Exercise 1
For each of the following, give a proof or a counterexample: for all sets A, B, C, it holds that...

a) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
b) if $A \subseteq C$, $B \subseteq C$, and $x \in A$, then $x \in B$.
c) if F is a set of sets and $\bigcup F \subseteq A$, then $F \subseteq \mathcal{P}(A)$.
d) if $A \cap C \subseteq B$ and $a \in C$, then $a \notin A \setminus B$.
Hint: It may help to rephrase the goal to 'for any a, if $a \in A$, then $a \in B$'.

Solution:

a) True. If $x \in A \cup (B \cap C)$, then either $x \in A$ or ($x \in B$ and $x \in C$). In both cases, we have $x \in A \cup B$ and $x \in A \cup C$ and thus $x \in (A \cup B) \cap (A \cap C)$.

If on the other hand $x \in (A \cup B) \cap (A \cup C)$, then we have $x \in A \cup B$ and $x \in A \cup C$. We distinguish two cases:

- If $x \in A$, then obviously $x \in A \cup (B \cap C)$ and we are done.
- If $x \notin A$, then $x \in A \cup B$ implies $x \in B$ and $x \in A \cup C$ implies $x \in C$. Thus we have $x \in B \cap C$ and thus $x \in A \cup (B \cap C)$.

b) False. E.g. $A = \{1, 2\}, B = \{3, 4\}, C = \{1, 2, 3, 4\}$, and $x = 1$

c) True. We need to show that $F \subseteq \mathcal{P}(A)$.

To prove that some set is a subset of another set, we need to show that any element of the former is also an element of the latter. Thus, let X be an arbitrary element of F (note: X is a set). Our goal is to show that $X \in \mathcal{P}(A)$. By definition, $X \in \mathcal{P}(A)$ is equivalent to $X \subseteq A$.

In order to show $X \subseteq A$, let y be an arbitrary element of X. Since $y \in X$ and $X \in F$, clearly $y \in \bigcup F$ by definition of $\bigcup$. Since $\bigcup F \subseteq A$, we then also have $y \in A$ and we are done.

d) True. We distinguish two cases:

- Suppose $a \in A$. Then since $a \in C$ we have $a \in A \cap C$. But because $A \cap C \subseteq B$, it then follows that $a \in B$ and therefore $a \notin A \setminus B$.
- Suppose $a \notin A$. Then by definition we also have $a \notin A \setminus B$.

Exercise 2
Prove the following statements using induction. Explicitly state your base case, induction hypothesis, induction step, and where you use the induction hypothesis in the induction step.

a) $2^n \geq n + 1$ for every $n \in \mathbb{N}$
b) $2^n > n^2$ for every $n \in \mathbb{N}$ with $n \geq 5$

Hint: In b), you will need to do a modified induction with the base case $n = 5$.

Solution:

a) We prove that $2^n \geq n + 1$ for any $n \in \mathbb{N}$ by mathematical induction on n.
Base case: We have to prove $2^0 \geq 0 + 1$, which is obviously true.

Induction step: Let $n \in \mathbb{N}$ be arbitrary and assume as IH (induction hypothesis): $2^n \geq n+1$
Our goal is to show $2^{n+1} \geq n+1+1 = n+2$. We have:

$$2^{n+1} = 2 \cdot 2^n \overset{\text{IH}}{\geq} 2 \cdot (n+1) = 2n+2 \geq n+2$$

This concludes the proof.

b) We prove that $2^n > n^2$ for any $n \in \mathbb{N}$ with $n \geq 5$ by mathematical induction on n (starting at $n = 5$ instead of the usual 0).
Base case: We have to prove $2^5 > 5^2$, which simplifies to $32 > 25$.
Induction step: Let $n \in \mathbb{N}$ be arbitrary with $n \geq 5$ and assume as IH: $2^n > n^2$
Our goal is to show $2^{n+1} > (n+1)^2$. To make matters somewhat easier, we rewrite the goal into the equivalent form $2^{n+1} - (n+1)^2 > 0$. We now have:

$$\begin{aligned} 2^{n+1} - (n+1)^2 &= 2 \cdot 2^n - n^2 - 2n - 1 && (1)\\ &\overset{\text{IH}}{>} 2n^2 - n^2 - 2n - 1 && (2)\\ &= n^2 - 2n - 1 && (3)\\ &\geq 5n - 2n - 1 && (4)\\ &= 3n - 1 \geq 15 - 1 > 0 && (5) \end{aligned}$$

This concludes the proof. Note that steps (4) and (5) we used the assumption $n \geq 5$.

Exercise 3
Consider tuples of the form $(x_1, \ldots, x_n) \in \{0,1,2\}^n$, i.e. lists of some length n where each list element is 0, 1, or 2. We now look at the set A_n of these lists of length n where every 1 and 2 is immediately followed by a 0.

For ease of notation, we drop all commas and parentheses and simply write the list $(a_1, \ldots, a_n)$ as $a_1 \ldots a_n$, e.g. 0120 instead of $(0,1,2,0)$. The empty list (), i.e. the only list of length 0, will be written as ε and a list of length 1 like (0) simply as 0.

Examples: $A_0 = \{\varepsilon\}$, $A_1 = \{0\}$, $A_2 = \{00, 10, 20\}$, $A_3 = \{000, 010, 020, 100, 200\}$

a) Convert the above informal description of A_n into a formal one using the set-builder notation from the lecture: $\{(x_1, \ldots, x_n) \in \{0,1,2\}^n \mid \textit{some property involving } x_1 \textit{ to } x_n\}$.
b) Write a Haskell program[1] to compute A_n and, from that, $f_n = |A_n|$. Use it to find f_0 to f_{10}.
c) Take a look at the values of f_n you have computed so far. Can you find any pattern? Can you conjecture a closed-form formula for f_n?

Solution:

a)

$$A_n = \{(x_1, \ldots, x_n) \in \{0,1,2\}^n \mid \text{for all } i \in [n] : \text{ if } x_i \in \{1,2\} \text{ then } i < n \text{ and } x_{i+1} = 0\}$$

Or with more logical syntax:

$$A_n = \{(x_1, \ldots, x_n) \in \{0,1,2\}^n \mid \forall i \in [n](x_i \in \{1,2\} \to i < n \wedge x_{i+1} = 0)\}$$

[1] Or some other programming language of your choice, but we do recommend Haskell.

b)
```
import Control.Monad

check :: [Int] -> Bool
check []           = True
check (1 : 0 : xs) = check xs
check (2 : 0 : xs) = check xs
check (0 : xs)     = check xs
check _            = False

-- A less efficient, but more direct version
-- (note that the indices here run from 0 to n-1, not from 1 to n)
check' xs = all (\i -> xs !! i == 0 || (i < n-1 && xs !! (i+1) == 0)) [0..n-1]
  where n = length xs

f :: Int -> Int
f n = length (filter check (replicateM n [0, 1, 2]))
```

This gives us the following values for f_n:

n	0	1	2	3	4	5	6	7	8	9	10
f_n	1	1	3	5	11	21	43	85	171	341	683

c) Following the hint, we obtain:

n	0	1	2	3	4	5	6	7	8	9	10
$3f_n$	3	3	9	15	33	63	129	255	513	1023	2049

All the numbers are *almost* powers of two. More precisely, they alternate between being one greater than 2^{n+1} and one less than 2^{n+1}. That gives us the following formula:

$$f_n = \left\{ \begin{array}{ll} \frac{1}{3}(2^{n+1}+1) & \text{if } n \text{ even} \\ \frac{1}{3}(2^{n+1}-1) & \text{if } n \text{ odd} \end{array} \right\} = \tfrac{1}{3}(2^{n+1} + (-1)^n)$$

Bonus exercise

a) Try to find a *recurrence relation* (that is, a recursive equation) for f_n from Exercise 3. E.g. the well-known *Fibonacci numbers* satisfy the recurrence $h_n = h_{n-1} + h_{n-2}$ for $n \geq 2$.

b) Use the recursive equation from a) to write a more efficient program to compute f_n. Use it to find f_n for n up to 25. Can your program compute f_{100}? What about $f_{10{,}000}$?

c) Use any of the information you gathered so far and any resources at your disposal (books, internet, etc.) to find out more about f_n. Does it have a name? Can you find a closed formula?

d) Test the closed formula you found/conjectured for f_n with your program. Do you have any idea how you could prove the correctness of the formula?

Solution:

a) Consider a list w from A_n with $n \geq 2$. If the list starts with 0, the rest of the list can be anything from A_{n-1}. If the list starts with 1 or 2, the next number has to be 0, but the remaining list can again be anything from A_{n-2}. We thus get the following set recurrence:

$$A_n = \{0w \mid w \in A_{n-1}\} \cup \{10w \mid w \in A_{n-2}\} \cup \{20w \mid w \in A_{n-2}\} \qquad \text{for } n \geq 2$$

The base cases $n = 0$ and $n = 1$ are not covered by this recurrence, but it is easy to see that $A_0 = \{\varepsilon\}$ and $A_1 = \{0\}$. [2]

Since all three sets that we are taking the union of here are disjoint, this gives us following the recurrence for f_n:

$$f_0 = f_1 = 1 \qquad \text{and} \qquad f_n = f_{n-1} + 2f_{n-2} \quad \text{for } n \geq 2$$

b) A naïve implementation would be this:

```
f :: Int -> Integer
f 0 = 1
f 1 = 1
f n = f (n - 1) + 2 * f (n - 2)
```

This implementation is however not efficient enough for larger inputs such as $n = 100$. This is due to the fact that each recursion step produces two new recursive calls, leading to an infinite recursion tree. Instead, it makes sense to remember the last two values we computed in an accumulator, i.e. to start with the pair $(f_0, f_1) = (1, 1)$ and then transform (f_i, f_{i+1}) to (f_{i+1}, f_{i+2}) using the recurrence repeatedly until we have the value we want.

```
f :: Int -> Integer
f n = aux (1, 1) n
  where aux (acc1, acc2) n =
          if n == 0 then acc1 else aux (acc2, 2 * acc1 + acc2) (n - 1)
```

An imperative implementation using a `for` loop in e.g. C/Python/Java achieves a similar running time. Note however that it is crucial to use arbitary-precision integers here (e.g. `Integer` instead of `Int` in Haskell or `BigInteger` instead of `int` in Java) because the numbers quickly grow beyond the capacity of machine-width integers.

In Python 3, one could e.g. do this:

```
def f(n):
    (x, y) = (1, 1)
    for i in range(0, n): (x, y) = (y, 2 * x + y)
    return x
```

c) Typing the values into the OEIS tells us that f_n is the sequence of *Jacobsthal numbers* shifted by one, i.e. f_n is the $(n+1)$-th Jacobsthal number. Since the n-th Jacobsthal number has the closed-form formula $\frac{1}{3}(2^n - (-1)^n)$, our f_n has the closed-form formula

$$f_n = \tfrac{1}{3}(2^{n+1} + (-1)^n) \ ,$$

which agrees with what we found in Exercise 3.

This solution can also be found by typing the recurrence `a(0) = 1, a(1) = 1, a(n) = a(n-1) + 2 a(n-2)` into WolframAlpha, or by using the `RSolve` function of Mathematica or one of its equivalents in another computer algebra system.

d) We can test our formula from c) or Exercise 3 e.g. for all n up to 100 like this:

```
all (\n -> 3 * f n == 2 ^ (n + 1) + (-1) ^ n) [0..100]
```

It evaluates to `True`, so at least there is no counterexample for $n \leq 100$. An alternative solution using arbitrary-precision rational numbers would be this:

[2] Alternatively, one could have also extended the validity of the recurrence to $n \geq 1$ with the convention that $A_{-1} = \emptyset$, since there are no lists of length -1.

```
all (\n -> fromIntegral (f n) == (2^(n+1) + (-1)^n :: Rational) / 3) [0..100]
```

Note that using floating-point numbers here is not ideal since then the equality test will not be exact due to rounding errors. Especially for larger integers (the numbers here grow fast!), the limited precision of floating-point numbers becomes a big issue.

As for the proof, let us define $g_n = \frac{1}{3}(2^{n+1} + (-1)^n)$. It is easy to check that $g_0 = g_1 = 1$ and $g_n = g_{n-1} + 2g_{n-2}$ for all $n \geq 2$. Since this is exactly the same recurrence as the one we got for f_n in b), we have $f_n = g_n$ for all n.

Proving this can be done with a simple induction. However, we need a special kind of induction that we have not seen in the lecture so far: We have two base cases $n = 0$ and $n = 1$ and in the induction step we have two induction hypotheses. The basic proof structure is like this:

- $f_0 = g_0$
- $f_1 = g_1$
- $f_{n-2} = g_{n-2}$ and $f_{n-1} = g_{n-1}$ imply $f_n = g_n$ for any $n \geq 2$

We will see later in the lecture why this is a valid kind of induction. Alternatively, we can also use *strong induction*, which we will learn about in the Week 3 lecture.