and a strategy may have to be checked for dominance multiple times, as a strategy that was not previously dominated may become dominated after the removal of some opponent strategies. Iterated strict dominance is not only applicable to 2-player games, but also for any N-player games for any $N \geq 1$.

Exercise 11.1

Find a Nash equilibrium for the normal form game represented by the following matrix.

$$\begin{array}{c|ccc} & D & E & F \\ \hline A & 1,5 & 9,2 & 7,9 \\ B & 3,1 & 4,0 & 5,0 \\ C & 2,2 & 5,0 & 8,1 \end{array}$$

11.2 Finding Mixed Strategies

The general procedure for manually finding solutions to normal form games begins by the elimination of strategies from both players that are strictly dominated. If this leads to both players having only one strategy left, then that strategy combination is the unique Nash equilibrium.

Otherwise the game has one or more mixed strategies. There are systematic procedures for finding them, but in this course we look at a simple incomplete method that works with 2-player games with a 2×2 strategy / pay-off matrix. The same idea can be attempted with bigger games, as well, but it does not always work.

Exercise 11.2

Find a Nash equilibrium for the normal form game represented by the following matrix.

$$\begin{array}{c|ccc} & C & D & E \\ \hline A & 1,2 & 0,0 & 2,1 \\ B & 3,0 & 4,3 & 0,4 \end{array}$$

11.3 Extensive Form Games

For detailed information on games in extensive form, see Appendix C.

Exercise 11.3

1. Consider extensive form games of the form given in Figure 2. How many different strategies do players 1 and 2 have? (For each node, the player to choose an action in that node is indicated by the number.)

[2]The critical underlying assumption here is that all players are *rational*. If any of the agents behave irrationally, then there is of course no guarantee that the strategy combinations left are truly the ones that are played.

[3]A strategy is *weakly dominated* if it is never better than another strategy and in some cases worse. Pay-off vector $\langle p_1, \ldots, p_n \rangle$ is strictly dominated by $\langle r_1, \ldots, r_n \rangle$ if $p_i < r_i$ for all i such that $1 \leq i \leq n$. It is weakly dominated if $p_i \leq r_i$ for all i such that $1 \leq i \leq n$ and $p_i < r_i$ for some i such that $1 \leq i \leq n$.

When solving a game, if weakly dominated strategies are eliminated, and one ends up with only one strategy for all players, that strategy combination is a (pure strategy) Nash equilibrium. However, that Nash equilibrium is not necessarily the only Nash equilibrium, so the iterated elimination of weakly dominated strategies is not a method for finding *all* Nash equilibria.

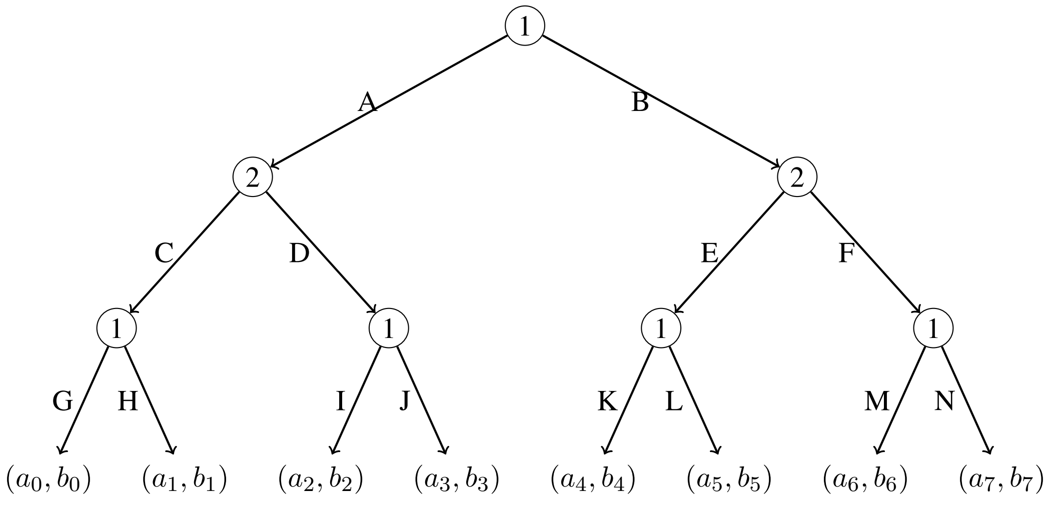


Figure 2: An extensive form game

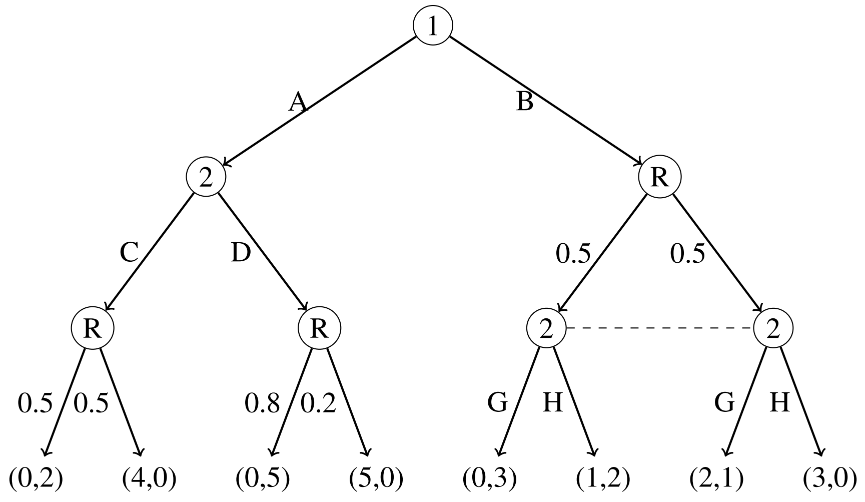


Figure 3: An imperfect information game with chance moves

2. Consider extensive form games (with perfect information) in which two players respectively have n and m nodes, and there are two actions to choose in each node. How many strategies does each player have? What is the size of the corresponding game in normal form?

Exercise 11.4

For the extensive form game in Figure 3 construct the corresponding normal form game, and find its Nash equilibria.

11.4 Solutions

Answer of exercise 11.1

The game can be solved by iterated strict dominance.

1. E is eliminated because D dominates it: (2,0,0) is worse than (5,1,2).
2. A is eliminated because C dominates it: (1,-,7) is worse than (2,-,8) (notice that the elimination of E before is important, because otherwise the middle payoff could not be ignored.)
3. F is eliminated because D dominates it.
4. C is eliminated because B dominates it.

Answer of exercise 11.2

As the very first thing, as always, we check that none of the strategies of neither of the players is strictly dominated by another strategy. And for this game this is indeed so.[4]

We denote the utilities player 1 (the row player) obtains from the strategies A and B by u_A and u_B, and the utilities of player 2 (the column player) similarly by u_C, u_D, and u_E.

First we observe that the strategy D for the column player is dominated by the strategy E, and hence D can be eliminated from consideration. No other strategy can be eliminated based on strict domination, so we proceed to identifying a mixed strategy equilibrium.

The probabilities of the strategies of a player have to be such that the opponent is indifferent between his strategies, that is, both opponent strategies have to have the same expected value considering the player's randomization between his strategies. Otherwise the opponent would be better off playing the better one of his two strategies, and the Nash equilibrium would collapse, as the player himself could also achieve a better outcome to switch to one his two strategies, and not randomize at all. So for the Nash equilibrium to hold, this condition of indifference has to hold for both players.

The utilities of each strategy, given the randomization probabilities of the respective opponents strategies, are as follows.

$$u_C = p_A \cdot 2 + p_B \cdot 0 \tag{16}$$
$$u_E = p_A \cdot 1 + p_B \cdot 4 \tag{17}$$
$$u_A = p_C \cdot 1 + p_E \cdot 2 \tag{18}$$
$$u_B = p_C \cdot 3 + p_E \cdot 0 \tag{19}$$

As the probabilities of the strategies for each player are respectively related by $p_A + p_B = 1$ and $p_C + p_E = 1$, we can express the equalities as follows.

$$u_C = p_A \cdot 2 + (1 - p_A) \cdot 0 \tag{20}$$
$$u_E = p_A \cdot 1 + (1 - p_A) \cdot 4 \tag{21}$$
$$u_A = p_C \cdot 1 + (1 - p_C) \cdot 2 \tag{22}$$
$$u_B = p_C \cdot 3 + (1 - p_C) \cdot 0 \tag{23}$$

After obvious simplifications we have:

$$u_C = 2p_A \tag{24}$$
$$u_E = p_A + 4(1 - p_A) \tag{25}$$
$$u_A = p_C + 2(1 - p_C) \tag{26}$$
$$u_B = 3p_C \tag{27}$$

Now, the indifference condition $u_C = u_E$ and $u_A = u_B$ results in the following.

$$2p_A = p_A + 4(1 - p_A) \tag{28}$$
$$p_C + 2(1 - p_C) = 3p_C \tag{29}$$

From this we solve the probabilities $p_A = \frac{4}{5}$ and $p_C = \frac{1}{2}$. Hence the probabilities for strategies A, B, C, D, and E are as follows.

$$\begin{array}{l} p_A = \frac{4}{5} \; p_B = \frac{1}{5} \\ p_C = \frac{1}{2} \; p_D = 0 \; p_E = \frac{1}{2} \end{array}$$

The utilities of all of the strategies are obtained from (16)-(19) by instantiating the probabilities. (Notice that this confirms that for both players, given the opponent's randomization probabilities, both strategies yield the same utility.)

$$u_C = \frac{8}{5} \; u_E = \frac{8}{5} \; u_A = \frac{3}{2} \; u_B = \frac{3}{2}$$

Now, clearly, utility of player 1 under this Nash equilibrium is $\frac{3}{2}$, and the utility of player 2 is $\frac{8}{5}$.

Answer of exercise 11.3

[4]The method presented for finding mixed strategies will typically lead to probabilities that don't make sense, e.g. by being negative, if there is a unique pure strategy equilibrium.

1. Player 1 has 5 nodes, with 2 actions in each of them, and as this game has perfect information, choices in these 5 nodes are independent, so the number of different strategies is $2^5 = 32$. Similarly, player 2 has 2 nodes, so the number of strategies is $2^2 = 4$.
2. The choices in each node are independent, and hence the first player has 2^n different strategies, and the second player has 2^m different strategies. The corresponding normal form game is represented as a 2-dimensional table with as many rows and columns as there are strategies for the respective players, and hence there is a total of $2^n \times 2^m = 2^{n+m}$ cells in that table.

Answer of exercise 11.4

The two players have the following choices in any strategy.

1. Player 1 can choose between actions A and B.
2. Player 2 can choose between action C and D, and between action G and H. Note that although the choice between G and H is in two nodes, the choice in a given strategy must be the same for both nodes, as player 2 cannot distinguish between the nodes.

Hence the strategies that are available in this game are as follows.

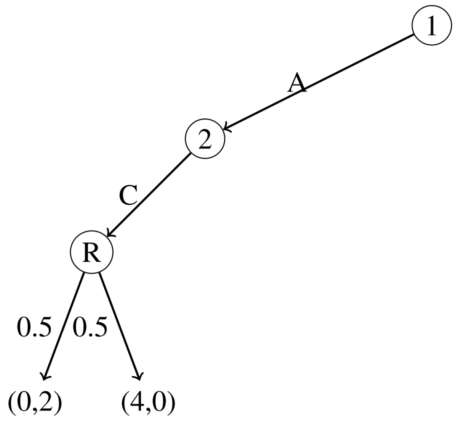
player	the strategy	explanation
1	A	player 1 chooses action A
1	B	player 1 chooses action B
2	CG	player 2 chooses actions C and G in the respective nodes
2	CH	player 2 chooses actions C and H in the respective nodes
2	DG	player 2 chooses actions D and G in the respective nodes
2	DH	player 2 chooses actions D and H in the respective nodes

In a slightly different game, in which the dashed line between the two nodes on the right was missing, player 2 could independently choose G and H in those two nodes, and the number of strategies for player 2 would be 8 instead of 4. But because player 2 cannot distinguish between those two nodes, this is not the case.

Next, we will construct the normal form game by determining the pay-off vectors for the two players for every possible strategy combination. We make player 1 the row player, and player 2 the column player, so there are 2 rows and 4 columns in the matrix representation of the normal form game.

	CG	CH	DG	DH
A				
B				

To calculate the pay-offs for any strategy combination, we look which parts of the extended form game are actually going to be played. Consider the strategy combination *A,CG*. If we keep only those parts of the game-tree which are relevant for this strategy, we get the following sub-graph of the original extended form game.



The leaf nodes yield pay-offs $(0, 2)$ and $(4, 0)$, and the chance move leads to them with the probabilities 0.5 and 0.5, respectively, so we obtain the value for the chance nodes by expected values, which in this case is simply $(2, 1)$. And this is the pay-off vector for the strategy combination *A,CG*. Note that we immediately get the same pay-off vector for the strategy combination *A,CH* because player 1 playing A makes the choice between G and H irrelevant, as that part of the extended game is never played under this strategy profile.

The values of the other strategy combinations are obtained similarly.

- Leaf nodes' pay-off vectors are indicated in the leaf nodes explicitly.
- Player nodes' pay-offs are obtained from the child node indicated by the strategy profile.
- For nodes with chance moves the pay-off vectors are obtained as a weighted sum of the child-nodes, with the probabilities used as the weights.

	CG	CH	DG	DH
A	2,1	2,1	1,4	1,4
B	1,2	2,1	1,2	2,1

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

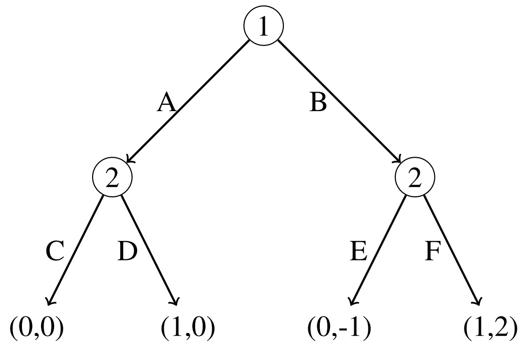
Logical equivalences that are sometimes useful about reasoning about logical formulas are listed in Table 1.

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 $v \models \phi$.



	CE	CF	DE	DF
A	0,0	0,0	1,0	1,0
B	0,-1	1,2	0,-1	1,2

Figure 4: A game in extensive form and a corresponding game in normal form

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.

C Extensive Form Games in Game Theory

Games in normal form abstract away much of the features of real-world games, including the different states a game can be in during its execution, and the observability of the state of the game. *Extensive form* of games makes these aspects of a game explicit. Essentially, a game in extensive form is a game-tree in which the different stages of a sequential game are made explicit, with players strategies decomposed to the individual decisions the players make at different stages, and the information available to the players is made explicit. Games in extensive form can also include random moves that the players cannot control.

A simple game in extensive form and a corresponding normal form game are depicted in Figure 4.

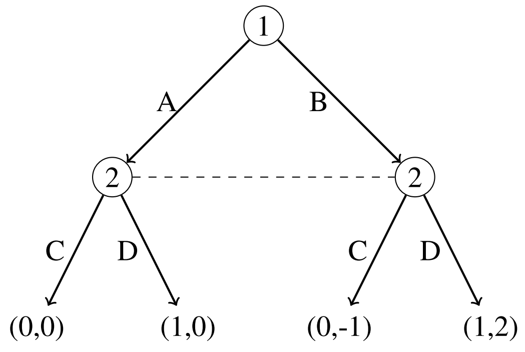
Note that what is a single strategy in the normal form game, for example CE, corresponds to two separate action choices in the extensive form game. Further, depending on which strategy the row player plays, some choices in the strategy for the column player are irrelevant. For example, if the first player plays A, the choice between E and F has no impact. Naturally in this case, the outcome for both strategy profiles (A,CE) and (A,CF) is the same, $(0, 0)$.

C.1 Games with Imperfect Information

In the extensive game in Figure 4 the second player can condition his move based on the action the first player chose. So the state of the game after the first move was observable to the second player. This is not so in all games, which might be only partially observable.

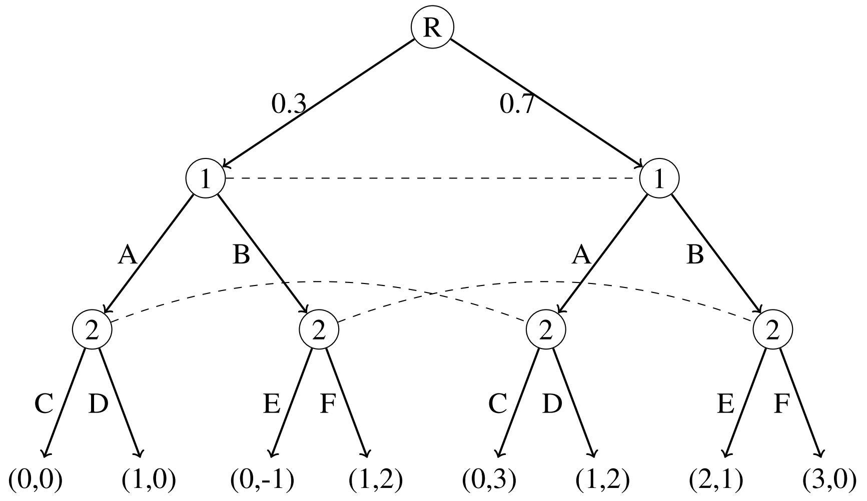
The game in Figure 5 demonstrates partial observability or *imperfect information*. The first player can choose between actions A and B, but the second player cannot observe that action, and therefore cannot use different actions as a response to the first player's different actions. The indistinguishability of the two nodes following the first player's action is indicated by the dotted line between the nodes. For two nodes connected by a dotted line the actions available to the player have to be the same, as the player has no way of distinguishing between the nodes. As you can see from the corresponding normal form game, the second player in this case has only two strategies, because the only choice the player has is between actions C and D, and the same choice applies to both of the player's nodes, irrespective of the preceding action by the first player. In contrast, in the game in Figure 4, the player could condition the actions on the first player's action, and hence there were two independent action choices for the two nodes.

Since the second player cannot distinguish between the moves of the first player (because of observability), and the first player cannot distinguish between the moves of the second player (because the second player's turn is *after* the first player), the action choices for the two players are independent of each other. This means that the extensive form game could be re-structured so that player 2 would move first, and player 1 would move after that, with the two nodes for player 1 being observationally indistinguishable. That game would have exactly the same representation as a normal form game



	C	D
A	0,0	1,0
B	0,-1	1,2

Figure 5: An imperfect information game and a corresponding game in normal form



	CE	CF	DE	DF
A	0,2.1	0,2.1	1,1.4	1,1.4
B	1.4,0.4	1.7,1.3	2.1,-0.3	2.4,0.6

Figure 6: An imperfect information game with chance moves and a corresponding game in normal form

(the one already shown in Figure 5). This is how one can also formalize any game in which two or more players choose their moves *simultaneously*.

C.2 Games with Randomness

Another aspect of many real-world situations that can be made explicit in games in extensive form is *randomness*. Part of a game may depend on events and circumstances that are outside the control of the players. This can be formalized as random *chance moves*, which determine the next state of the game only with certain probabilities.

As an example of a game with chance moves, consider the game in Figure 6. There is only one chance move in this game, as the first move in the root node. The randomness in this node can play out in two different ways, respectively with probabilities 0.3 and 0.7, leading to two alternative sub-games.

The two sub-games after the chance move have the same actions for all players. The only difference in the two sub-games is the payoff vectors in the leaf nodes. The two players cannot tell which sub-game they are playing, as indicated by the dashed lines.

After the chance move, the leftmost sub-game is as in Figure 4, with the corresponding normal form game.

	CE	CF	DE	DF
A	0,0	0,0	1,0	1,0
B	0,-1	1,2	0,-1	1,2

The rightmost sub-game corresponds to the following normal form game.

	CE	CF	DE	DF
A	0,3	0,3	1,2	1,2
B	2,1	2,1	3,0	3,0

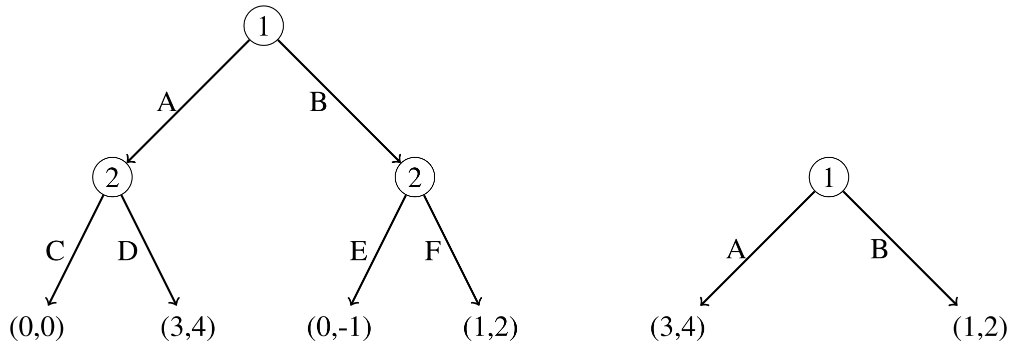


Figure 7: Backward induction

Now, the whole game, which starts with the chance move, corresponds to a normal form game in which the payoffs are obtained as a weighted sum of the payoffs of the two sub-games, with the probabilities as the weight. These payoffs are the expected pay-offs, considering the uncertainty about how the chance move plays out.

	CE	CF	DE	DF
A	0,2.1	0,2.1	1,1.4	1, 1.4
B	1.4,0.4	1.7,1.3	2.1,-0.3	2.4,0.6

Extensive games, also including imperfect information and chance moves, can always be reduced to games in normal form, and solved by using the applicable methods. The main obstacle for using this reduction is the very large size of the normal form games obtained as a result. That's why more specialized methods have been developed for extensive form games, especially in special cases in which the solutions (Nash equilibria) are simpler than in the general case.

C.3 Backward Induction

Minimaxing as in computer game-playing for zero-sum games can be applied to some extensive form games. For minimaxing, known as *backward induction* in game-theory, is not in general applicable to games with imperfect information, and not even in all games with perfect information, when an agent has to choose between two or more actions that all would yield the same pay-off.

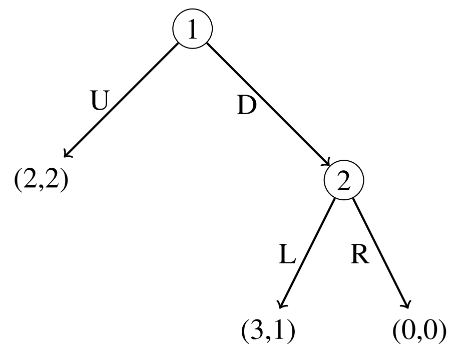
Consider the extensive form game on the left in Figure 7. Look at the choices of Player 2. In the leftmost node for Player 2, there is the choice between C and D, with reward vectors $(0,0)$ and $(3,4)$, respectively giving the pay-offs 0 and 4 for Player 2. Clearly, focusing on this sub-game only, Player 2 would choose to play D. Similarly, the choice between actions E and F is easy, as E yields -1 and F yields 2, so action F is the rational choice in that node.

Assuming that Player 2 plays his way, the game can be simplified to the one depicted on the right in Figure 7. Now same reasoning can be be applied to Player 1, resulting in the pay-off vector $(3,4)$.

Note that backward induction would not yield a unique outcome for a game if one of the players at some stage had choice between two actions that yield the same pay-off for that player. Also, if there is imperfect information, so that a player does not unambiguously know which node is currently being played, backward induction would not determine which action to choose.

Backward induction, when it is applicable, can also demonstrate that the reduction to normal form games loses information that may be critical in determining what happens when a game is played sequentially. Consider the extensive game in Figure 8 and the corresponding game in normal form. This game has the obvious Nash equilibrium D, L with pay-offs $(3,1)$. Interestingly, there is another pure-strategy Nash equilibrium U, R, with pay-offs $(2,2)$, obvious from the normal form game. However, it is clear that player 2 would never play R, because R yields $(0,0)$ and L yields $(3,1)$: so if we only look at the sub-game with player 2's moves, then L is the only possible move. And given that player 2 always plays L, player 1 would necessarily play D. So, looking at the sequential games (and not the corresponding normal form game), D, L is the only plausible Nash equilibrium this game has.

In this case, the difference between the normal form game and the extensive form game is that in the former, the players "commit" to a given strategy, and the strategies yield certain outcomes, without it being possible to observe what is happening during the "execution" of the game. In the extensive form game the situation is different: a certain strategy can be chosen, like player 2 choosing strategy R, but during the execution it may turn out that an action in that strategy is



	L	R
U	2,2	2,2
D	3,1	0,0

Figure 8: An extensive game with an odd Nash equilibrium

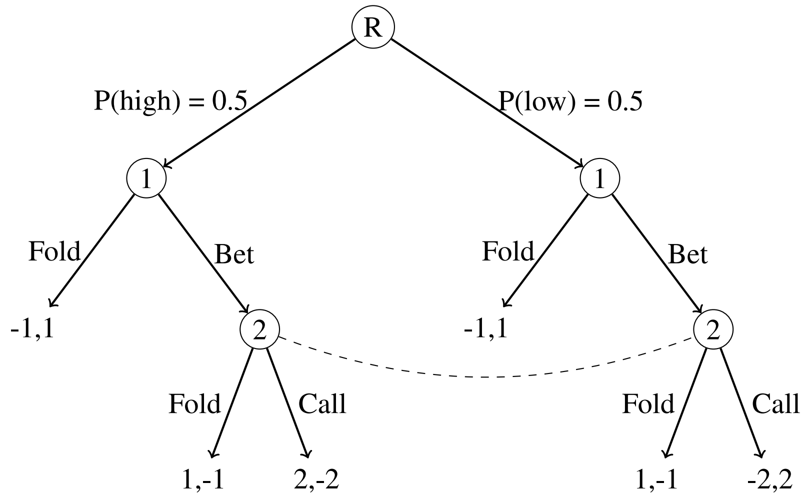


Figure 9: A simplified Poker game

not the best possible one, when the execution actually reaches a stage where that action becomes possible. The essential thing in this example is that in the strategy profile U, R the choice between R and L is irrelevant, as player 1's choice of U makes is impossible to reach the node in which R or L could be executed.

The term *sub-game perfect equilibrium* was introduced to characterize Nash equilibria for extensive form games in which every agent's behavior is optimal also when only considered in a subtree of the whole game-tree that forms the extensive form game. In the above example, the Nash equilibrium D, L is sub-game perfect, but the Nash equilibrium U, R is not.

C.4 Example: Poker

We illustrate concepts like imperfect information and chance moves with a simple "real-world" example.

Games like Poker can be analyzed by representing their important aspects in terms of an extensive form game. The shuffling of the deck of cards can be represented as chance moves, and the fact that the contents of the deck and the cards of the other players cannot be directly observed, is a form of imperfect information as understood in extensive form games.

A very simple modeling of uncertainty, observability and bidding in Poker is given in Figure 9. Modeling a full-scale Poker game as an extensive game is not practical due to the astronomic number of states and gameplays. In this simplified version both players bet 1 unit of money, only one card is dealt to the first player, the first player bets one unit more money to stay in the game, or folds (exits the game, losing his initial bet to the second player), and then the second player either folds (losing his initial bet), or calls (by betting more money) to see if the card the first player holds has a high value. If it is, then the first player wins all money, and otherwise the second player wins.

Note that this game cannot be solved with backward induction. While for the two nodes for player 2 it is clear that on the left, player 2 should choose *Fold*, and on the right, player 2 should choose *Call*, player 2 never knows which node is being played, and hence the same action must be chosen in both. Therefore, what is left is to reduce the game to a normal form game, and solve it through methods for normal form games.

This game, while too simple to be exciting to play for fun, is sufficiently complex to demonstrate that bluffing (betting more money even when the cards are bad) is a part of the optimal strategy in Poker. Specifically, any deterministic strategy for the first player (betting more money exactly when the card is of high value) is not optimal.

The optimal strategy for the first player is a mixed strategy, with randomly betting more money when the card is bad. The required probability can be determined by constructing the equivalent normal form game and finding its Nash equilibrium.