General Computer Science I (320101) Fall 2014

Michael Kohlhase
Jacobs University Bremen
For Course Purposes Only

November 11, 2014

Contents

Assignment 1: Elementary Math
(Given Sep. 11., Due Sep. 18.)

Problem 1.1 (Induction on an inequality)
Prove by induction the following inequality: 35pt

$$\frac{a_1 + a_2 + \ldots + a_n}{n} \geq \sqrt[n]{a_1 \cdot a_2 \cdot \ldots \cdot a_n}$$

for $n = 2^k$, $a_i \geq 0$, $a_i \in \mathbb{R}$ and $k \in \mathbb{N}$

Note: It actually holds for all $n \in \mathbb{N}$.

Solution: Base case 1:

$$a_0 \geq a_0$$

Base case 2:

$$\frac{a_0 + a_1}{2} \geq \sqrt{a_0 \cdot a_1} \iff \tag{1}$$

$$\frac{a_0 - 2 \cdot \sqrt{a_0 \cdot a_1} + a_1}{2} \geq 0 \iff \tag{2}$$

$$\frac{(\sqrt{a_0} - \sqrt{a_1})^2}{2} \geq 0 \tag{3}$$

which is always true. Hence, the base case holds.
Assume that for some $n = 2^k$ the statement holds true. (A)
For $n = 2^{k+1}$ we have the following:

$$\frac{a_1 + a_2 + \ldots + a_{2^{k+1}}}{2^{k+1}} = \frac{\frac{a_1 + a_2 + \ldots + a_{2^k}}{2^k} + \frac{a_{2^k+1} + a_{2^k+2} + \ldots + a_{2^k}}{2^k}}{2} \overset{(A)}{\geq}$$

$$\overset{(A)}{\geq} \frac{\sqrt[2^k]{a_1 \cdot a_2 \cdot \ldots \cdot a_{2^k}} + \sqrt[2^k]{a_{2^k+1} \cdot a_{2^k+2} \cdot \ldots \cdot a_{2^{k+1}}}}{2} \overset{(3)}{\geq} \sqrt[2^{k+1}]{a_1 \cdot a_2 \cdot \ldots \cdot a_n}$$

Our proof is now done. Note that the induction is done on the variable k.

Problem 1.2 (Unary Natural Numbers)
Let $\oplus$ be the addition operation and $\odot$ be the multiplication operation on unary natural numbers as defined on the slides. Prove or refute that: 30pt

1. $a \oplus b = b \oplus a$
2. $(a \oplus b) \odot c = a \odot c \oplus b \odot c$
3. $a \odot b = b \odot a$

Solution:

1. **Proof**: We proceed by induction over b:

P.1 Base case:

$$a \oplus 0 = 0 \oplus a$$

P.1 W e have $a \oplus 0 = a$ by the addition axiom, so we need $a = 0 \oplus a$. This is easily proven by induction over a.

P.3 Step case: We first prove by induction over a that

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

P.3 B ase case: $a = 0$. We need $s(b) \oplus 0 = b \oplus s(0)$. But from the previous step we know:

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

Step case, assume $s(b) \oplus a = b \oplus s(a)$. Now:

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

Back to the problem, assume $a \oplus b = b \oplus a$ and let's prove that $a \oplus s(b) = s(b) \oplus a$.

P.4 W e have $a \oplus s(b) = s(a \oplus b) = s(b \oplus a) = b \oplus s(a)$. This is equal to $s(b) \oplus a$, which is what we need to prove. □

2. **Proof**: We proceed by induction over c.

P.1 Base case: $(a \oplus b) \odot 0 = a \odot 0 \oplus b \odot 0$.

P.1 T his is true by the fact that $\forall n.n \odot 0 = 0$.

P.3 Step case: Assume that $(a \oplus b) \odot c = a \odot c \oplus b \odot c$. We need $(a \oplus b) \odot s(c) = a \odot s(c) \oplus b \odot s(c)$

P.3 W e have, from the multiplication axioms:

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

□

3. **Proof**: We proceed by induction over b.

P.1 Base case: $a \odot 0 = 0 \odot a$ can be easily proven by induciton over a.

P.2 Step case, assume $a \odot b = b \odot a$. We need $a \odot s(b) = s(b) \odot a$.

P.2 W e 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. □

Problem 1.3: Figure out the functions on natural numbers for the following defining equations 15pt

$$\delta(o) = o$$

$$\delta(s(n)) = s(s(\delta(n)))$$

Solution:

The function δ doubles its argument.

Problem 1.4 (A wrong induction proof)

What is wrong with the following "proof by induction"? 20pt

Theorem: All students of Jacobs University have the same hair color.

Proof: We prove the assertion by induction over the number n of students at Jacobs University.

base case: $n = 1$. If there is only one student at Jacobs University, then the assertion is obviously true.

step case: $n > 1$. We assume that the assertion is true for all sets of n students and show that it holds for sets of $n + 1$ students. So let us take a set S of $n + 1$ students. As $n > 1$, we can choose students $s \in S$ and $t \in S$ with $s \neq t$ and consider sets $S_s = S\backslash\{s\}$ and $S_t := S\backslash\{t\}$. Clearly, $\#(S_s) = \#(S_t) = n$, so all students in S_s and have the same hair-color by inductive hypothesis, and the same holds for S_t. But $S = S_s \cup S_t$, so any $u \in S$ has the same hair color as the students in $S_s \cap S_t$, which have the same hair color as s and t, and thus all students in S have the same hair color □

Solution:

The problem with the proof is that the inductive step should also cover the case when $n = 1$, which it doesn't. The argument relies on the fact that there intersection of S_s and S_t is non-empty, giving a mediating element that has the same hair color as s and t. But for $n = 1$, $S = \{s, t\}$, and $S_s = \{t\}$, and $S_t = \{s\}$, so $S_s \cap S_t = \emptyset$.

Assignment 2: Elementary Math
(Given Sep. 18., Due Sep. 25.)

Problem 2.1 (Are bijective functions with composition a group?)
A group is a set G with a binary operation $*\colon G \times G \to G$, obeying the following axioms: 25pt

Closure: G is closed under $*$, i. e. $\forall a, b \in G . a * b \in G$

Associativity: $\forall a, b, c \in G . (a * b) * c = a * (b * c)$

Identity element: $\exists e \in G . \forall a \in G . a * e = e * a = a$.

Inverse elements: $\forall a \in G . \exists a^{-1} \in G . a * a^{-1} = e$.

If, additionally, the following axiom holds, the group is called "commutative" or "Abelian":

Commutativity: $\forall a, b \in G . a * b = b * a$.

- Now prove or refute whether the set of all bijective functions $f\colon A \to A$ on a set A with the function composition $\circ$ forms a group.
- Is it commutative?

Solution:

Associativity:

Identity element: the identity function $\lambda x.x$

Inverse elements: f^{-1}, exists due to bijectivity

Commutativity: No, easy counter-example.

Problem 2.2: Are the following functions total, injective, surjective and/or bijective? 25pt

- $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2 - 3x + 6$
- $g : \mathbb{N} \to \mathbb{N}, g(x)$ represents the number of distinct prime divisors of x
- $h\colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$,

$$h(m, n) := \frac{(m + n - 2)(m + n - 1)}{2} + m$$

Prove your answers.

Solution: f is total because $f(x)$ is defined for all real values of x. f is not injective because $f(0) = f(9) = 6$. f is also not surjective because the minimal value that f can obtain is $f(4.5) = 12.75$.
g is total because every natural number can be represented as a number of prime divisors. g is

not injective because both 4 and 9 have the same amount of prime divisors(1). g is surjective because for any number of natural divisors k, the number $p_1 \cdot p_2 \cdot \ldots \cdot p_k$ will be a number that has exactly k prime divisors.
h is total because it takes a defined value for every two natural numbers m and n. h is not injective because $h(1,0) = h(1,1) = 1$. h is surjective(I will write the proof soon).

Problem 2.3: Check whether the following are equivalence relations(you can assume all of them are from the set of natural numbers to itself): 25pt

- $x \sim y \iff x = y(mod 3)\ \forall x, y \in \mathbb{N}$
- $x \sim y \iff x^2 = y^2\ \forall x, y \in \mathbb{N}$
- $x \sim y \iff x|y$

Solution:

- $\sim$ is an equivalence relation. It is reflexive because $x = x(mod\ 3)$ for all $x \in \mathbb{N}$. It is also symmetric because if $x = y(mod\ 3)$ then $y = x(mod\ 3)$. Transitivity follows from $x = y(mod\ 3) \wedge y = z(mod\ 3) \implies x = z(mod\ 3)$.
- $\sim$ is an equivalence relation. $x^2 = x^2$ shows reflexivity, while $x^2 = y^2 \implies y^2 = x^2$ and $x^2 = y^2 \wedge y^2 = z^2 \implies x^2 = z^2$ due to positivity prove symmetry and transitivity.
- $\sim$ is not an equivalence relation since it isn't symmetric. For example, $3|6$ but $6 \not| 3$.

Problem 2.4 (Relation Cardinality)
Let S be a set of n elements. 25pt

1. How many different relations on S are there?
2. How many of these relations are reflexive?
3. How many of these relations are symmetric?
4. How many of these relations are reflexive and antisymmetric at the same time?

Justify your answers.

Solution:
Proof:

P.1 We first calculate the number of total relations on S. We have $\binom{n}{2}$ different pairs of elements.

P.2 The number of relations on two different elements $\langle a, b\rangle \in S$ and $\langle b, a\rangle \in S$ is $2 \cdot \binom{n}{2}$.

P.3 If we also include the reflexive relations $\langle a, a\rangle \in S$ we will have $2 \cdot \binom{n}{2} + n$ total relations.

P.4 Now we need to find the number of ways to select these relations to form a relation on $\{S\}$, by either choosing the respective relation, or not. We then find $2^2 \cdot \binom{n}{2} + n = 2^n \cdot (n-1) + n = 2^{n^2}$

P.5 We can simplify the next steps a bit by imagining an $n \cdot n$ grid to account for the tuples. For reflexive relations we have the diagonal elements fixed, thus starting from n elements.

P.6 We still have the possibility of including any of the other tuples, be they below or on top of the main diagonal, thus reaching $2^{n^2} - n = 2^{n\ times(n-1)}$

P.7 We may use the grid discretization again. For symmetry, we of course need to include $\langle b, a\rangle$ every time we have $\langle a, b\rangle$. This results in considering any elements on and above (or on and below) the main diagonal

P.8 There are a total of $\frac{n\cdot(n+1)}{2}$ such possibilities so our desired number will be $2^{\frac{n\cdot(n+1)}{2}}$

P.9 Let us first consider antisymmetry. By following the same matrix/grid procedure, we distinguish between diagonal and non-diagonal elements.

P.10 For diagonal elements, there are two possibilities for each, and since we have n such elements, we obtain a total of 2^n possibilites.

P.11 For each pair of non-diagonal elements a and b, we have the following possibilities:

1. $tupa, b$ in, $tupb, a$ out
2. $tupa, b$ out, $tupb, a$ out
3. $tupa, b$ out, $tupb, a$ in

The number of distinct pairs of elements is $fracn \cdot (n-1)2$, and since we have three possibilities for each pair, we will have a total number of $3^{\frac{n\cdot(n-1)}{2}}$ possibilities for non-diagonal elements

P.12 This leads to a total number of antisymmetric relations of $2^n \cdot 3^{\frac{n\cdot(n-1)}{2}}$

P.13 In order to get both reflexive and antisymmetric relations, we simply lose the degree of freedom in choosing the diagonal elements. Hence we end up with $3^{\frac{n\cdot(n-1)}{2}}$ as our final answer. □

Assignment 3: SML Language
(Given Sep. 25., Due Oct. 2.)

Problem 3.1: We call the sequence $a_0 = 0, a_1 = 1, a_n = a_{n-1} + a_{n-2}$ a Fibonacci sequence. Write an SML function $fib = fn : int \to int$ that calculates the nth Fibonacci number. Also, write an SML function $isfib = fn : int \to bool$ that, given a number n, calculates whether n is a fibonacci number or not. You are allowed to define additional functions to help you. 25pt

---

Solution:

```
fun fib(n) = if n<=0 then 0 else if n=1 then 1 else fib(n−1) + fib(n−2);
```

Note: this solution is extremely inefficient.A better one is covered in Chapter 6.

```
fun is_fib(x) = let
                    fun helper(current, value) =
                     if fib(current) < value then helper(current + 1, value)
                        else if fib(current = value) then true else false;
                in
                    helper(0,x)
                end;
```

Problem 3.2: A palindrome is a sequence that is the same regardless of whether it's read right to left or left to right. For example, 12321 is a palindrome while 145 is not. Write an SML function $palindrome = fn :' a\ list \to bool$ that takes a list and outputs whether that list represents a palindrome or not. 15pt

---

Solution:

```
fun rev(nil) = nil
   |rev(a::b) = rev(b)@([a]);

fun palindrome(x) = x=rev(x);
```

Another way to implement reverse is to keep all the elements to a new list in backwards order.

```
fun rev_helper(nil, x) = x
   |rev_helper(a::b, x) = rev_helper(b, a::x);

fun rev(x) = rev_helper(x, nil);
```

---

lstsetlanguage=ML

Problem 3.3 (SML Rap functions)

The time has come, you have decided that you and your friends are rather talented and want to go into the music business. However the group's imagination is not very good so you need to create an SML program that will generate your rap lyrics. This program rap(words,numbers) which will take two lists and repeat each word the corresponding number of times in the same position in the numbers list. If both lists do not have the same length just ouput the empty list. 25pt

Function signature and example:

```
val rap = fn : 'a list * int list -> 'a list
- rap(["we","love","the","s","m","l"],[1,3,0,3,2,1]);
val it = ["we","love","love","love","s","s","s","m","m","l"] : string list
```

Solution:

```
fun repeat(word,num) = if num > 0 then word::(repeat(word,num-1)) else nil

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

fun rap(a,nil) = nil
  | rap(nil, a) = nil
  | rap(a::words, b::nums) = if (len(words) = len(nums)) then repeat(a,b)@(rap(words,nums)) else nil
```

Problem 3.4: Given a sequence a_n we call $a_0 - a_1 + a_2 - a_3 + ...$ the alternating sum of the sequence. Write an SML function $alternating = fn : intlist \to int$ that calculates the alternating sum of the list. 15pt

---

Solution:

```
fun alternating(nil) = 0
   |alternating(a::b) = a - alternating(b);
```

Problem 3.5 (Remove duplicates) 20pt

Write an SML function removeDupl that takes a string as an argument and replaces all adjacent repeated occurrences of a character with a single instance of that character.

```
val removeDupl = fn : string -> string
```

For example,

```
- removeDupl("aabbacccdd");
val it = "abacd" : string
```

Solution:

```
(* Sample Solution *)
fun removeD [] = []
| removeD(x::nil)=[x]
| removeD(x::xs)= if x=hd(xs) then removeD(xs) else x::removeD(xs);

fun removeDupl l = implode(removeD(explode l));


(* Test cases *)
val test1 = removeDupl "aaaaaabbbbbbcccc" = "abc";
val test2 = removeDupl "fGGGGGHiiiii" = "fGHi";
val test3 = removeDupl "" = "";
```

Assignment 4: Datatypes in SML (Given Oct. 2., Due Oct. 9.)

Problem 4.1 (Counting nucleotides)

One of the tasks Bioinformatics students used to have was computing the frequency of DNA nucleotides (the structural elements of the nucleic acid). But, one day, after learning SML in GenCS, a student came with the idea of writing a function that automatically does this job for him. He created the following datatype: 20pt

```
datatype DNA = T| C | G | A;
```

He asks you to help him implement a function that returns a tuple that represents the frequencies of T, C, G and A respectively and which has the following signature:

```
val frequency = fn : DNA list -> real * real * real * real
```

Example:

```
- frequency([T,C,C,A,A,G,A,G]);
val it = (0.125,0.25,0.25,0.375) : real * real * real * real;
```

Solution:

```
datatype DNA = T | C | G | A;

fun helper( [] ) = (0.0,0.0,0.0,0.0,0.0)
  | helper( T :: L) = let val (nt, nc, ng, na, n) = helper(L) in (nt+1.0, nc,ng,na,n+1.0) end
  | helper( C :: L) = let val (nt, nc, ng, na, n) = helper(L) in (nt, nc+1.0,ng,na,n+1.0) end
  | helper( G :: L) = let val (nt, nc, ng, na, n) = helper(L) in (nt, nc,ng+1.0,na,n+1.0) end
  | helper( A :: L) = let val (nt, nc, ng, na, n) = helper(L) in (nt, nc,ng,na+1.0,n+1.0) end;


fun frequency(X) = let val (nt,nc,ng,na,n) = helper(X) in (nt/n,nc/n,ng/n,na/n) end;
```

Problem 4.2 (Sort the mushroom!)

Red Riding Hood collected in her basket three types of mushrooms: red ones, green ones and yellow ones. We know the following about each type of mushroom: 30pt

- The red mushrooms have a number of spots on their cap, and the more the merrier.
- The green mushrooms are edible only when they have a black spot on their cap.
- The yellow mushrooms are tastier when they are less dense (and we can only measure weight and volume, but we still can determine which mushroom is tastier, can't we?).

As soon as she arrived at her grandmother's house, she started to arrange the mushrooms by color (red first, then green and then yellow), and each mushroom by it's property (for red ones, she wanted increasing number of spots, for green ones, the edible should be at the right of non-edible, and for yellow ones, in increasing order of tastiness). Help her finish faster by writing an SML data type mushroom that can represent a mushroom and a

function sort that, given a list of mushrooms, returns them in the order Red Riding Hood wants them.

For example, say that the girl found 2 red mushrooms (r_1 with 10 dots and r_2 with 12 dots), 1 yellow mushroom (y with weight 2 and volume 3) and 2 green mushroom (g_1 with a black dot and g_2 without). The result of calling sort on the list $[r_1, r_2, y, g_1, g_2]$ would be the list $[r_1, r_2, g_2, g_1, y]$.

Solution:

```
datatype mushroom = red of int | green of bool | yellow of int*int;

(* use this to faster compare two mushroom(s) *)
fun compare(red(a), red(b)) = a<b
  | compare(red(_), green(_)) = true
  | compare(green(_), red(_)) = false
  | compare(red(_), yellow(_,_)) = true
  | compare(yellow(_,_), red(_)) = false
  | compare(green(a), green(b)) = a=false orelse b=true
  | compare(green(_), yellow(_,_)) = true
  | compare(yellow(_,_), green(_)) = false
  | compare(yellow(a1,b1), yellow(a2,b2)) = a1*b2 < a2*b1; (* always prefer multiplication over division! *)

(* classical mergesort algo *)
fun merge([], B) = B
  | merge(A, []) = A
  | merge(a::A, b::B) = if compare(a,b) then a::merge(A,b::B) else b::merge(a::A, B);


fun sort([a]) = [a]
  |sort(A) = let val len = List.length(A)
                 val l1 = List.take(A, len div 2)
                 val l2 = List.drop(A, len div 2)
             in
                 merge(sort(l1), sort(l2))
             end;
```

Problem 4.3 (High-Order functions - Look up)

You are given the following SML datatype 25pt

```
datatype num = undefined | value of int;
```

Your task is to program a function create_lookup in SML. The function takes three arguments as inputs: two functions f and g and a list l. create_lookup returns a new function as a result. If we call the result *lookup* then it is defined in the following way

$$lookup(x) = \begin{cases} value(y) & \text{if } \exists v \in l. f(v) = x \wedge g(v) = y \\ undefined & \text{if } \nexists v \in l. f(v) = x \end{cases}$$

If the condition $f(v) = x \wedge g(v) = y$ is satisfied by more that one v just pick any pair.

The signature of create_lookup is:

```
create_lookup = fn : ('a -> int) * ('a -> int) * 'a list -> int -> num
```

Here is an example:

```
fun identity x = x;
fun inverse x = ~x;
val l = [1,2,3];
val lookup = create_lookup (inverse,identity,l);
lookup 1;
  val it = undefined : num
lookup ~2;
  val it = value 2 : num
```

Solution:

```
datatype num = undefined | value of int;

fun combine nil nil = nil
  | combine (a::l) nil = (a,a)::(combine l nil)
  | combine nil (a::l) = (a,a)::(combine nil l)
  | combine (a::l) (b::m) = (a,b)::(combine l m);

fun create_lookup (funa,funb,l) =
  let
    val map1 = map funa l;
    val map2 = map funb l;
    val table = combine map1 map2;
  in
    fn x => ( foldl
      (fn ((ca,cb),p) => if p=undefined andalso ca = x then value(cb) else p)
      undefined
      table
      )
  end;

(* Test cases *)

(* we need some functions to test with *)
val l = [1,2,3,4,~5,6];
fun add1 x = x+1;
fun inv x = ~x;
val test_fun = create_lookup (add1,inv,l);

val test1 = test_fun 0 = undefined;
val test2 = test_fun 1 = undefined;
val test3 = test_fun ~1 = undefined;
val test4 = test_fun ~2 = undefined;
val test5 = test_fun ~3 = undefined;
val test6 = test_fun 10 = undefined;
val test7 = test_fun 2 = value(~1);
val test8 = test_fun 3 = value(~2);
val test9 = test_fun 4 = value(~3);
val test10 = test_fun 5 = value(~4);
val test11 = test_fun 6 = undefined;
val test12 = test_fun 7 = value(~6);
val test13 = test_fun ~4 = value(5);
```

Problem 4.4 (Understaning map)
Given the SML higher-order function **fun** f x = **fn** (y) => y::x and the list **val** l = [1,2,3]. 15pt

- Determine the types of f and map (f l)
- evaluate the exression map (f l) l

Solution:

```
- fun f x = fn (y) => y::x;
val f = fn : 'a list -> 'a -> 'a list
- map (f l);
val it = fn : int list -> int list list

- map (f l) l;
val it = [[1,1,2,3],[2,1,2,3],[3,1,2,3]] : int list list
```

Problem 4.5: Given the following SML data type for an arithmetic expressions 25pt

```
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 *)
```

give the representation of the expression $(4x + 5) - 3x$.

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.

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:

```
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 *)

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

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;

(* 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
- *)
```

Assignment 5: Elementary Math
(Given Oct. 8., Due Oct. 11.)

Problem 5.1 (Abstract Procedures on a Deck of Cards)
You are given the following abstract data type for a deck of cards: 35pt

$$\langle \{\mathbb{D}, \mathbb{C}, \mathbb{S}, \mathbb{N}, \mathbb{B}\}, \{[nil\colon \mathbb{D}], [S\colon \mathbb{S}], [H\colon \mathbb{S}], [C\colon \mathbb{S}], [D\colon \mathbb{S}], [o\colon \mathbb{N}], [s\colon \mathbb{N} \to \mathbb{N}], [card\colon \mathbb{S} \times \mathbb{N} \to \mathbb{C}], [add\colon \mathbb{C} \times \mathbb{D} \to \mathbb{D}], [T\colon \mathbb{B}], [F\colon \mathbb{B}]$$

1. given a deck, count the number of cards of a specific suit:

   $$count\_suit(add(card(S, s(s(o))), add(card(D, s(s(s(o)))), add(card(S, s(s(s(o)))), nil))), S)$$
   $$= s(s(o))$$

2. given a deck, count the number of cards with a specific number:

   $$count\_number(add(card(S, s(s(o))), add(card(D, s(s(s(o)))), add(card(S, s(s(s(o)))), nil))),$$
   $$s(s(s(o)))) = s(s(o))$$

3. given a deck, reverse the order of the cards inside:

   $$reverse(add(card(H, s(s(o))), add(card(D, s(s(s(o)))), add(card(S, s(s(s(o)))), nil))))$$
   $$= add(card(S, s(s(s(o)))), add(card(D, s(s(s(o)))), add(card(H, s(s(o))))), nil)$$

4. given 2 decks, check if the first is included in the second:

   $$included(add(card(D, s(s(o))), add(card(S, s(o)), nil)),$$
   $$add(card(D, s(s(o))), add(card(H, s(s(s(o)))), add(card(S, s(o)), nil)))) = T$$

5. given 2 decks, check if they contain they contain the same cards:

   $$same(add(card(D, s(s(o))), add(card(S, s(s(o))), nil)), add(card(D, s(s(o))),$$
   $$add(card(H, s(s(o))), nil))) = F$$

6. given a deck, sort the cards inside so that they are first ordered by suit (spades, hearts, diamonds and then clubs) and then in increasing order by their number:

   $$sort(add(card(D, s(s(o))), add(card(H, s(s(o))), add(card(D, s(s(s(o)))),$$
   $$add(card(S, s(s(o))), add(card(C, s(s(s(o)))), nil))))))$$
   $$= add(card(S, s(s(o))), add(card(H, s(s(o))), add(card(D, s(s(o))),$$
   $$add(card(D, s(s(s(o)))), add(card(C, s(s(s(o)))), nil)))))$$

Solution:

Problem 5.2 (ADT for rational numbers)

Define an abstract data type for rational numbers. Write the numbers $\frac{2}{3}$ and $-\frac{1}{2}$ using your definition. Also define all other data types you need to construct rationals. 25pt

Note: Make sure that there is only one representation for the integer 0!

A proper data type for rational numbers would not be allowed to have more than one representation for one value. As reducing fractions is a bit out of scope here, you can ignore this problem and permit e. g. two distinct representations for $\frac{2}{4}$ and $\frac{1}{2}$, although these rational numbers are actually equal.

Solution: Thanks to Dimitar "Dasenov" Asenov for contributing this exercise. Thanks to Felix Schlesinger for contributing to the disclaimer about mathematical soundness.

$$\langle \{\mathbb{P}, \mathbb{I}, \mathbb{Q}\}, \{[one\colon \mathbb{P}], [suc\colon \mathbb{P} \to \mathbb{P}], [zero\colon \mathbb{I}], [pos\colon \mathbb{P} \to \mathbb{I}], [neg\colon \mathbb{P} \to \mathbb{I}], [q\colon \mathbb{I} \times \mathbb{P} \to \mathbb{Q}]\} \rangle$$

... where $\mathbb{P}$ are the positive integers, $\mathbb{I}$ all integers and $\mathbb{Q}$ rational numbers.

There are actually many ways to define this, but in any case it's important that 0 and any number $\neq 0$ are of different sorts.

$$\frac{2}{3} = q(pos(suc(one)), suc(suc(one)))$$
$$-\frac{1}{2} = q(neg(one), suc(one))$$

Grading for 100% = 4 points:

- 0.5 points for each example (they're easy once the ADT works)
- 1 point per sort: naturals, integers, fractions

Problem 5.3 (Applying substitutions)

Given the expressions $s = h(a, x, f(g(y, a), z, x))$ and $t = f(x, y, g(z, h(x)))$ and the substitutions $\sigma := [(h(y))/x], [(g(z, x))/y], [x/z]$ and $\tau := [(f(x, g(y, a)))/a], [z/x], [(h(x, y))/y], [(f(a))/z]$ write down all combinations of substitution applications. 10pt

Note: We don't care about the type in this problem, instead we assume that all symbols are appropriately typed.

Solution: $\sigma(s) = h(a, h(y), f(g(g(z, x), a), x, h(y)))$
$\sigma(t) = f(h(y), g(z, x), g(x, h(h(y))))$
$\tau(s) = h(f(x, g(y, a)), z, f(g(h(x, y), f(x, g(y, a))), f(a), z))$
$\tau(t) = f(z, h(x, y), g(f(a), h(z)))$

Problem 5.4 (SML datatype and ADT)

The following SML datatype: 30pt

```
datatype figure = f_id of int;
datatype hw = title_page of int | solution_page of int*hw | figure_page of figure*hw;
```

represents homeworks. The number of each homework is indicated on the title page. After that more pages follow (or none if the student didn't submit anything). A solution page is characterized by the number of the problem it contains the solution to. A figure page contains a single figure. A figure is characterized by an id.

1. Convert the datatypes above to an ADT. Use natural numbers for integers.
2. Show in your ADT representation how the example below will look like.
   solution_page(2,figure_page(f_id(1),solution_page(3,solution_page(1,title_page(2))))).

Solution:

1. $\langle\{\mathbb{N}, \mathbb{F}, \mathbb{H}\}, \{[o\colon \mathbb{N}], [s\colon \mathbb{N} \to \mathbb{N}], [f\_id\colon \mathbb{N} \to \mathbb{F}], [tp\colon \mathbb{N} \to \mathbb{H}], [sp\colon \mathbb{N} \times \mathbb{H} \to \mathbb{H}], [fp\colon \mathbb{F} \times \mathbb{H} \to \mathbb{H}]\}\rangle$
2. $sp(s(s(o)), fp(f\_id(s(o), sp(s(s(s(o))), sp(s(o), tp(s(s(o))))))))$

Assignment 6: More SML and Formal Languages (Given Oct. 16., Due Oct. 30.)

Problem 6.1 (Mutual Recursion) 25pt

Write two mutually recursive SML functions odd and even that given $n > 0$ return how many words of length n over the alphabet $\{A, B, C, D\}$ there are which contain an odd number of Bs and an even number of Bs respectively. Make sure you raise appropriate exceptions. Signature and example:

```
val odd = fn : int -> int
val even = fn : int -> int

- odd(2);
val it = 6 : int
- even(3);
val it = 36 : int
```

Explanation for the first example: The words of length two with an odd number of Bs are BA, BC, BD, AB, CB and DB.

Solution:

```
exception nonpositive;

fun odd(1) = 1
   | odd(n) = if n<1 then raise nonpositive else 3*odd(n-1) + even(n-1)
and even(1) = 3
   | even(n) = if n<1 then raise nonpositive else 3*even(n-1) + odd(n-1);
```

Problem 6.2: A Turing machine is a machine that has an infinite tape, a set of states and a set of rules. Let's call $\mathcal{T}$ a simplified Turing machine that has a starting input(string) and state actions that are defined in a file that has the following format: 30pt

```
state_name new_symbol
```

state_name can be either p, q or r. *new_symbol* is either 0 or 1 and represents the character with which we're replacing the current character. The simplified Turing machine has a head that is originally pointing to the beginning of the string. When the machine is in state p the head moves to the right, when it is in state q it moves to the left and when it is in state r it stays in the same place. The current state of the machine is determined by the *state_name* of the file(the first line is the first state, the second line is the second state etc.). When a state is finished executing, the machine moves to the next state in the file. Raise appropriate exceptions when the current state moves the head outside of the limits of the string and for all other cases that you can think of.

Your task is to write an SML function *evaluate* that takes a filename that contains the rules for the states and a string as the input and outputs a string that is the result when all states finish executing. Your function must have the following signature:

```
val evaluate = fn: string->string->string

evaluate "filename.txt" "11100";
-val "10101":string
```

In the example above, *filename.txt* might look like this:

```
p 1
p 0
p 1
p 0
p 1
```

Problem 6.3 (Basis for Huffman encoding in SML)

30pt

Huffman is an encoding algorithm usually used for compressing files based on the principle that the characters that occur most often have the shortest code. Your task is to define a prepareHuffman function in SML that will take a string as input and will output a list of tuples of character and the frequency of the character in the string. The list should be sorted by the frequency in an ascending manner.

The signature of the function should be:

```
val prepareHuffman = fn : string -> (char * int) list
```

An example output of is:

```
- prepareHuffman("Some sample text");
val it =
  [(#"S",1),(#"o",1),(#"s",1),(#"a",1),(#"p",1),(#"l",1),(#"x",1),(#"m",2),
   (#" ",2),(#"t",2),(#"e",3)] : (char * int) list
```

Solution:

```
fun eliminate(a, nil) = nil
  | eliminate(a, b::r) = if a = b then eliminate(a,r) else b::eliminate(a,r);

fun count(a, nil) = 0
  | count(a, b::r) = if a = b then 1 + count(a, r) else count(a, r);

fun doWork(nil) = nil
  | doWork(h::t) = (h,count(h,h::t))::doWork(eliminate(h,t));

fun findsmallest(nil, a) = a
  | findsmallest((e,c)::r, (e1,c1)) = if (c < c1) then findsmallest(r, (e,c)) else findsmallest(r, (e1,c1));

fun sort(nil) = nil
  | sort(l) = let val x = findsmallest(l,hd(l)) in x::sort(eliminate(x,l)) end;

fun prepareHuffman(string) = sort(doWork(explode(string)));

(* Test cases: *)
```

```
fun eliminatetup(c,f,nil) = nil
  | eliminatetup(c,f,(a,b)::r) = if (c = a) andalso (f = b) then r else (a,b)::eliminatetup(c,f,r);

fun checkOne(f, c, nil) = false
  | checkOne(f, c, (c1, f1)::r) = if (f <> f1) then false else if (c = c1) then true else checkOne(f,c,r);

fun checkInternal(f, nil, nil) = (true, nil, nil)
  | checkInternal(f, a, nil) = (false, a, nil)
  | checkInternal(f, nil, a) = (false, nil, a)
  | checkInternal(f, (c1,f1)::r1, l2) = if (f = f1) then if checkOne(f, c1, l2) then checkInternal(f, r1, eliminatetup(c1,f
(true, (c1,f1)::r1, l2);

fun check(nil, nil) = true
  | check(a, nil) = false
  | check(nil, b) = false
  | check(a, b) =
    let val (c,f) = hd(a)
    in
let val (res, r1, r2) = checkInternal(f,a,b)
in if res = false then false else check(r1,r2)
end
    end;

val test1 = check([],prepareHuffman(""));
val test2 = check([(#"a",7)], prepareHuffman("aaaaaaa"));
val test3 = check([(#"J",1),(#"a",1),(#"c",1),(#"o",1),(#"b",1),(#" ",1),(#"U",1),(#"n",1),
   (#"v",1),(#"e",1),(#"r",1),(#"t",1),(#"y",1),(#"s",2),(#"i",2)], prepareHuffman("Jacobs University"));
val test4 = check([(#"G",1),(#"l",1),(#"o",1),(#"u",1),(#"s",1),(#"i",1),(#"d",1),(#"a",2),
   (#"m",2),(#"p",2),(#"n",3),(#"r",3),(#" ",3),(#"c",3),(#"t",3),(#"e",7)], prepareHuffman("General compute
val test5 = check([(#"d",1),(#"c",2),(#"b",3),(#"a",4)], prepareHuffman("aaaabbbccd"));
```

Problem 6.4: Given the alphabet $A = \{0, \#, @\}$ and a $L := \bigcup_{i=0}^{\infty} L_i$, where 15pt

- $L_0 = \{0\}$
- $L_{i+1} = \{xx\#, x@y \mid x, y \in \bigcup_{k=0}^{i} L_k\}$

Write down all the strings in L_2

Solution:

- $L_2 = \{00\#, 00\#00\#\#, 0@00@0\#, 0@0, 0@00\#, 0@0@0, 00\#@0, 00\#@00\#, 00\#@0@0, 0@0@0, 0@0@00\#, 0@$

Assignment 7: Encodings
(Given Oct. 30., Due Nov. 5.)

Problem 7.1 (UTF encodings)

30pt

UTF encodings are very popular in the modern world. The three main types are UTF-8, UTF-16 and UTF-32.
Your task is to write SML functions:

- **encodeUTF8** that takes a string that represents the Unicode code point of the character(The euro sign is represented as U+20AC) and returns the binary UTF-8 encoding as a string
- **decodeUTF8** that takes a string representing the binary encoding in UTF-8 and returns the Unicode code point as a string.
- **encodeUTF16** that encodes the Unicode code point in UTF-16
- **decodeUTF16** that decodes a binary UTF-16 to a Unicode code point.

```
val encodeUTF8 = fn: string -> string
val encodeUTF16 = fn: string -> string
val decodeUTF8 = fn: string -> string
val decodeUTF16 = fn: string -> string

encodeUTF8("U+20AC");
val it="111000101000001010101100":string

decodeUTF8("111000101000001010101100");
val it="U+20AC":string

encodeUTF16("U+20AC");
val it="0010000010101100":string

decodeUTF16("0010000010101100");
val it="U+20AC":string
```

Please use the following links for referencing how to encode and decode the strings.
`http://en.wikipedia.org/wiki/UTF-8`
`http://en.wikipedia.org/wiki/UTF-16`

Problem 7.2 (Huffman in SML)

35pt

In your last homework you were supposed to do a preparation for the Huffman algorithm, namely you created an ordered list of characters and their frequencies. This time you will need to implement the remainding part of the algorithm.

For this purpose you are supposed to build up on the solution from last week and create a huffman function which should return a list of tuples of characters and their codes.

Basically you should first generate the tree and then get the codes from the tree (See `http://www.siggraph.org/education/materials/HyperGraph/video/mpeg/mpegfaq/huffman_tutorial.html` for an excellent tutorial of how the tree is built).

Your tree should be of datatype huffmantree which is defined as:

```
datatype huffmantree = leaf of (int * char) | node of (huffmantree * huffmantree * int);
```

The signature of huffman should be:

```
val it = fn : string -> (char * string) list
```

An example output is:

```
- huffman("aaaabbbccdd");
val it = [(#"a","0"),(#"b","10"),(#"c","110"),(#"d","111")] : (char * string) list
```

Solution:

```
datatype huffmantree = leaf of (int * char) | node of (huffmantree * huffmantree * int);

(* code from last week *)

fun eliminate(a, nil) = nil
  | eliminate(a, b::r) = if a = b then eliminate(a,r) else b::eliminate(a,r);

fun count(a, nil) = 0
  | count(a, b::r) = if a = b then 1 + count(a, r) else count(a, r);

fun doWork(nil) = nil
  | doWork(h::t) = (h,count(h,h::t))::doWork(eliminate(h,t));

fun findsmallest(nil, a) = a
  | findsmallest((e,c)::r, (e1,c1)) = if (c < c1) then findsmallest(r, (e,c)) else findsmallest(r, (e1,c1));

fun sort(nil) = nil
  | sort(l) = let val x = findsmallest(l,hd(l)) in x::sort(eliminate(x,l)) end;

fun prepareHuffman(string) = sort(doWork(explode(string)));


(* new code *)


fun toHufLeafList(nil) = nil
  | toHufLeafList((a,b)::r) = leaf(b,a)::toHufLeafList(r);

fun getCode(leaf(n,ch),l) = if l <> nil then [(ch,implode(l))] else [(ch,"0")]
  | getCode(node(t1,t2,n),l) = getCode(t1,l@[#"0"]) @ getCode(t2,l@[#"1"]);

fun getFreq(leaf(n,c)) = n
  | getFreq(node(t1,t2,n)) = n;

fun findSmallestHuf(nil, sm) = sm
  | findSmallestHuf(a::r, sm) = if getFreq(a) < getFreq(sm) then findSmallestHuf(r,a) else findSmallestHuf(r,sm);
```

```
fun sortHufList(nil) = nil
  | sortHufList(l) = let val sm = findSmallestHuf(l,hd(l)) in sm::sortHufList(eliminate(sm,l)) end;

fun huffmanHelp(nil) = nil
  | huffmanHelp(a::nil) = getCode(a,nil)
  | huffmanHelp(a::b::l) = huffmanHelp(sortHufList(node(a,b,getFreq(a)+getFreq(b))::l));

fun huffman(st) = huffmanHelp(toHufLeafList(prepareHuffman(st)));
```

Problem 7.3 (Finding a Prefix Code)

Write an SML function prefix_code that takes an original string and an encoded version of this string and returns a prefix code that induces such encoding. The signature of your function should be 35pt

```
val prefix_code = fn : string -> string -> (char * string) list
```

If there is no such prefix code, raise an exception DoesntExist. The original string may contain duplicate characters.

Solution:

```
exception DoesntExist;
exception BadCode;

fun check_prefix [] b = true
  | check_prefix a [] = false
  | check_prefix ((h1:char)::l1) (h2::l2) = if (h1 <> h2) then false
                                           else check_prefix l1 l2;

fun is_prefix_of_any code [] = false
  | is_prefix_of_any code ((_,h)::t) = if (check_prefix code h orelse check_prefix h code)
                                       then true else is_prefix_of_any code t;

fun is_prefix_code [] = true
  | is_prefix_code ((_,h)::t) = if (is_prefix_of_any h t) then false
                                else is_prefix_code t;

fun find_char_code c [] = []
  | find_char_code c ((h,code)::t) = if (h = c) then code else (find_char_code c t);

(* checks that the code fits into the encoded string and returns the rest of the encoded string *)
fun check_next [] encoded = encoded
  | check_next l [] = raise DoesntExist
  | check_next (h1::t1) (h2::t2) = if (h1 = h2) then (check_next t1 t2) else raise DoesntExist;

(* the boolean variable indicates whether a new character code should be started (if true) *)
fun partition b [] [] codelist = if (is_prefix_code codelist) then codelist else raise DoesntExist
  | partition b [] (h::t) codelist = raise DoesntExist
  | partition b (h::t) [] codelist = raise DoesntExist
  | partition false l1 l2 [] = raise DoesntExist
  | partition false (h1::t1) (h2::t2) ((_,code)::ct) =
      (partition true t1 t2 ((h1,code@[h2])::ct)
      handle DoesntExist => partition false (h1::t1) t2 ((h1,code@[h2])::ct))
  | partition true (h1::t1) (h2::t2) codelist =
```

```
        let val code = find_char_code h1 codelist;
          val next = check_next code (h2::t2);
        in
          if (code = []) then partition true t1 t2 ((h1,[h2])::codelist)
                                handle DoesntExist => partition false (h1::t1) t2 ((h1,[h2])::codelist)
                         else partition true t1 next codelist
        end;

fun implode_code [] = []
  | implode_code ((c,list)::t) = (c,implode list)::(implode_code t);

fun prefix_code original encoded = implode_code (partition true (explode original) (explode encoded) []);
```

Solution:

```
(*TEST CASES*)
exception CharacterMissing;
fun find_code c [] = raise CharacterMissing
  | find_code c ((h,code)::t) = if (c = h) then (explode code) else (find_code c t);
fun assemble [] code = []
  | assemble (h::t) code = (find_code h code)@(assemble t code);
fun explode_code [] = []
  | explode_code ((c,list)::t) = (c,explode list)::(explode_code t);
fun check_code original encoded exc = let val code = prefix_code original encoded
                                      in (implode (assemble (explode original) code) = encoded)
                                         andalso is_prefix_code (explode_code code)
                                      end
                                      handle CharacterMissing => false
                                           | DoesntExist => exc;

(* simple codes *)
val test_ok_0 = check_code "" "" false;
val test_ok_1 = check_code "a" "1" false;
val test_ok_2 = check_code "a" "10" false;
val test_ok_3 = check_code "ab" "10" false;
val test_ok_4 = check_code "ab" "101" false;
val test_ok_5 = check_code "ab" "1001" false;
val test_ok_6 = check_code "abc" "101001" false;
val test_ok_7 = check_code "abc" "abc" false;
val test_ok_8 = check_code "abc" "cba" false;
val test_ok_9 = check_code "abcd" "01100011" false;
val test_ok_10 = check_code "ab" "0001" false;
val test_ok_11 = check_code "ab" "1011" false;
(* repeated characters *)
val test_ok_12 = check_code "aab" "110" false;
val test_ok_13 = check_code "abba" "11000011" false;
val test_ok_14 = check_code "hello" "1100101001" false;
val test_ok_15 = check_code "aloha" "1100100111110" false;
val test_ok_16 = check_code "aa" "11" false;
(* nonexistent codes *)
val test_exc_1 = check_code "a" "" true;
val test_exc_2 = check_code "ab" "1" true;
val test_exc_3 = check_code "ab" "11" true;
```

```
val test_exc_4 = check_code "aa" "1110" true;
val test_exc_5 = check_code "ab" "111" true;
val test_exc_6 = check_code "ababab" "1001020010201" true;
val test_exc_7 = check_code "hello" "1100101000101110001" true;
val test_exc_8 = check_code "gencs" "1110010011" true;
val test_exc_9 = check_code "itworks!" "00100111001" true;
```
