Symbolic Methods for AI
–
SS 2025
–
Homework Assignments

Michael Kohlhase
Informatik, FAU Erlangen-Nürnberg

2026-02-02

Contents

1 Talking about Math

Problem 1.1 (Find Aggregations)

Open your favorite math or computer science textbook and find some of the phenomena of mathematical vernacular that we have covered in the lecture. Submit the text, author name and title of the book as well as the page number for the source.

Objective: understand mathematical vernacular

Hint: URLs to fragments of online textbooks are perfectly acceptable.

---

1. Aggregation of statements and objects.

   2 pt

   Objective: apply mathematical vernacular

   Solution: no master solution

   ---

2. An argument sequence to a flexary operator.

   2 pt

   Objective: apply argument sequence

   Objective: apply flexary

   Solution: no master solution

   ---

3. A theorem, lemma, and corollary

   2 pt

   Objective: apply theorem

   Objective: apply lemma

   Objective: apply corollary

   Solution: no master solution

   ---

4. Occurrences of "*iff*", "*hence*", and "*thus*".

   2 pt

   Solution: no master solution

   ---

5. A sequence of three declarations; specify the scope.

   2 pt

   Objective: apply declaration

   Solution: no master solution

Problem 1.2 (Adding two numbers)

2 pt

Objective: apply defining equation

Remember the two defining equations for addition on unary natural numbers:

- $n \oplus o = n$
- $m \oplus s(n) = s(m \oplus n)$

Determine the unary natural number represented by $s(s(o))\oplus s(s(s(o)))$ by repeatedly applying the given equations. Don't skip intermediate steps.

Solution:

$$\begin{aligned} s(s(o))\oplus s(s(s(o))) &= s(s(s(o))\oplus s(s(o))) \\ &= s(s(s(s(o))\oplus s(o))) \\ &= s(s(s(s(s(o))\oplus o))) \\ &= s(s(s(s(s(o))))) \end{aligned}$$

Problem 1.3 (Unary Natural Numbers Practice) 30 pt

Objective: understand Peano axiom

Let's redefine the successor function for unary natural numbers as follows:
$s(0) := 1$ and $s(n) := n^2 - 3n + 2$ for $n > 0$
Using this definition, write down the set of unary natural numbers.
Do the first four Peano axioms still hold? If not, which one fails and why?

Solution:

- The set is: $\{0, 1\}$.
- **P1:** YES (0 is a unary natural number)
- **P2:** YES ($s(0) = 1,\ s(1) = 0$)
- **P3:** NO ($s(1) = 0$)
- **P4:** YES ($s(0) \neq 1s(1)$)

Problem 1.4 (Greatest Common Divisor)

Objective: apply defining equation

Provide a definition for the function that finds the greatest common divisor of two unary natural numbers, considering the following inductive definitions of addition and multiplication on unary natural numbers. You may also create any other functions you may wish to use.

1. addition: $\alpha(n, o) = n$ and $\alpha(n, s(o)) = s(\alpha(n, m))$
2. multiplication: $\mu(n, o) = o$ and $\mu(n, s(m)) = \alpha(n, \mu(n, m))$

Hint: It is handy to use the Euclidean algorithm here. Recall:

- $gcd(n, 0) = n$
- $gcd(n, m) = gcd(m, n \ mod \ m)$

Create your own auxiliary functions on unary natural numbers and, using them, convert this definition of the greatest common divisor algorithm on natural numbers into one on the unary natural numbers.

Solution: To convert the formulae for gcd to formulae on unary natural numbers, we need the definition of modulus.

The modulus $n \ mod \ m$ equals $(n - m) \ mod \ m$ in case that $n \geq m$ and returns n otherwise. The subtraction function always returns o if $n < m$, so we can make use of this fact. We will do that by applying a Boolean function. Since the modulus function can return a value when it has reached a state when $n < m$, we can set the Boolean function to true ($s(o)$), and return n. Before that, while we are still in a state when the Boolean function returns false (o), we can just calculate $(n - m) \ mod \ m$. To include the effects of the boolean and the subtraction function, without influencing the output, we will be adding their product to $(n - m) \ mod \ m$. This does not change the output, because in case $n < m$, we add $(n - m)$ to zero (which is the remainder of zero divided by anything), and otherwise we add zero (because the boolean evaluates to zero) to $(n - m) \ mod \ m$.

For subtraction we will use the predecessor function, i.e. which does the opposite of the successor function.

This is sufficient to convert the whole greatest common divisor formula to one that works on the unary natural numbers.

Here are the inductive definitions:

1. previous element:
   - $\pi(o) = o$
   - $\pi(s(n)) = n$
2. subtraction (which returns o if the $m > n$ in $m - n$):
   - $\delta(o, m) = o$
   - $\delta(n, o) = n$
   - $\delta(n, s(m)) = \pi(\delta(n, m))$
3. boolean:
   - $\beta(o) = o$
   - $\beta(s(n)) = s(o)$
4. modulus:
   - $\varphi(o, m) = o$

- $\varphi(n,m) = \alpha(\varphi(\alpha(n,m),m),\mu(\alpha(n,m),\beta(\delta(m,n))))$

5. greatest common divisor:
   - $\gamma(n,o) = n$
   - $\gamma(n,m) = \gamma(m,\varphi(n,m))$

Problem 1.5 (Unary Natural Numbers)

Objective: apply mathematical induction

Objective: apply defining equation

Let $\oplus$ be the addition operation and $\odot$ be the multiplication operation on unary natural numbers as defined on the slides.

1. Prove or refute that $a \oplus b = b \oplus a$ 0 pt

Solution:

Proof: We proceed by induction over b:

1. Base case: $a \oplus 0 = 0 \oplus a$
   1.1. We have $a \oplus 0 = a$ by the addition axiom, so we need $a = 0 \oplus a$. This is easily proven by induction over a.
3. Step case:
   3.1. We first prove by induction over a that $s(b) \oplus a = b \oplus s(a)$
      3.1.1. Base case: $a = 0$
      We need $s(b) \oplus 0 = b \oplus s(0)$. But from the previous step we know:

$$s(b) \oplus 0 = 0 \oplus s(b) = s(b \oplus 0) = b \oplus s(0)$$

      3.1.3. Step case
         3.1.3.1. assume $s(b) \oplus a = b \oplus s(a)$.
         3.1.3.2. Now:

$$s(b) \oplus s(a) = s(s(b) \oplus a) = s(b \oplus s(a)) = b \oplus s(s(a))$$

   3.3. Back to the problem, assume $a \oplus b = b \oplus a$ and let's prove that $a \oplus s(b) = s(b) \oplus a$.
      3.3.1. We have $a \oplus s(b) = s(a \oplus b) = s(b \oplus a) = b \oplus s(a)$.
      3.3.2. This is equal to $s(b) \oplus a$, which is what we need to prove.

□

---

2. Prove or refute that $(a \oplus b) \odot c = a \odot c \oplus b \odot c$ 0 pt

---

Solution:

Proof: We proceed by induction over c.

1. Base case: $(a \oplus b) \odot 0 = a \odot 0 \oplus b \odot 0$.
   1.1. This is true by the fact that $\forall n. n \odot 0 = 0$.
3. Step case
   3.1. Assume that $(a \oplus b) \odot c = a \odot c \oplus b \odot c$ and show $(a \oplus b) \odot s(c) = a \odot s(c) \oplus b \odot s(c)$
   3.2. From the multiplication axioms we have:

$$\begin{aligned}(a \oplus b) \odot s(c) &= a \oplus b \oplus (a \oplus b) \odot c \\ &= a \oplus b \oplus a \odot c \oplus b \odot c \\ &= a \oplus a \odot c \oplus b \oplus b \odot c \\ &= a \odot s(c) \oplus b \odot s(c)\end{aligned}$$

□

---

3. Prove or refute that $a \odot b = b \odot a$ 0 pt

Solution:

Proof: We proceed by induction over b.

1. Base case
   $a \odot 0 = 0 \odot a$ *can be easily proven by induciton over* a.
3. Step case
   3.1. We assume $a \odot b = b \odot a$ and show $a \odot s(b) = s(b) \odot a$.
   3.2. We have:

$$\begin{aligned}a \odot s(b) &= a \oplus a \odot b \\ &= s(0) \odot a \oplus b \odot a \\ &= (s(0) \oplus b) \odot a \\ &= (b \oplus s(0)) \odot a \\ &= s(b) \odot a\end{aligned}$$

where we used the fact that $a = s(0) \odot a$, something that can be quickly proven by induction over a.

□

Assignment1 – MathTalk & ProofTalk

Problem 1.1 (Five Djinn Postulates) 5 pt

Objective: apply MathTalk

Restate in MathTalk the following five Postulates of the Djinns:

1. Genie is a djinn.
2. Every djinn has a meta (which is also a djinn).
3. Genie is not the meta of any djinn.
4. Different djinns have different metas.
5. If Genie has X, and each djinn that has X gives X to its meta, then all djinns get X.

Solution: This actually is a restatement of Peano Axioms, which could be written as:

1. $g_0 \in D$
2. $\forall d \in D.\exists m \in D.meta(m,d)$
3. $\nexists d \in D.meta(g_0,d)$
4. $\forall a,b \in D.(a \neq b) \wedge meta(m,a) \wedge meta(n,b) \Rightarrow (m \neq n)$
5. $\forall X.X(g_0) \wedge (\forall d \in D.X(d) \wedge meta(m,d) \Rightarrow X(m)) \Rightarrow (\forall d \in D.X(d))$

Problem 1.2 (Math Talk) 3 pt

Objective: apply MathTalk

Transliterate (turn into natural language) $\forall x.\exists S.x \in S \Leftrightarrow \neg x \notin S$

Solution: For all x, there is a (set) S, such that x is an element of S, iff it is not the case that x is not an element of S.

Problem 1.3 (Mathtalk)

In the following formalization tasks do not use anything else than the elementary set constructors and operattions and natural number arithmetics; i.e. addition, multiplication, and ordering.

1. Write down the set of even numbers between 1 and 999 using MathTalk. 2 pt

Objective: apply MathTalk

Solution: $\{x \in \mathbb{N} \mid 1 \leq x \leq 999 \wedge (\exists y \in \mathbb{N}. x = 2y)\}$

2. Give a definition of the set of prime numbers using MathTalk. 2 pt

Objective: apply MathTalk

Solution: $\{x \in \mathbb{N} \mid x > 1 \wedge (\nexists a \in \mathbb{N}. \exists b \in \mathbb{N}. 1 < a < x \wedge 1 < b < x \wedge x = ab)\}$

3. Prove from the definition of $\cap$ or refute by giving a counterexample 3 pt

Objective: apply proof

$$A \cap (B \cup C) = (A \cup B) \cap C$$

Solution: Counterexample: Let $A = \emptyset, B = \{1\}, C = \{1\}$. Then $A \cap (B \cup C) = \emptyset$, while $(A \cup B) \cap C = \{1\}$.

4. Let $A = \{1, 2, 3, 4, 5\}$. Which elements are in the set $\{x \in A \mid x \leq 4 \Rightarrow x^2 < \#(A)\}$ 1 pt

Objective: apply rsetst

Solution: $\{1, 2, 5\}$

Problem 1.4 (Math Talk) 5 pt

Objective: apply MathTalk

Write the following statement in MathTalk

> For all mandatory courses, there is a student such that if the student is from the major in which the course is mandatory then the student is not taking that course.

Hint: You may want to define short abbreviations for some of the concepts, properties, and relationships in the statement above to make your life easier.

Solution: First we define:

$C :=$ "the set of all mandatory courses"
$M :=$ "the set of all majors"

$S :=$ "the set of all students"
$Mm := \{(x,y) \mid$ "x is a mandatory course in major y"$\}$
$Sm := \{(x,y) \mid$ "x is a student from the major y"$\}$
$Sc := \{(x,y) \mid$ "x is a student taking the course y"$\}$
With these definitions we can write the statement as: $\forall c \in C.\exists s \in S.\exists m \in M.(s,m) \in Sm \wedge (c,m) \in Cm \Rightarrow (s,c) \notin Sc$

Problem 1.5 (Set operations)

Consider the sets A, B, C, and D.

1. Prove or refute that $(A \cup B) \cap (C \cup D) = A \cap B \cup C \cap D$.

   3 pt

   Objective: apply prove
   Objective: apply refute
   Objective: apply intersection
   Objective: apply union

   Solution: no master solution yet

   ---

2. Prove or refute that $(A \cup B) \times C = A \times C \cup B \times C$, where $X \times Y$ is the Cartesian product of sets X and Y.

   3 pt

   Objective: apply prove
   Objective: apply refute
   Objective: apply Cartesian product
   Objective: apply union

   Solution: no master solution yet

Assignment2 – Sets and Functions

Problem 2.1 (Function properties)

Prove or refute the following:

1. If $f, g : \mathbb{R} \to \mathbb{R}$ are injective, then $f \circ g$ is injective. — 4 pt

Objective: apply injective

Solution:

Proof: We show that $a = b$, if $f \circ g(a) = f \circ g(b)$.

1. Let $f \circ g(a) = f \circ g(b)$
   1.1. Then $g(a) = g(b)$ as f is injective
   1.2. And so $a = b$ as g is injective
3. Thus $f \circ g$ is injective by definition.

□

---

2. If $f, g, h : \mathbb{R} \to \mathbb{R}$ and $f \circ g \circ h$ is surjective, then f, g, and h are surjective. — 4 pt

Objective: apply surjective

Solution:

Proof: We show that for all $x \in \mathbb{R}$, there is a $y \in \mathbb{R}$, such that $f \circ g \circ h(y) = x$

1. f is surjective, so there is an $a \in \mathbb{R}$, such that $f(a) = x$.
2. g is surjective, so there is an $b \in \mathbb{R}$, such that $g(b) = a$.
3. h is surjective, so there is an $c \in \mathbb{R}$, such that $h(c) = b$.
4. So $f \circ g \circ h(c) = f(g(h(c))b)a = x$.

□

---

3. If $f, g : \mathbb{Z} \to \mathbb{Z}$ are injective, $f + g$ (function multiplication: $f + g(x) = f(x) + g(x)$) is surjective. — 4 pt

Objective: apply injective

Objective: apply surjective

Solution: This is false: $f : x \mapsto x$ and $g : x \mapsto -x$ constitues a counterexample as $f + gx = 0$ for all $x \in \mathbb{Z}$.

Problem 2.2 (Associativity of Relation Composition) — 5 pt

Let R, S, and T be relations on a set M. Prove or refute that the composition operation for relations is associative, i. e. that

Objective: understand composition

Objective: understand associative

$$(T \circ S) \circ R = T \circ (S \circ R)$$

Solution:

Proof:

1. Let $(x,y) \in (T \circ S \circ R)$.
2. $\exists z_1 \in M.(x,z_1) \in R \wedge (z_1,y) \in (T \circ S)$
3. $\exists z_1, z_2 \in M.(x,z_1) \in R \wedge (z_1,z_2) \in S \wedge (z_2,y) \in T$
4. $\exists z_2 \in M.(x,z_2) \in (S \circ R) \wedge (z_2,y) \in T$
5. $(x,y) \in (T \circ S \circ R)$.

□

Problem 2.3 (Basic Ordering Relations)

Say that $a \preceq b$ if a is a divisor of b.

1. Is the relation $\preceq$ a partial order on $\mathbb{N}^+$? Justify your answers by a proof or counter-example. 4 pt

Objective: remember partial ordering

Objective: understand transitive

Objective: understand antisymmetric

Objective: understand symmetric

Solution:

Proof: To be a partial order $\preceq$ must be reflexive, antisymmetric, transitive.

1. reflexivity
   $a \preceq a$, iff $a|a$, which is always the case by definition.
3. antisymmetry
   $a|b$ and $b|a$ are simultaneously true, iff $a = b$.
5. transitivity
   It is easy to prove that if $a|b$ and $b|c$ then $a|c$.
7. This concludes the proof that all condition for partial orders are met.

□

2. Is it a total order on $\mathbb{N}^+$? Justify your answers by a proof or counter-example. 2 pt

Objective: apply linear ordering

Solution: $\preceq$ is not a total order as not all positive natural numbers are comparable: 2 is not a divisor of 3 and 3 is not a divisor of 2.

Problem 2.4 (A Necessary Condition for Bijectivity) 5 pt

Given a finite sets A and B and a function $f : A \to B$. Prove or refute that if f is bijective then $\#(A) = \#(B)$.

Objective: apply bijective

Objective: apply cardinality

Solution: If f is bijective then $\#(A) = \#(B)$.

Proof:

1. Let f be bijective, then it is total, injective and surjective.
2. So, for all $a, b \in A$ we have $f(a) \neq f(b)$ if $a \neq b$.
3. Thus f cannot identify objects in A, and we have $\#(A) \leq \#(B)$.
4. Furthermore, since f is surjective, each $b \in B$ has a f-preimage,
5. so $\#(B) \leq \#(A)$ and hence $\#(A) = \#(B)$

□

Problem 2.5 (Existence of Functions)

For each of the following functions write down the function (represented as a set of pairs (x,y)) or briefly state why such function does not exist.

Objective: understand function

1. $f : \emptyset \to \mathcal{P}(\emptyset)$, f is an injective function. — 1 pt

Objective: understand injective

Solution: $f = \emptyset$

2. $f : \emptyset \to \mathcal{P}(\emptyset)$, f is a surjective function. — 2 pt

Objective: understand surjective

Solution: Such a function does not exist: the domain is empty, which means that the element $\emptyset$ from the codomain does cannot have a pair in f, so f cannot be not surjective.

3. $f : \mathcal{P}(\mathcal{P}(\emptyset)) \to \mathcal{P}(\emptyset)$, f is an injective function. — 2 pt

Objective: understand power set

Solution: Such a function does not exist since $\#(\mathbf{dom}(f)) > \#(\mathbf{codom}(f))$, a total function f cannot be injective.

4. $f : \mathcal{P}(\emptyset) \to \emptyset$, f is a partial function. — 1 pt

Objective: understand partial function

Objective: understand power set

Solution: $f = \emptyset$

5. $f : \emptyset \to \emptyset$, f is a bijective function. — 1 pt

Objective: understand bijective

Solution: $f = \emptyset$

6. $f : \mathcal{P}(\emptyset) \to \mathcal{P}(\emptyset)$, f is a bijective function

2 pt

Objective: understand bijective

Objective: understand power set

Solution: $f = \{(\emptyset,\emptyset)\}$

Problem 2.6 (*jectivity)

6 pt

Objective: apply injective

Objective: apply surjective

Objective: apply bijective

Objective: apply inverse function

For the functions below determine whether they are injective, surjective or bijective. To prove they are not, give counterexamples. Give the inverse function if possible.

$f : \mathbb{N} \to \mathbb{N}$	$x \mapsto 2x$
$h : \{2,3,4\} \to \{8,6,4\}$	$x \mapsto 12 - 2x$
$g : \mathbb{Z} \to \mathbb{N}$	$x \mapsto \lvert x \rvert$

where $\mathbb{Z}$ is the set of all integers (positive and negative) and $|x|$ is the absolute value function.

Solution:

1. f is injective and not surjective (e.g. $3 \notin f(\mathbb{N})$), so f is not bijective
2. h is injective and surjective, thus bijective. Inverse function $6 - y/2$
3. Not injective, but surjective

Assignment3 – Equivalence Classes & Quotients

Problem 3.1 (Invariance of Equivalence Relations) 4 pt

Let A be a set and $R, S \in A^2$ be equivalence relations. Prove or refute that $R \cup S$ is an equivalence relation too.

Objective: apply equivalence relation

Solution: The claim is not valid: $R \cup S$ is reflexive, symmetric, but not transitive: if we have $(a,b) \in R$ and $(b,c) \in S$, then we do not know that $(a,c) \in (R \cup S)$.

Problem 3.2

Check whether the following are equivalence relations on $\mathbb{N}$. Justify your answers with a proof or refutation.

1. $x \sim y$ iff $x \equiv y \bmod 3$ for all $x, y \in \mathbb{N}$ 3 pt

   Objective: remember equivalence relation
   Objective: understand reflexive
   Objective: understand symmetric
   Objective: understand transitive

   Solution: $\sim$ is an equivalence relation. It is reflexive because $x \equiv x \bmod 3$) for all $x \in \mathbb{N}$. It is also symmetric because if $x \equiv y \bmod 3$ then $y \equiv x \bmod 3$. Transitivity follows from $(x \equiv y \bmod 3) \land (y \equiv z \bmod 3) \Rightarrow (x \equiv z \bmod 3)$.

   ---

2. $x \sim y$ iff $x^2 = y^2$ for all $x, y \in \mathbb{N}$ 4 pt

   Objective: remember equivalence relation
   Objective: understand reflexive
   Objective: understand symmetric
   Objective: understand transitive

   Solution: $\sim$ is an equivalence relation. $x^2 = x^2$ shows reflexivity, while $x^2 = y^2 \Rightarrow y^2 = x^2$ and $x^2 = y^2 \land y^2 = z^2 \Rightarrow x^2 = z^2$ due to positivity prove symmetry and transitivity.

   ---

3. $x \sim y$ iff $x|y$ (x is a divisor of y). 4 pt

   Objective: remember equivalence relation
   Objective: understand reflexive
   Objective: understand symmetric
   Objective: understand transitive

   Solution: $\sim$ is not an equivalence relation since it isn't symmetric. For example, $3|6$ but $\neg 6|3$.

Problem 3.3 (Equivalence Classes)

Consider the equivalence relation $R \subseteq \mathbb{R}^2$ defined by $x \, R \, y$ if $x^2 = y^2$.

1. How many members does an equivalence class $[z]_R$ have for $z \in \mathbb{R}$? 2 pt

   Objective: apply equivalence class

   Solution: For $z = 0$, $[z]_R$ has one member; for any other $z \in \mathbb{R}$, it has the

two members z and $-z$.

---

2. How are the members of $[z]_R$ arithmetically related to the other one?

2 pt

Objective: apply equivalence class

Solution: For $z \neq 0$, the members of $[z]_R$ have the same absolute value, i.e. z and $-z$.

Assignment4 – SML Basics

Problem 4.1 (Square the list) 3 pt

Objective: apply recursion

Write an SML function `squareList` that takes an `int list` and returns the list with every element squared. Do not use higher-order functions.

Example:

```
- squareList [1, 4, 3, 10];
  val it = [1, 16, 9, 100] : int list;
```

Solution:

```
fun squareList [] = []
  | squareList x::l = (x*x) :: (squareList l);
```

Problem 4.2 (List functions)

Objective: understand list

Objective: apply recursion

Implement the following functions in SML, use some simple examples to test the functionality.

1. `len(list)` of type `'a list -> int`, which computes the length of a list 2 pt

   Solution:

   ```
   fun len([]) = 0
     | len(a::b) = 1+len(b);
   ```

   ---

2. `extract(elem, list)` of type `'a * 'a list -> 'a list`, which returns a copy of `list` with all occurrences of `elem` eliminated. 3 pt

   ---

   Solution:

   ```
   fun extract(x, []) = []
     | extract(x, a::b) =
         if ( a=x )
         then extract(x,b)
         else a::extract(x,b);
   ```

   ---

3. `frequency(elem, list)` of type `'a * 'a list -> int`, which counts for how many times `elem` occurs in `list`. 3 pt

   ---

   Solution:

```
fun frequency(x, []) = 0
  | frequency(x, a::b) =
  if ( a=x )
  then 1+frequency(x,b)
  else frequency(x,b);
```

Problem 4.3 10 pt

Objective: apply recursion

Write functions to zip and unzip lists. zip will take two lists as input and return pairs of elements, one from each list, as follows: `zip [1,2,3] [0,2,4]` ⇝ `[[1,0],[2,2],[3,4]]`.

`unzip` is the inverse function, taking one list of pairs as argument and returns two separate lists. `unzip [[1,4],[2,5],[3,6]]` ⇝ `[1,2,3] [4,5,6]`.

Solution: Zipping is relatively simple, we will just define a recursive function by considering 4 cases:

```
fun zip nil nil = nil
  | zip nil l = l
  | zip l nil = l
  | zip (h::t) (k::l) = [h,k]::(zip t l)
```

Unzipping is slightly more difficult. We need selectors for the first and second elements of a two-element list over the zipped list. Since the problem is somewhat under-specified by the example, we will put the rest of the longer list into the first list. To avoid problems with the empty tails for the shorter list, we use the `mapcan` function that appends the tail lists.

```
fun mapcan(f,nil) = nil | mapcan(f,h::t) = (f h)@(mapcan(f,t))
fun unzip (l) = if (l = nil) then nil
                else [(map head l),(mapcan tail l)]
```

Problem 4.4 (Prime numbers)

Objective: apply recursion

A natural number ≥ 2 is called a prime number, if it is only divisible by 1 and itself.

1. Write a function `divisible: int * int list -> bool` that tests (using `myexists`) whether a natural number is divisible by at least one element of a list of natural numbers. 3 pt

   ---

   Solution:

   ```
   fun divisible n = myexists (fn x => n mod x = 0)
   ```

   ---

2. Write a function `nextPrime: int * int list -> int` that computes the smallest prime number $p \geq x$, given a number $x > 1$ and a list ps of prime numbers that contains all prime numbers smaller than x. 2 pt

---

Solution:

```
fun nextprime n ps = if not (divisible n ps) then n
                     else nextprime(n+1) ps
```

---

3. Write a function `primes: int -> int list` that computes the list of the first n prime numbers, given $n \in \mathbb{N}$. 3 pt

---

Solution:

```
fun primes 0 = nil
  | primes 1 = nil
  | primes n = let ps = primes (n-1)
               in ps @ [nextprime(n,ps)]
               end
```

Problem 4.5 (Higher-Order Functions)

Write three higher-order functions that take a predicate p (a function with result type `bool`) and a list l. Do not use any higher-order functions in the implementation.

Hint: If you are in need of a test predicate, you can work on a list l of `int`s and use the "even number" predicate:

```
fun even n = n mod 2 = 0;
```

---

1. `myfilter` that returns the list of all members a of l where $p(a)$ evaluates to `true`. 3 pt

---

Solution:

```
fun myfilter p nil = nil
  | myfilter p h::t =
    if (p h) then h :: myfilter p t else myfilter p t
```

Hint: For the last two, we can make use of the predefined functions `orelse` and `andalso`, which are useful abbreviations: e_1 `andalso` e_2 abbreviates `if` e_1 `then` e_2 `else false` and e_1 `orelse` e_2 abbreviates `if` e_1 `then true else` e_2.

2. `myexists` that returns `true` if there is at least one element a in l, such that $p(a)$ evaluates to `true`. 3 pt

Solution:

```
fun myexists p nil = false
  | myexists p h::t =
    if (p h) then true else myexists p t
```

or (with `andalso`):

```
fun myforall f nil = true
  | myforall f (x::xr) = f x andalso myforall f xr
```

3. `myforall` that returns `true` if $p(a)$ evaluates to `true` on all elements of l. 3 pt

Solution:

```
fun myforall p nil = true
  | myforall p h::t =
    if (p h) then myforall p t else false
```

or (with `orelse`):

```
fun myexists f nil = false
  | myexists f (x::xr) = f x orelse myexists f xr
```

Problem 4.6 (Duplicates) 6 pt

Write an SML function that removes all duplicate elements from a list. For instance

Objective: apply recursion

```
remove_duplicates([true, true, false]) = [true, false];
remove_duplicates([5,3,12,3,3,2]) = [5,3,12,2];
```

Hint: Write a helper function that removes duplicates, but remembers what it has already found in an argument.

Solution:

```
fun member(a, h::t) = a=h orelse member(a, t)
  | member(_, nil) = false;

fun helper(found_already, h::t) =
               if member(h, found_already) then
                      helper(found_already, t)
               else
                      h::helper(h::found_already, t)
  | helper(_, nil) = nil;

fun remove_duplicates(l) = helper(nil, l);
```

Problem 4.7 (Set Operations)

Objective: apply recursion
Objective: understand list

Consider integer sets in SML as lists of integers without duplicates. Implement the three functions below.

Hint: You might want to consider *member* function to solve the problem:

```
- member(2,[2,3,4]);
val it = true: bool
```

---

1. A function that returns the union of two sets of integers. It should have the following signature:

   2 pt

   Objective: apply union
   Objective: apply element

   ```
   val union = fn : int list * int list -> int list
   ```

   Example:

   ```
   - union([1,2,3],[2,3,4]);
   val it = [1,2,3,4] : int list
   ```

   Solution: The approach would be to implement a member function that checks if an element is in a set and then use it for the two functions:

   ```
   fun member (a, []) = false
     | member (a, x::xls) =
         if a=x then true
         else member(a, xls);
   fun union ([], x) = x
     | union (x::ls, y) =
         if member(x,y) then union(ls, y)
         else x::union(ls, y);
   ```

   ---

2. A function that returns the intersection of two sets. It should have the following signature:

   1 pt

   Objective: apply intersection

```
val intersect = fn : int list * int list -> int list
```

Example:

```
- intersect([2,3,1],[4,1,7,6]);
val it = [1] : int list
```

Solution:

```
fun intersect ([], _) = []
  | intersect (a::lsa, lsb) =
      if member(a, lsb)
      then a::intersect(lsa, lsb)
      else intersect(lsa, lsb);
```

---

3. A function that returns the set difference between them. It should have the following signature: 1 pt

Objective: apply set difference

```
val setdifference = fn : int list * int list -> int list
```

Example:

```
- setdifference([3,1,7,8],[2,1,5,3]);
val it = [7,8] : int list
```

Solution:

```
fun setdifference ([]:int list, _) = []
  | setdifference (a::lsa, lsb) =
      if member(a, lsb) then setdifference(lsa, lsb)
      else a::setdifference(lsa, lsb);
```

Problem 4.8 (Permutations) 15 pt

Objective: apply permutation

Objective: apply power

Objective: apply recursion

For a set $A := \{1, 2, \ldots, n\}$, a permutation σ is a bijective function $\sigma : A \to A$. Permutation multiplication can be defined as:

$$(\sigma \cdot \pi)(i) := \sigma(\pi(i))$$

Intuitively, we can also define the "raise to power" operation for permutations as multiplication applied on itself:

$$\sigma^k := \underbrace{\sigma \cdot \sigma \cdot \ldots \cdot \sigma}_{k \text{ times}}$$

We can also define the **identity permutation** as: $id(i) := i$. One of its important properties is that $id \cdot \sigma = \sigma \cdot id = \sigma$.

Your task is to write an SML function `findPower` that, given a permutation σ, returns the minimum power $k > 0$ for which $\sigma^k = id$. The function should have the following signature:

```
val findPower = fn : int list -> int
```

Examples:

```
- findPower [1,2,3];
val it = 1 : int
- findPower [3,1,2];
val it = 3 : int
- findPower [5,1,9,12,3,4,8,7,2,6,11,10];
val it = 20 : int
```

Hint: You might consider writing a function for permutation multiplication, and also a function for raising a permutation to a power.

Solution:

```
(* multiply two permutations *)
fun permMul [] _ = []
  | permMul (a::f) l = List.nth(l,a-1) :: (permMul f l);

(* generate an identical permutation *)
fun upto(_, 0) = []
  | upto(n, x) = n :: upto(n+1, x-1)

(* permutation to power k *)
fun permPower l 0 = upto(1, List.length(l))
  | permPower l n =
              let val x = permPower l (n div 2)
              in
                  if n mod 2=1 then
                        permMul (permMul x x) l
                    else
                        permMul x x
              end;

(* wrapper *)
fun findPower l =
       let val id = upto(1, List.length(l))
           (* test if sigma^n=1 and stop if true *)
           fun test l n =
               if (permPower l n)=id then n else (test l (n+1))
       in
               test l 1
       end;
```

Problem 4.9 (Checking relation properties)

10 pt

Objective: apply recursion

Objective: understand reflexive

Objective: understand symmetric

Objective: understand transitive

Objective: understand equivalence relation

Objective: understand linear ordering

Given a set A and a relation R on this set, implement SML functions to check whether R is reflexive, symmetric, transitive, an equivalence relation, and a linear ordering. Each function should take two arguments: a list of elements (the set A) and a list of pairs (the members of R), and should return a Boolean value.

Test examples:

```
- reflexive([1,2,3],[(1,1),(3,3),(2,2)]);
val it = true : bool
- symmetric([1,2,3],[(1,3),(3,1)]);
val it = true : bool
- transitive([1,2,3],[(1,3),(3,1)]);
val it = false : bool
- transitive([1,2,3],[(1,2),(2,3),(1,3)]);
val it = true : bool
- equivalenceRelation([1,2,3],[(1,1),(2,2),(3,3),(1,2),(2,1)]);
val it = true : bool
- linearOrder([1,2,3],[(1,2),(1,3),(2,3)]);
val it = true : bool
```

Hint: For shorter solutions, you may make use of higher-order SML functions rather than recursion everywhere. Consult `http://www.standardml.org/Basis/list.html#LIST:SIG:SPEC` for a description of a variety of functions on lists.

Solution: A couple of auxiliary functions: The function `forall` returns `true`, iff

```
fun forall(nil) = true
  | forall(true::ls) = forall(ls)
  | forall(false::ls) = false;

fun member(n,nil) = false
  | member(n,h::r) = (n = h) orelse member(n,r);

fun generatePairs(nil) = nil
  | generatePairs([a]) = nil
  | generatePairs(a::b::l) =
      [(a,b)] @ generatePairs(a::l) @ generatePairs(b::l);
```

The function `checkPairs` checks whether all the elements in a list of pairs (second argument) belong to a given set (first argument)

```
fun checkPairs(_,nil) = true
  | checkPairs(a,(x,y)::l) =
      member(x,a) andalso member(y,a) andalso checkPairs(a,l);

(* removing all occurrences of an element from a list *)
fun remove(a,nil) = nil
  | remove(a,b::l) = if a = b
                     then remove(a,l)
                     else b::remove(a,l);
```

Reflexivity: check whether each member of the first list participates in a pair of equal elements from the second one

```
fun checkReflexivePair y = fn a => (List.exists (fn x => (a,a) = x) y);
fun checkReflexiveEach(x,y) = map (checkReflexivePair(y)) x;
fun reflexive(x,y) = forall(checkReflexiveEach(x,y));
```

Symmetry: check (a,b) and (b,a) in the second list, and also element belonging to the first list

```
fun symmetric(x,nil) = true
  | symmetric(x,(a,b)::l) =
      if a =b then symmetric(x,l)
      else member(a,x) andalso member(b,x)
                       andalso (List.exists (fn f => (b,a) = f) l)
                       andalso symmetric(x,remove((b,a),l));
```

Transitivity: check (a,b), (b,c), (a,c) and belonging to first list

```
fun checkTransPair (a, _) ls (_, c) =
  List.exists (fn x => (a, c) = x) ls;
fun checkTransList ls (a, b) =
  let val bls = List.filter (fn (x, _) => x = b) ls
      val tls = map (checkTransPair (a, b) ls) bls
  in forall(tls)
  end;
fun transitive(a,ls) = checkPairs(a,ls) andalso
                       forall (map (checkTransList ls) ls);
```

`equivalenceRelation` simply combines the three functions for an equivalence relation

```
fun equivalenceRelation (x,y) =
      reflexive(x,y) andalso
      symmetric(x,y) andalso
      transitive(x,y);
```

checking for linear order by transitivity and generating all pairs and checking whether mirrorred pairs are contained in the second list.

```
fun linProperty(nil,y) = true
  | linProperty((a,b)::l,y) =
       if a = b then linProperty(l,y) else
       if ((List.exists (fn f => (a,b) = f) y) andalso
           (List.exists(fn f => (b,a) = f) y)) orelse
           ((List.exists (fn f => (a,b) = f) y) = false andalso
           (List.exists(fn f => (b,a) = f) y) = false)
       then false
       else linProperty(l,y);
fun linearOrder(x,y) = transitive(x,y) andalso linProperty(generatePairs(x),y);
```

Problem 4.10 (Truth value combinations) 10 pt

Write an SML function `combine` of type `int -> int list list` that takes an integer n and returns a list of 2^n lists of length n which represent all combinations of ones and zeros in increasing order (as in a truth table). For example

```
- combine 2;
val it = [[0,0],[0,1],[1,0],[1,1]] : int list list
```

Solution:

```
fun combine 1 = [[0],[1]]
  | combine n = if (n>0) then
      (map (fn lst => 0::lst) (combine (n-1)))
      @ (map (fn lst => 1::lst) (combine (n-1)))
    else [];
```

Assignment5 – SML Data Types

Problem 5.1 3 pt

Objective: apply ADT

Objective: apply pattern matching

Declare an SML datatype `pair` representing pairs of integers and define SML functions `fst` and `snd` where `fst` returns the first- and `snd` the second component of the pair. Give the type of the constructor of `pair` as well as of the two functions `fst` and `snd`.

Use SML syntax for the whole problem.

Solution:

```
datatype pair = pair of int * int; (* val pair = fn : int * int -> pair *)

fun fst(pair(x,_)) = x; (* val fst = fn : pair -> int *)
fun snd(pair(_,y)) = y; (* val snd = fn : pair -> int *)
```

Problem 5.2 (Member Functions)

Write three variants of the `member` function in SML, where `member(x,xs)` returns `true`, iff `x` is an element in `xs`.

1. The first variant should not use another function. 2 pt

   Objective: apply recursion

   Solution:

   ```
   fun member1 (x,nil) = false
     | member1 (x,h::t) = if (h=x) then true else member1(x,t)
   ```

   ---

2. The second variant should be non-recursive, using the function `myexists` (write that as well) that takes a predicate `p` and a list `l` as arguments and returns `true`, iff there is an element `a` in `l` such that `p(a)` evaluates to `true`. 2 pt

   Objective: understand recursion

   Solution:

   ```
   fun myexists (_,nil) = false
     | myexists (p,h::t) = p(h) orelse myexists(p,t)
   fun member2 (x,xs) = myexists (fn (y) => x=y) xs
   ```

   ---

3. The third variant should be non-recursive using the `foldl` function. 2 pt

   Objective: apply foldl

   Solution:

```
fun member3 (x,xs) = foldl (fn (y,b) => b orelse x=y) false
```

Problem 5.3 (List functions via folding)

Objective: apply foldl
Objective: apply foldr

Write the following functions using `foldl` or `foldr`.

1. `length` which computes the length of a list — 2 pt

Solution:

```
fun length xs = foldl (fn (x,n) => n+1) 0 xs
```

---

2. `concat`, which gets a list of lists and concatenates them to a list. — 2 pt

---

Solution:

```
fun concat xs = foldr op@ nil xs
```

---

3. `map`, which assigns a function over a list. — 2 pt

Solution:

```
fun map f = foldr (fn (x,yr) => (f x)::yr) nil
```

---

4. `myfilter` that returns the list of all members a of l where $p(a)$ evaluates to `true`. — 2 pt

---

Solution:

```
fun myfilter f = foldr (fn (x,ys) => if f x then x::ys else ys) nil
```

---

5. `myexists` that returns `true` if there is at least one element a in l, such that $p(a)$ evaluates to `true`. — 2 pt

---

Solution:

```
fun myexists f = foldl (fn (x,b) => b orelse f x) false
```

---

6. `myforall` that returns `true` if $p(a)$ evaluates to `true` on all elements of l. 2 pt

---

Solution:

```
fun myall f = foldl (fn (x,b) => b andalso f x) true
```

Problem 5.4

Objective: apply ADT

Objective: apply recursion

Given the following SML data type for an arithmetic expressions

```
datatype arithexp = aec of int (* 0,1,2,... *)
                  | aeadd of arithexp * arithexp (* addition *)
                  | aemul of arithexp * arithexp (* multiplication *)
                  | aesub of arithexp * arithexp (* subtraction *)
                  | aediv of arithexp * arithexp (* division *)
                  | aemod of arithexp * arithexp (* modulo *)
                  | aev of int (* variable *)
```

1. Give the representation of the expression $(4x + 5) - 3x$. 2 pt

Objective: apply ADT

Solution:

```
aesub(aeadd(aemul(aec(4),aev(1)),aec(5)),aemul(aec(3),aev(1)))
```

---

2. Write a (cascading) function `eval : (int -> int) -> arithexp -> int` that takes a variable assignment φ and an arithmetic expresson e and returns its evaluation as a value. 4 pt

Objective: apply recursion

Objective: understand evaluation

Note: A variable assignment is a function that maps variables to (integer) values, here it is represented as function φ of type `int -> int` that assigns $\varphi(n)$ to the variable `aev(`n`)`.

Solution:

```
fun eval phi =
let
    fun calc (aev(x)) = phi(x) |
        calc (aec(x)) = x |
        calc (aeadd(e1,e2)) = calc(e1) + calc(e2) |
        calc (aesub(e1,e2)) = calc(e1) - calc(e2) |
        calc (aemul(e1,e2)) = calc(e1) * calc(e2) |
        calc (aediv(e1,e2)) = calc(e1) div calc(e2) |
```

```
        calc (aemod(e1,e2)) = calc(e1) mod calc(e2);
in fn x => calc(x)
end;
```

Here is a test:

```
- eval (fn 1=>6) (aesub(aeadd(aemul(aec(4),aev(1)),aec(5)),aemul(aec(3),aev(1))));
stdIn:14.7-14.14 Warning: match nonexhaustive
1 => ...
val it = 11 : int
```

Problem 5.5 (Flip Binary Tree)

A binary tree is a relation T on $\mathbb{N}$, such that for every $n \in \mathbb{N}$ there are two or zero $m \in \mathbb{N}$, such that $(n,m) \in T$, and exactly one $r \in \mathbb{N}$, such that there is no $p \in \mathbb{N}$ with $(p,r) \in T$.

1. Provide an SML datatype declaration `btree`. 2 pt

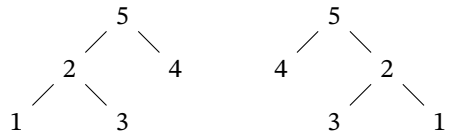
   Objective: apply ADT

   Solution: `datatype btree = leaf of int  branch of int * btree * btree;|`

   ---

2. Represent the binary trees below as SML expressions of type `btree`. 2 pt

   Objective: apply ADT



   Solution:

```
bintree1 = branch(5,(branch(2,(leaf 1),(leaf 3))),(leaf 4));
bintree2 = branch(5,(leaf 4),(branch(2,(leaf 3),(leaf 1))));
```

   ---

3. Write an SML function that takes an binary tree and returns it flipped around its vertical axis, i.e. the function transforms the left tree into the right one and the other way around. 3 pt

   Objective: apply ADT

   Objective: apply recursion

   Solution:

```
fun flip (branch (n,lb,rb)) = branch(n,flip(rb),flip(lb))
  | flip (leaf n) = (leaf n);
```

Assignment6 – Graphs & Trees

Problem 6.1 (Graph Examples)

Present a mathematical representation of the following graphs.

1. A directed graph with 4 nodes and 6 edges. 2 pt

Objective: create directed graph

Solution: Any graph will do, for instance the following one (we only give the mathematical formulation here, go draw them yourselves)
$\langle\{a,b,c,d\},\{\langle a,b\rangle,\langle b,c\rangle,\langle c,d\rangle,\langle d,a\rangle,\langle a,c\rangle,\langle b,d\rangle\}\rangle$

---

2. A undirected graph with 7 nodes and 8 edges. 2 pt

Objective: create undirected graph

Solution: Any graph will do, for instance the following one (we only give the mathematical formulation here, go draw them yourselves)
$\langle\{a,b,c,d,e,f,g\},\{\{a,b\},\{b,c\},\{c,d\},\{d,e\},\{e,f\}\{f,g\},\{g,a\},\{a,c\},\{c,e\}\}\rangle$

Problem 6.2 (In-degrees in acyclic digraphs) 8 pt

Prove by induction or refute that any acyclic graph with non-empty set of nodes has at least one node with indegree 0.

Objective: understand DAG

Objective: understand indegree

Solution:

Proof: Proof by induction on the size of (number of nodes in) the graph:

1. n=1
   1.1. trivial
3. Step case: $n \Longrightarrow n+1$
   3.1. Note that for any acyclic digraph, every subgraph is also acyclic (otherwise a cycle in the subgraph would make the entire graph cyclic). This is what makes induction possible.
   3.2. So assume an acyclic digraph with n nodes and at least a node with indegree 0 such that the graph is still acyclic. Call this node v_0. Now add an extra vertex, v_n, to the graph and any number of edges that connects it with the rest of the vertices.
   3.3. This new node has no incoming edge.
      3.3.1. This is the new 0 indegree node.
   3.5. v_0 still has no incoming edge.
      3.5.1. v_0 remains the 0 indegree node.
   3.7. v_0 has an incoming edge from v_n and $\text{indeg}(v_n) > 0$

3.7.1. Then there exists an edge from v_i to v_n. If v_i has indegree 0, then this is a 0 indegree node.

3.7.2. Otherwise, since the graph is acyclic, v_i has no incoming edge from v_0 or v_n, so there is an edge from another node in the graph.

3.7.3. Continuing this process either leads to identifying a 0 indegree node or to the situation where a last "untouched" node is reached – but the graph is acyclic, so this node has to have indegree 0.

□

Problem 6.3 (Graphs and trees)

Given the following directed graph G:

$$G = \langle \{a,b,c,d,e,f,g\}, \{(b,a),(a,d),(b,c),(c,d),(d,e),(e,d),(d,g),(c,f),(e,f)\} \rangle$$

1. Find a tree T that contains all the vertices of G as nodes and its set of edges is contained in the set of edges of G. — 3 pt

   Objective: apply tree

   Solution: $T = \langle \{a,b,c,d,e,f,g\}, \{(b,a),(a,d),(b,c),(d,e),(d,g),(c,f)\} \rangle$

2. Explain your choice of the root. — 2 pt

   Objective: understand root

   Solution: B since it's a initial node (source), so we can't place it in the tree as a child of something because it has no parent.

3. Name two difference between graphs and trees. — 2 pt

   Objective: understand graph

   Objective: understand tree

   Solution: Trees are graphs with a single root, but without cycles.

Problem 6.4 (Tree representations) — 5 pt

Objective: apply parent

A tree can be represented in many different ways. One method is to specify the parent for each node. For example in a table:

0	1	2	3	4	5	6	7
-1	0	3	0	1	1	3	3

Here having a parent -1 represents a root.

Another representation would be the following SML datatype:

```
datatype tree = node of int * tree list;
```

Write an SML function `convert` that takes a tree, represented as an `(int * int) list` and converts it to the datatype representation.

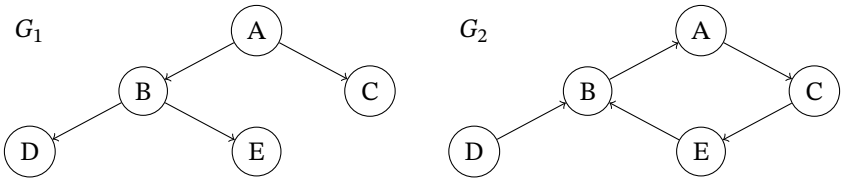
Signature and example:

```
val convert = fn : int list -> tree

- convert([1,~1,1,2]);
val it = node (1,[node (0,[]),node (2,[node (3,[])])]) : tree
```

Problem 6.5 (Number of Paths)

Consider the following two directed graphs:



1. Represent them as mathematical structures $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$. 4 pt
   Objective: apply directed graph
2. Give set $\Pi(G_1)$ of all paths in G_2.

   Solution: $\Pi(G_1) = \{\langle A, B\rangle, \langle A, C\rangle, \langle B, D\rangle, \langle B, E\rangle, \langle A, B, D\rangle, \langle A, B, E\rangle\}$. 2 pt
   Objective: apply pathset

3. How many paths does G_2 have? Justify your answer. 2 pt
   Objective: apply pathset
   Objective: understand cycle

   Solution: $\Pi(G_2)$ is infinite, because G_2 is cyclic.

Problem 6.6 (Graphs and SML)

A common way to describe directed graphs is to list the direct neighbors of each node in a structure called "adjacency list". In this problem, we will use a list of pairs (v,l), where l is the adjacency list of vertex v. For example, if we had node 1 connected to node 2 and 3, and 2 connected to 3 and 1, the list would look like:

```
var adjList = [(1, [2, 3]), (2, [3, 1])];
```

In this example, 3 is not connected to any other vertex, so it does not appear in the adjacency list as a first component.

1. Write an SML function `getNeighbors` which, given an graph represented with an adjacency list and a vertex v, returns the list of direct neighbors l corresponding to the vertex, or `nil` if the vertex has no neighbors. 0 pt
   Objective: understand edge

   Function signatures and example:

```
val getNeighbors = fn : (int * int list) list * int -> int list;
- val adjList = [ (1, [2,3]), (2, [4, 5, 6, 7]), (5, [15, 12]) ];
- getNeighbors(adjList, 2);
val it = [4, 5, 6, 7] : int list;
- getNeighbors(adjList, 4);
val it = [] : int list;
```

2. Write an SML function `isTree` which, given a graph represented with an adjacency list, checks whether it is a tree. Remember that trees are directed acyclic graphs with a single node with indegree 0 and all nodes but the root node has indegree 1.
   Function signatures and example:

   0 pt
   Objective: apply tree

```
val isTree = fn : (int * int list) list -> bool;
- val adjList = [ (1, [2,3]), (2, [4, 5, 6, 7]), (5, [15, 12]) ];
- isTree(adjList);
val it = true;
```

3. Write the SML functions `size` and `depth` which return the size and the depth of a tree represented with an adjacency list.
   Function signatures and example:

   0 pt
   Objective: apply size
   Objective: apply depth

```
val size = fn : (int * int list) list -> int;
val depth = fn : (int * int list) list -> int;
- val adjList = [ (1, [2,3]), (2, [4, 5, 6, 7]), (5, [15, 12]) ];
- size(adjList);
val it = 9: int;
- depth(adjList);
val it = 4;
```

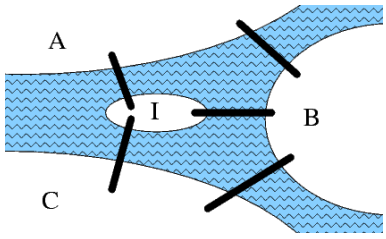
Solution:

```
fun size (leaf) = 0 | size (parent(x,y)) = 1+ size(x) + size(y)
fun depth (leaf) = 0 | depth (parent(x,y)) = 1+ max(depth(x),size(y))
```

Problem 6.7 ((Modified) Königsberg Bridge Problem) 4 pt

Objective: understand cycle
Objective: apply degree

Consider a river fork with three banks (A,B,C) and one island (I) connected with bridges as shown in the diagram below.



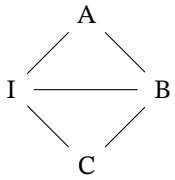
Is it possible to walk accross each of the bridges exactly once in an uninterrupted tour and return to the starting point?

To justify your answer first translate the question into a graph problem where the banks and the island are modeled as nodes and the bridges as undirected edges.

Hint: Consider the degree of each node (i.e.the number for edges connected to it). Relate the degrees of the nodes to the constraint of an uninterrupted tour.

Solution: If there is a cycle which passes all edges of the graph once then each node must have an even degree, because whenever the path enters a node it must leave it again.

As in the given bridge problem the nodes I and B have odd degree there cannot is no such a cycle and thus round trip in the diagram above.



Problem 6.8 (Node Connectivity Relation is an Equivalence Relation) 4 pt

Objective: understand path

Objective: apply equivalence relation

Let $G = \langle V, E \rangle$ be a directed graph and the relation C be defined as

$$C := \{(u,v) \mid \text{there is a path from } u \text{ to } v\}$$

Prove or refute that C is an equivalence relation.

Hint: Recall the properties of an equivalence relation!

Solution:

Proof: We consider the three properties of an equivalence relation

1. reflexivity
   From every node, there is a zero-length path to itself.
3. symmetry
   If there is a path from a node u to a node v, it's not necessary that there is a path from a node v to a node u. So symmetry doesn't hold for C.

5. transitivity
   If there is a path from u to v and a path from v to w, we can reach w from u via v.

□

Problem 6.9 (Operations on Binary Trees) 7 pt

Objective: apply subtree

Objective: understand binary tree

Given the SML datatype `btree` for binary trees and `position` for a position pointer into a binary tree:

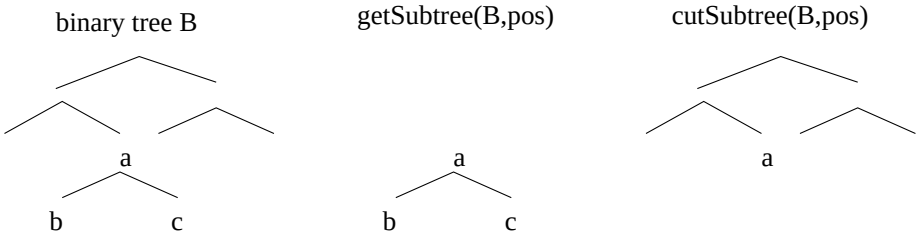
```
datatype btree = leaf | parent of btree * btree;
datatype position = stop | right of position | left of position;
```

The interpretation of a position `right(left(stop))` is like a reversed path: start from root follow the right branch then the left and then stop.

Write two SML functions:

1. `getSubtree` that takes a binary tree and a position and returns the subtree of the that binary at the corresponding position.
2. `cutSubtree` that takes a binary tree and a position and returns the binary tree where the subtree at the corresponding position is cut off; i.e replaced by a leaf.

pos := left(right(stop))



In both cases an exception should be raised if the position exceeds the observed binary tree.

Solution:

```
datatype btree = leaf | parent of btree * btree;
datatype position = stop | right of position | left of position;

exception TreeExceeded;

fun getSubtree (parent(l, r), right(pos)) = getSubtree(r, pos)
  | getSubtree (parent(l, r), left(pos)) = getSubtree(l, pos)
  | getSubtree (tree, stop) = tree
  | getSubtree (_, _) = raise TreeExceeded;
```

```
fun cutSubtree (parent(l, r), right(pos)) = parent(l, cutSubtree(r, pos))
  | cutSubtree (parent(l, r), left(pos)) = parent(cutSubtree(l, pos), r)
  | cutSubtree (tree, stop) = leaf
  | cutSubtree (_, _) = raise TreeExceeded;

val testTree = parent(parent(leaf, parent(leaf, leaf)),parent(leaf, leaf));

val testPos = left(right(stop));

(* test get Subtree:
- getSubtree(testTree, testPos);
val it = parent (leaf,leaf) : btree
- cutSubtree(testTree, testPos);
val it = parent (parent (leaf,leaf),parent (leaf,leaf)) : btree
*)
```

Problem 6.10 (In-degrees in acyclic digraphs) 8 pt

Prove by induction or refute that any acyclic graph with non-empty set of nodes has at least one node with indegree 0.

Objective: understand DAG

Objective: understand indegree

Solution:

Proof: Proof by induction on the size of (number of nodes in) the graph:

1. n=1
   1.1. trivial
3. Step case: $n \Longrightarrow n+1$
   3.1. Note that for any acyclic digraph, every subgraph is also acyclic (otherwise a cycle in the subgraph would make the entire graph cyclic). This is what makes induction possible.
   3.2. So assume an acyclic digraph with n nodes and at least a node with indegree 0 such that the graph is still acyclic. Call this node v_0. Now add an extra vertex, v_n, to the graph and any number of edges that connects it with the rest of the vertices.
   3.3. This new node has no incoming edge.
      3.3.1. This is the new 0 indegree node.
   3.5. v_0 still has no incoming edge.
      3.5.1. v_0 remains the 0 indegree node.
   3.7. v_0 has an incoming edge from v_n and $\text{indeg}(v_n) > 0$
      3.7.1. Then there exists an edge from v_i to v_n. If v_i has indegree 0, then this is a 0 indegree node.
      3.7.2. Otherwise, since the graph is acyclic, v_i has no incoming edge from v_0 or v_n, so there is an edge from another node in the graph.
      3.7.3. Continuing this process either leads to identifying a 0 indegree node or to the situation where a last "untouched" node is reached – but the graph is acyclic, so this node has to have indegree 0.

□

Problem 6.11 (Tree representations) 5 pt

Objective: apply parent

A tree can be represented in many different ways. One method is to specify the parent for each node. For example in a table:

0	1	2	3	4	5	6	7
-1	0	3	0	1	1	3	3

Here having a parent -1 represents a root.

Another representation would be the following SML datatype:

```
datatype tree = node of int * tree list;
```

Write an SML function `convert` that takes a tree, represented as an `(int * int) list` and converts it to the datatype representation.

Signature and example:

```
val convert = fn : int list -> tree

- convert([1,~1,1,2]);
val it = node (1,[node (0,[]),node (2,[node (3,[])])]) : tree
```

Problem 6.12 (Graph basics)

Objective: apply tree

Objective: apply graph depth

Objective: remember path

Objective: apply length

For each of the five directed graphs below do the following:

- State whether the graph is also a tree and explain why.
- Determine the depth of the graph.
- Write out in math notation a path from A to E if one exists and determine its length.

1. $G_1 := \langle \{A,B,C,D,E\}, \{(A,B),(A,C),(A,D),(D,E)\} \rangle$ 3 pt

---

Solution:

- Yes, because it has no cycles and the graph is connected.
- 2
- $\langle A,D,E \rangle$ - length 2

---

2. $G_2 := \langle \{A,B,C,D,E\}, \{(A,B),(B,C),(C,A),(C,D),(C,E)\} \rangle$ 3 pt

---

Solution:

- No, because it has a cycle: $\langle A, B, C, A \rangle$.
- infinite
- $\langle A, B, C, E \rangle$: length 3

---

3. $G_3 := \langle \{A, B, C, D, E\}, \{(A,B), (B,C), (B,D), (C,E)\} \rangle$ 3 pt

---

Solution:

- Yes, because it has no cycles and the graph is connected.
- 3
- $\langle A, B, C, E \rangle$: length 3

---

4. $G_4 := \langle \{A, B, C, D, E\}, \{(A,B), (A,C), (B,D), (D,C), (C,B), (A,D)\} \rangle$ 3 pt

---

Solution:

- No, because it has a cycle: $\langle B, D, C, B \rangle$.
- infinite
- E is not reachable from A, since $indeg(E) = 0$ i.e. E is a source.

---

5. $G_5 := \langle \{A, B, C, D, E\}, \{(D,A), (D,B), (D,E), (D,C)\} \rangle$ 3 pt

---

Solution:

- Yes, because it has no cycles and the graph is connected.
- 1
- E is not reachable from A, since $outdeg(A) = 0$ i.e. A is a sink.

Assignment7 – Structures

Problem 7.1 (Complexity of loops)

For each of the following program fragments compute the worst-case time complexity (as a function of n). You can assume that ⟪body⟫ is executed in constant time.

Hint: Write nested summations to express the number of times each of the loop bodies is executed.

1.
```
i, j := 0
while i ≤ n − 1
   i := i + 1
   while j ≤ 10
      j := j + 1
      ⟪body⟫
   end
end
```

3 pt

Objective: understand time complexity

Solution: We determine the complexities by systematic counting:
The outer loop is executed n times and the inner loop is executed 10 times for each execution of the outer loop. So ⟪body⟫ is executed $10n$ times.
Another way to think about it the number of times the loop body is executed is given by $\sum_{i=0}^{n-1} \sum_{j=1}^{10} 1 = \sum_{i=0}^{n-1} 10 = 10n$ thus the time complexity is in $\mathcal{O}(n)$.

2.
```
i := 0
while i ≤ n − 1
   i := i + 1
   ⟪body⟫
end
i := 0
while i ≤ n − 1
   i := i + 1
   j := i
   while j ≤ n − 1
      j := j + 1
      ⟪body⟫
  end
end
```

3 pt

Objective: apply time complexity

Solution: We determine the complexities by systematic counting:

The first loop has asymptotic time complexity $\mathcal{O}(n)$. For the pair of nested loops, the loop body is executed $\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n - 1 = nn - 1 \operatorname{div} 2$ so the second set of loops has asymptotic time complexity in $\mathcal{O}(n^2)$. Thus the overall asymptotic time complexity of the entire program segment is in $\mathcal{O}(n^2)$.

Problem 7.2 (Algorithm complexities)

Express the time complexity of the following operations in big-O notation in terms of the length N of the unary natural number n, and the length M of m (assuming n is the longer one).

1. Addition π of two unary natural numbers, where π is defined by the equations $\pi(n, o) = o$ and $\pi(n, s(m)) = s(\pi(n, m))$. 2 pt

   Objective: apply batmost

   Solution: $\mathcal{O}(NM)$

   ---

2. Multiplication μ of two unary natural numbers, where μ is defined by the equations $\mu(n, o) = o$ and $\mu(n, s(m)) = \pi(n, \mu(n, m))$. 2 pt

   Objective: apply batmost

   Solution: $\mathcal{O}(N^2M)$

   ---

3. Exponentiation ϵ of unary natural numbers, where $\epsilon(n, o) = s(o)$ and $\epsilon(n, s(m)) = \mu(n, \epsilon(n, m))$. 2 pt

   Objective: apply batmost

   Solution: $\Theta(n^n)$

   ---

4. The follwing SML function (Take one evaluation step to be a creation of a head in function `unwork` and disregard other operations.) 5 pt

   Objective: apply batmost

```
fun gigatwist lst = let
     fun unwork nil = nil |
         unwork(hd::tl) = hd::unwork(tl)
     fun nextwork(nil, _) = nil |
         nextwork(hd::tl, fnc) = fnc(lst)@nextwork(tl, fnc)
     fun nthwork 1 = unwork |
         nthwork n = let
            fun work arg = nextwork(arg, nthwork(n-1))
           in work end
   in nthwork(length lst) lst end
```

Problem 7.3 (Relations among polynomials) 5 pt

Objective: understand batmost

Objective: understand polynomial

Prove or refute that $\mathcal{O}(n^i) \subseteq \mathcal{O}(n^j)$ for $0 \leq i < j, n$ $(i, j, n \in \mathbb{N})$.

Problem 7.4 (Landau sets) 3 pt

Objective: understand Landau set

Order the Landau sets below by specifying which ones are proper subsets and which ones are equal (e.g.: $\mathcal{O}(a) \subset \mathcal{O}(b) \subset \mathcal{O}(c) \equiv \mathcal{O}(d) \subset \mathcal{O}(e)$ ...)

$$\mathcal{O}(n^2);\ \mathcal{O}(n!);\ \mathcal{O}(\sin n + 2);\ \mathcal{O}(n^n);\ \mathcal{O}(1);\ \mathcal{O}(2^n);\ \mathcal{O}(2n^2 + 2^{72})$$

Solution: $\mathcal{O}(\sin n + 2) \equiv \mathcal{O}(1) \subset \mathcal{O}(2n^2 + 2^{72}) \equiv \mathcal{O}(n^2) \subset \mathcal{O}(2^n) \subset \mathcal{O}(n!) \subset \mathcal{O}(n^n)$

Problem 7.5 (Sorting Landau Sets) 7 pt

Objective: apply Landau set

For two functions f and g let us define $f \sim g$ iff $f \in \Theta(g)$ and $f \prec g$ iff $f \in \mathcal{O}(g)$. Write down in terms of these two ordering relations how the following functions are related to each other.

Hint: Since these relations induce a total ordering, it is not necessary to write down explicitly every relation for each pair of functions. Instead you can determine the whole ordering in one line; e.g. in the form of ... $\prec f \sim g \prec h \prec$ ....

---

1. $f_1(n) := 2^{n+3}$
2. $f_2(n) := log(n^2)$
3. $f_3(n) := n^4$
4. $f_4(n) := 2^n$
5. $f_5(n) := log(n/2)$
6. $f_6(n) := 3^n$
7. $f_7(n) := n + 2^4$

Solution: $f_2 \sim f_5 \prec f_3 \sim f_7 \prec f_4 \sim f_1 \prec f_6$

Problem 7.6 (Time complexity of an SML function)

Consider the the following SML function:

```
1 fun twist(nil) = nil
2   | twist([hd1]) = [hd1]
3   | twist(hd1::(hd2::tl)) = twist(hd1::tl)@twist(hd2::tl)
```

1. How many times is line 3 executed to compute `twist [5,6,2,9]`? 2 pt

Objective: apply evaluation

Solution: To compute a list of 4 elements, we execute line 3 eight times: For each recursive calls (and there are 3 for a four-element list) the number of executions doubles (two recursive calls).

---

2. What is the time complexity of `twist` ($\Theta(f)$) in terms of the length of the input list. Justify your answer! 2 pt

Objective: apply time complexity

Objective: understand recursive call

Solution: The recursive case of the function above has two recursive calls to argument lists that are one element shorter, so the evaluations double with every increment in input length. So, time complexity is $\Theta(2^n)$.

Problem 7.7 (Induction for Landau sets) 8 pt

Objective: apply Landau set

Recall the definition of the Landau set $\mathcal{O}(f)$. Prove the following statement by induction on a:

$$\forall n, a \in \mathbb{N}. n^a \in \mathcal{O}(2^n)$$

Please note that any other method of proving the statement will earn you ONLY partial credit.

Hint: Do NOT use limits. When using the definition of the Landau sets try to prove that the formula holds for all even n and then prove it for the odd numbers as well.

Solution: This solution is missing some details, but it's complete enough for the general picture.

Proof:

For $a = 0$ we have $1 \in \mathcal{O}(2^n)$.

1. For $a = 1$ we have $n \in \mathcal{O}(2^n)$ for obvious reasons, and therefore for all $n \geq N_1$ there are N_1, c_1 with $n \leq c_1 * 2^n$.
2. Let's assume that for an arbitrary a the statement holds. We then there are N_2, c_2, such that $n^a \leq c_2 * 2^n$ for all $n \geq N_1$.
3. Multiplying the latter two cases we have:

$$c_1 * c_2 * 2^{2n} \geq n^{a+1} = \frac{2n^{a+1}}{2^{a+1}} \Rightarrow (\exists c. c * 2^{2n} \geq 2n^{a+1})$$

4. We proved that all even numbers are in $\mathcal{O}(2^n)$, so we need to prove it for the odd ones as well.

$$2n + 1^{a+1} \leq 2n + 2^{a+1} \leq c * 2^{2n+2} = (2 * c) * 2^{2n+1}$$

5. Therefore, $2n+1$ is in $\mathcal{O}(2^{2n+1})$, and hence(by setting the constant $k = 2 * c$) we have that $\forall n, a \in \mathbb{N}. n^a \in \mathcal{O}(2^n)$.

□

Problem 7.8 6 pt

Determine for the following functions f and g whether $f \in \mathcal{O}(g)$, or $f \in \Omega(g)$, or $f \in \Theta(g)$, explain your answers.

Objective: apply Landau set
Objective: apply batmost
Objective: apply batleast
Objective: apply bsameas

f	g	f	g
45	7284	$n + 7^2$	n^2
$\log_{10}(n)$	$\log_{10}(n^2)$	$3n^5 - 7n + 4$	n^3
$1^2n + 1 + n^2 - 87$	2^n	16^n	2^n

Solution: The following table summarizes the results.

Fact	Explanation
$45 \in \Theta(7284)$	For all $n \in \mathbb{N}$ we have $.001 \cdot 7284 \leq 45$ and $1000 \cdot 45 \leq 7284$
$n + 7^2 \in \Theta(n^2)$	For all $n \in \mathbb{N}$ we have $1 \cdot n^2 \leq n + 7^2$ and for all $n > 7$ we have $((2n)^2) \leq 4 \cdot (n^2) \leq ((n+7)^2)$
$\log_{10}(n) \in \Theta(\log_{10}(n^2))$	$\log_{10}(n^2) = 2\log_{10}(n)$, so take $c. := 3$ and $c := 3$
$3n^5 - 7n + 4 \in \mathcal{O}(n^3)$	larger exponents win
$2^{n+1} + n^2 - 87 \in \Theta(2^n)$	$2^{n+1} = 2 \cdot 2^n$, all other summands are dominated by this.
$16^n \in \Omega(2^n)$	$16^n = 8^n \cdot 2^n$

The latter explanations can be extended to former-style ones.

Problem 7.9 (Upper and lower bounds)

For each of the functions below determine whether $f \in \mathcal{O}(g)$, $f \in \Omega(g)$ or $f \in \Theta(g)$. Briefly explain your answers.

Objective: apply batmost
Objective: apply batleast
Objective: apply bsameas

1. $f(n) = 235,\ g(n) = 12$ 1 pt

Solution: $f \in \Theta(g)$ both are constants.

---

2. $f(n) = n,\ g(n) = 16n$ 1 pt

Solution: $f \in \Theta(g)$ the leading terms of the polynomial are of the same order.

3. $f(n) = \log_{10}(n),\ g(n) = 7n + 2$ 1 pt

Solution: $f \in \mathcal{O}(g)$ n grows faster than $\log_2(n)$ as in the slides.

4. $f(n) = 7n^3 + 4n - 2,\ g(n) = 3n^4 + 1$ 1 pt

Solution: $f \in \mathcal{O}(g)$ the leading term of f n^3 grows slower than n^4 from g.

5. $f(n) = \frac{\log_2(n)}{n},\ g(n) = \frac{n}{\log_2(n)}$ 1 pt

Solution: $f \in \mathcal{O}(g)$ The numerator of f grows slower that the numerator of g and the denominator of f grows faster than the one from g. There fore f clearly grows slower.

6. $f(n) = 8^n,\ g(n) = 2^n$ 1 pt

Solution: $f \in \Omega(g)$ $8^n = 2^{3n}$ which clearly grows faster than 2^n.

7. $f(n) = n^{\log_n(5)},\ g(n) = 2^n$ 1 pt

Solution: $f \in \mathcal{O}(g)$ 2^n is or a higher rank (see slides) and grows much faster.

8. $f(n) = n^n,\ g(n) = (\log_n(3))n!$ 1 pt

Solution: $f \in \Omega(g)$ n^n is clearly asymptotically bigger than $n!$. And the logarithm in front plays an insignificant role when n is large.

9. $f(n) = \binom{n}{2},\ g(n) = \binom{n}{4}$ 1 pt

Solution: $f \in \mathcal{O}(g)$ The first is a polynomial of degree 2 while the second is

a polynomial of degree 4.

Problem 7.10 (Landau Sets) 4 pt

Objective: apply batmost
Objective: apply batleast
Objective: apply bsameas

Determine for the following functions f and g whether $f \in \mathcal{O}(g)$, or $f \in \Omega(g)$, or $f \in \Theta(g)$, explain your answers.

f	g
$\log(n^4)$	$4n$
$\log(n^4)$	$\log(n)$
$(n+1)^2(n^2-6)$	n^4
$\log_2(5n)$	$n^{\log_n(5)}$

Solution: The following table summarizes the results.

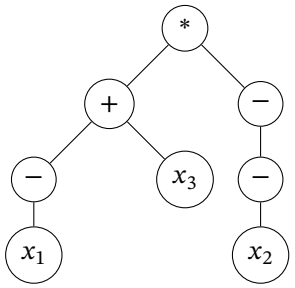
Fact	Explanation
$\log(n^4) \in \mathcal{O}(4n)$	Exponential grows slower than linear.
$\log n^4 \in \Theta(log(n))$	Since $\log(n^4) = 4 \cdot log(n)$
$(n+1)^2(n^2-6) \in \Theta(n^4)$	Same fastest-growing term.
$\log_2(5n) \in \Omega(n^{\log_n(5)})$	f is logarithmic, while g equals 5, i.e. constant.

Assignment8 – Parse Trees

Problem 8.1 (Graph and Trees)

1. Draw the parse tree of the expression string `(-X1+X3)*(-(-X2))` with respect to the grammar for arithmetic expressions presented in the SMAI course. 3 pt
   Objective: apply parse tree

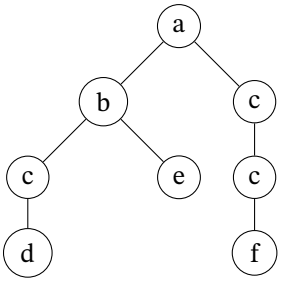
Solution:



---

2. Give an example of a different node labeled graph that is isomorphic to the parse tree above and write its mathematical representation in terms of concrete sets and functions. 3 pt
   Objective: apply isomorphism
   Objective: remember graph

Solution: One such graph is



A parse tree is a node labeled graph, i.e. a graph

$$\langle\{1,2,3,4,5,6,7,8\},\{(1,2),(2,3),(3,4),(2,5),(1,6),(6,7),(7,8)\}\rangle$$

and a node labeling function:

$$1 \mapsto a, 2 \mapsto b, 3 \mapsto c, 4 \mapsto d, 5 \mapsto e, 6 \mapsto c, 7 \mapsto c, 8 \mapsto f$$

---

3. Can the graph underlying the parse tree be isomorphic to a cyclic graph? Justify why, or why not? 2 pt

Objective: understand isomorphism

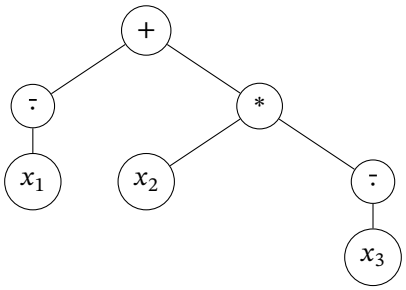
Objective: understand cycle

Solution: No, trees are acyclic and cycles are preserved under graph isomorphisms.

Problem 8.2 (Parse Tree)

Draw the parse tree of the expression $\overline{x_1}+x_2*\overline{x_3}$ and provide a mathematical representation of it.

Solution:



$G := \langle V, E, f \rangle$
$V := \{A, B, C, D, 1, 2, 3\}$
$E := \{(A,B), (A,C), (B,1), (C,2), (C,D), (D,3)\}$
$f(A) = +$, $f(B) = f(H) = \bar{\cdot}$, $f(C) = *$, $f(D) = x_1$, $f(F) = x_2$, $f(I) = x_3$

Problem 8.3 (Parse Tree) 3 pt

Draw the parse tree of the expression $(((x_1 * x_2) + x_3) * \overline{x_1 + x_4})$.

Problem 8.4 (Parse Tree) 3 pt

Draw the parse tree of the expression $((\overline{x_1 * x_2}) + (x_2 * (x_1 + x_4)))$ and present a mathematical representation of it.