1 Maths

10pt

Problem 1.1 (Interval Intersections)
You are given a set of N open intervals $I_1, I_2, \dots I_N$, with the property that:

$$\forall i, j. I_i \cap I_j \neq \emptyset$$

Prove **by induction** that:

$$\forall N \geq 2. I_1 \cap I_2 \cap \dots I_N \neq \emptyset$$

Solution:
Proof: We will prove by induction after N the hypothesis. Note that N is the number of intervals we are dealing with, irrespective to the intervals themselves.

P.1 Base case: For $N = 2$, it is obvious

P.2 Another base case: Consider that we have three intervals ($N = 3$): $I_1 := (a, b)$, $I_2 := (c, d)$, and $I_3 := (e, f)$, and suppose without loss of generality that $a \leq c \leq e$. Because of the hypothesis, we also have $e < b$ and $e < d$ (otherwise some intersections would be empty). It follows that at least the number $\frac{e+\min(d,b)}{2}$ is common to all the intervals.

P.3 Step case: We know that the hypothesis holds for N intervals, we want to prove it for $N+1$ intervals. Let us consider intervals $I_1, \dots I_{N+1}$ and define

$$J_k := I_k \cap I_{N+1}$$

From the hypothesis of the problem, we know that any two intervals have a common element, thus

$$\forall k. J_k \neq \emptyset$$

We also know that

$$J_{k_1} \cap J_{k_2} \equiv I_{k_1} \cap I_{k_2} \cap I_{N+1}$$

However, we know from the previous step that any three intervals with the property from the problem have are all together not disjunct, therefore:

$$\forall k_1, k_2. J_{k_1} \cap J_{k_2} \neq \emptyset$$

Therefore, all the N intervals $J_1 \dots J_N$ have the property from the statement, and thus from the inductive hypothesis we have that

$$J_1 \cap J_2 \cap \dots J_N \neq \emptyset$$

which can be rewritten as

$$I_1 \cap I_2 \cap \dots \cap I_N \cap I_{N+1} \neq \emptyset$$

□

---

2 Abstract Data Types

10pt

Problem 2.1 (ADT for UNN and prime numbers)
Design an ADT for unary natural numbers. Write a procedure that checks whether a number is prime.

Solution: The ADT is: $\langle\{\mathbb{N}\},\{[o\colon\mathbb{N}],[s\colon\mathbb{N}\to\mathbb{N}]\}\rangle$

The *cmp* procdure compares two unary natural numbers, the o procedure represents bigger or equal, $s(o)$ represents smaller:
$\langle cmp{::}\mathbb{N}\times\mathbb{N}\to\mathbb{N}\,;\,\{cmp(x,o)\rightsquigarrow o, cmp(o,x)\rightsquigarrow s(o), cmp(s(x),s(y))\rightsquigarrow cmp(x,y)\}\rangle$
$\langle sub{::}\mathbb{N}\times\mathbb{N}\to\mathbb{N}\,;\,\{sub(o,x)\rightsquigarrow o, sub(x,o)\rightsquigarrow x, sub(s(x),s(y))\rightsquigarrow sub(x,y)\}\rangle$
$\langle if{::}\mathbb{N}\times\mathbb{N}\times\mathbb{N}\to\mathbb{N}\,;\,\{if(o,x,y)\rightsquigarrow y, if(s(o),x,y)\rightsquigarrow x\}\rangle$

The *isDiv* procedure checks whether a number is divisible by another:
$\langle isDiv{::}\mathbb{N}\times\mathbb{N}\to\mathbb{N}\,;\,\{isDiv(o,x)\rightsquigarrow s(o), isDiv(s(x),y)\rightsquigarrow if(cmp(s(x),y),o,isDiv(sub(s(x),y),y))\}\rangle$
The next procedure iterates over possible divisors:
$\langle check{::}\mathbb{N}\times\mathbb{N}\to\mathbb{N}\,;\,\{check(x,k)\rightsquigarrow if(cmp(k,x),if(isDiv(x,k),o,check(x,s(k))),,s(o))\}\rangle$
The last procedure checks whether a number is prime:
$\langle isPrime{::}\mathbb{N}\to\mathbb{N}\,;\,\{isPrime(s(o))\rightsquigarrow o, isPrime(x)\rightsquigarrow check(x,s(s(o)))\}\rangle$

---

3 Standard ML

20pt

Problem 3.1 (Game)
Four players A,B,C,D are playing the following game: They have a number of red and green stones and one blue stone arranged in a circle. (We will represent the circle by a list). The players perform the following actions in turn:
Player A replaces the first red stone after the blue stone by a green stone.

```
For example: [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘r’’,#‘‘r’’]
would become [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘g’’,#‘‘r’’]
```

Player B shifts the blue stone to the clockwise (to the right) by 3 replacing all the red sones he finds by green stones, if he reaches the end of the “list” he starts at the beginning:

```
For example: [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘r’’]
would become [#‘‘b’’,#‘‘r’’,#‘‘g’’,#‘‘g’’,#‘‘g’’]
```

Player C changes the stone after the blue stone to a green stone:

```
For example: [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘r’’]
would become [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘r’’]

             [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘r’’,#‘‘r’’]
would become [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘r’’]
```

Player D shifts the blue stone to the left (counter clockweise) by 1, and puts a green stone in it’s original place:

```
For example: [#‘‘r’’,#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘r’’]
would become [#‘‘r’’,#‘‘b’’,#‘‘g’’,#‘‘g’’,#‘‘r’’]
```

The player who replaces the last red stone by a green stone wins.
Assuming player A starts first, and the players play in the order A,B,C,D, write a sml function that given the list with the arrangement of stones, determines which of the players will win, and how many moves player A makes. Don't forget to raise the appropriate exceptions.

Example and signature:

```
val game = fn : char list -> string * int
- game([#"r",#"g",#"r",#"b",#"r",#"g"]);
val it = ("A␣wins",2) : string * int
```

Solution:

```
exception wrong_stone;

fun last_element ([]) = raise wrong_stone (*find the last element of the list *)
  | last_element ([a]) = ([],a) (*and the list without the last element*)
  | last_element(a::l) = let val (c,d) = last_element(l) in (a::c,d) end;

fun count([]) = true (*check if there are only blue and green stones*)
  | count((#"r")::l) = false
  | count((#"b")::l) = count(l)
  | count((#"g")::l) = count(l)
  | count(r::l) = raise wrong_stone;

fun help_a(m,(#"r")::l,0) = help_a(m@[#"r"],l,0) (*move of player A*)
  | help_a(m,(#"g")::l,0) = help_a(m@[#"g"],l,0) (*find the blue stone*)
  | help_a(m,[(#"b")],0) = help_a([],m@[(#"b")],1) (*if it is at the end go to the beginning*)
  | help_a(m,(#"b")::l,0) = help_a(m@[#"b"],l,1) (*now we have found the blue stone*)
  | help_a(m,[(#"g")],1) = help_a([],m@[(#"g")],1)(*if the end of the list is reached*)
  | help_a(m,(#"g")::l,1) = help_a(m@[#"g"],l,1) (*search for a red stone*)
  | help_a(m,(#"r")::l,1) = m@[(#"g")]@l (*found it, return*)
  | help_a(m,[],0) = raise wrong_stone (*no blue stone*)
  | help_a(m,(#"b")::l,1) = raise wrong_stone (*two blue stones*)
  | help_a(m,y::l,x) = raise wrong_stone; (*some other stone*)

fun help_b(m,[(#"g")],0) = raise wrong_stone(*move of player B*)
  | help_b(m,[(#"r")],0) = raise wrong_stone (*no blue stone*)
  | help_b(m,(#"r")::l,0) = help_b(m@[(#"r")],l,0) (*search for blue stone*)
  | help_b(m,(#"g")::l,0) = help_b(m@[(#"g")],l,0) (*search for blue stone*)
  | help_b(m,[(#"b")],0) = help_b([],m@[(#"g")],1) (*if the blue stone is at the end of the lis
  | help_b(m,(#"b")::l,0) = help_b(m@[(#"g")],l,1) (*found blue stone*)
  | help_b(m,(#"g")::l,3) = m@[(#"b")]@l (*made 3 steps, so shift the blue stone, return*)
  | help_b(m,(#"r")::l,3) = m@[(#"b")]@l (*made 3 steps, so shift the blue stone, return*)
  | help_b(m,[(#"r")],x) = help_b([],m@[(#"g")],x+1) (*end of list, ruturn to the baginning of
  | help_b(m,[(#"g")],x) = help_b([],m@[(#"g")],x+1) (*end of list, ruturn to the baginning of
  | help_b(m,(#"g")::l,x) = help_b(m@[(#"g")],l,x+1) (*go along the list, changing the stones*)
  | help_b(m,(#"r")::l,x) = help_b(m@[(#"g")],l,x+1) (*go along the list, changing the stones*)
  | help_b(m,a::l,x) = raise wrong_stone; (*some other stone*)

fun help_c(m,(#"r")::l,0) = help_c(m@[#"r"],l,0) (*move of player C*)
```

```
  | help_c(m,(#"g")::l,0) = help_c(m@[#"g"],l,0) (*search for the blue stone*)
  | help_c(m,[],0) = raise wrong_stone (*no blue stone*)
  | help_c(m,[(#"b")],0) = help_c([],m@[(#"b")],1) (*blue stone last in the list, so return to
  | help_c(m,(#"b")::l,0) = help_c(m@[#"b"],l,1) (*found blue stone*)
  | help_c(m,(#"g")::l,1) = m@[(#"g")]@l (*replace the stone after the blue stone*)
  | help_c(m,(#"r")::l,1) = m@[(#"g")]@l (*replace the stone after the blue stone*)
  | help_c(m,(#"b")::l,1) = raise wrong_stone (*two blue stones*)
  | help_c(m,y::l,x) = raise wrong_stone; (*some other stone*)

fun help_d(m,(#"r")::l) = help_d(m@[(#"r")],l) (*move of player D*)
  | help_d(m,(#"g")::l) = help_d(m@[(#"g")],l)(*search for the blue stone*)
  | help_d([],(#"b")::l) = let val (a,b) = last_element(l) in [(#"g")]@a@[#"b"] end (*if the bl
  | help_d(m,(#"b")::l) = let val (a,b) = last_element(m) in a@[#"b"]@[(#"g")]@l end (*replace
  | help_d(m,[]) = raise wrong_stone (*no blue stone*)
  | help_d(m,a::l) = raise wrong_stone; (*some other stone*)

fun game_a(l,x) = let val c = help_a([],l,0) in if count(c) then ("A␣wins", x+1) else game_b(c
(*play in turn*)
and game_b(l,x) = let val c = help_b([],l,0) in if count(c) then ("B␣wins", x+1) else game_c(c
and game_c(l,x) = let val c = help_c([],l,0) in if count(c) then ("C␣wins", x+1) else game_d(c
and game_d(l,x) = let val c = help_d([],l) in if count(c) then ("D␣wins", x+1) else game_a(c,x

fun game(l) = game_a(l,0); (*player A starts*)
```

Problem 3.2 (Sum decomposition) 10pt

Design an SML function that takes an integer $n > 0$ and returns all the possible ways in which n can be written as sum of strictly positive integers. Encode the result as a string.

Function signature and example:

```
val decompose = fn : int -> string list
- decompose 3;
val it = ["3","2␣+␣1","1␣+␣2","1␣+␣1␣+␣1"] : string list
```

How many decompositions exist for an n ? (Write your answer and a short argument at the end of the source file)

Solution:

```
Control.Print.printLength := 1000;

fun append x ll = map (fn ls => x :: ls) ll
fun addOne ll = map (fn (h :: t) => (1 + h) :: t) ll

fun decomposeInList 1 = [[1]]
    |decomposeInList n =
       let val ll = decomposeInList (n - 1)
       in addOne ll @ append 1 ll
       end

fun convertToString [h] = Int.toString h
    | convertToString (h :: t) = Int.toString h ^ "␣+␣" ^ convertToString t

fun decompose n = map convertToString (decomposeInList n)
```

---

4 Formal Languages

Problem 4.1 (Formal Languages) 10pt

You are given the alphabet $A = \{a, b, c\}$ and a $L := \bigcup_{i=0}^{\infty} L_i$, where $L_0 = \{a\}$ and $L_{i+1} = \{xxb, xcy \mid x, y \in \bigcup_{k=0}^{i} L_k\}$.

1. Determine the cardinality of L_2, **without** explicitly writing down the strings it contains.
2. For each of the strings below, determine whether it is in L. Explain why or why not!
   - $s_1 = accca$
   - $s_2 = acca$
   - $s_3 = acacaab$

Solution:

1. $L_1 = aab, aca$. Thus $\bigcup_{k=0}^{1} L_k = a, aab, aca$ with cardinality 3.
   When constructing L_2, we will get 3 strings from xxb, since x is the same string.
   In addition, we will get $3 \cdot 3 = 9$ strings from xcy, since x and y are different.
   However, in this way we will get $acaca$ twice: from xcy with a and aca, and from aca and a. There are no other such symmetries, so there are no other repetitions.
   So the cardinality of L_2 is $3 + 9 - 1 = 11$.
2. First we note that the number of characters in each string is always an odd number.
   - s_1 is not in L, because the construction rules do not allow two consecutive occurrances of the character c.
   - s_2 is not in L, because there cannot be strings with an even number of characters.
   - s_3 is in L: starting from the empty string, we get a, from there we get aca via xcy and aab via xxb, and then we get $acacaab$ via xcy.

Problem 4.2 (Code definitions) 7pt

Define the following concepts and give an example of each:

1. Character code.
2. String code.
3. Prefix code.

Why are prefix codes also string codes?

Solution:

1. Let A and B be alphabets, then we call an injective function c from A to B^+ a character code

2. Let c' be a function from A^* to B^* if it is injective then it induces a string code
3. A (character) code c : A to B+ is a prefix code if none of the codewords is a proper prefix to an other codeword

Proof of a prefix code being a string code in slide 130.

Problem 4.3 (Formal Languages and Concatenation and Intersection) 7pt

Given the alphabet $A = \{a, b\}$ and 3 formal languages in A $L_1 = \{a^{[n]} \,|\, n \in \mathbb{N}\}$, $L_2 = \{ba^{[n]} \,|\, n \in \mathbb{N}\}$, $L_3 = \{b^{[k]}a^{[2n]} \,|\, n \in \mathbb{N}, k \in \mathbb{N}\}$.

1. What is $L_1 \cap L_3$?
2. Write down three words that belong in $L_4 = \mathrm{conc}(L_2, L_1)$.

Solution:

1. $\{a^{[2n]} \,|\, n \in \mathbb{N}\}$
2. baa, b, baaa

5 Boolean Expressions

Problem 5.1 (Practising Quine McCluskey) 7pt

Use the algorithm of Quine-McCluskey to determine the minimal polynomial of the following function:

$x1$	$x2$	$x3$	f
F	F	F	F
F	F	T	F
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	T

Solution:

QMC_1 :

$$\begin{aligned} M_0 &= \{\overline{x_1}\, x_2\, \overline{x_3}, x_1\, \overline{x_2}\, \overline{x_3}, x_1\, \overline{x_2}\, x_3, x_1\, x_2\, x_3\} \\ M_1 &= \{x_1\, \overline{x_2}, x_1\, x_3\} \\ P_1 &= \{\overline{x_1}\, x_2\, \overline{x_3}\} \\ M_2 &= \emptyset \\ P_2 &= \{x_1\, \overline{x_2}, x_1\, x_3\} \end{aligned}$$

QMC_2 :

	FTF	TFF	TFT	TTT
$x_1\, \overline{x_2}$	F	T	T	F
$x_1\, x_3$	F	F	T	T
$\overline{x_1}\, x_2\, \overline{x_3}$	T	F	F	F

Final result: 1. $f = x_1 \overline{x_2} + x_1 x_3 + \overline{x_1} x_2 \overline{x_3}$

Problem 5.2 (Model for Boolean Expressions) 7pt

Give a variable assignment φ for which all the following expressions evaluate to true.

1. $e_1 := x_1 * \overline{x_2} + \overline{x_2 + x_3} * \overline{x_1 + \overline{x_3}}$
2. $e_2 := \overline{x_1} * (x_2 * \overline{x_3}) + x_1 * (\overline{x_2} * x_3)$
3. $e_3 := (x_1 + x_2) * (x_2 + x_3)$

Show your reasoning using truth tables.

Solution:
$\varphi = \{[T/x_1], [F/x_2], [T/x_3]\}$
We have the truth tables

x_1	x_2	x_3	$x_1 + x_2$	$x_2 + x_3$	e_3	$\overline{x_1} * (x_2 * \overline{x_3})$	$x_1 * (\overline{x_2} * x_3)$	e_2
T	T	T	T	T	T	F	F	F
T	T	F	T	T	T	F	F	F
T	F	T	T	T	T	F	T	T
T	F	F	T	F	F	F	F	F
F	T	T	T	T	T	F	F	F
F	T	F	T	T	T	T	F	T
F	F	T	F	T	F	F	F	F
F	F	F	F	F	F	F	F	F

From this table we find two possible assignments (the rows for which both e_2 and e_3 are true): $\varphi_1 = \{[T/x_1], [F/x_2], [T/x_3]\}$ and $\varphi_2 = \{[F/x_1], [T/x_2], [F/x_3]\}$.

Evaluating e_1 under φ_1 and φ_2 reveals that φ_1 is in fact the only solution. The truth table for e_1 is omitted.

---

6 Propositional Logic

Problem 6.1 (Hilbert calculus) 7pt

Prove the following theorem of Hilbert Calculus (using Hilbert Calculus rules only!!! - and make sure you specify the rules used on the way)

$$(S \Rightarrow R) \Rightarrow S \Rightarrow S \Rightarrow R$$

Solution:

P.1 $(S \Rightarrow R \Rightarrow (S \Rightarrow R)) \Rightarrow (S \Rightarrow R) \Rightarrow S \Rightarrow S \Rightarrow R$ (**S** with $[S/P]$, $[R/Q]$, $[S \Rightarrow R/R]$)

P.2 $R \Rightarrow S \Rightarrow R$ (**K** with $[R/P]$, $[S/Q]$)

P.3 $(R \Rightarrow S \Rightarrow R) \Rightarrow S \Rightarrow (R \Rightarrow S \Rightarrow R)$ (**K** with $[R \Rightarrow S \Rightarrow R/P]$, $[S/Q]$)

P.4 $S \Rightarrow R \Rightarrow S \Rightarrow R$ (**MP** on **P.3** and **P.2**)

P.5 $(S \Rightarrow R) \Rightarrow S \Rightarrow S \Rightarrow R$ (**MP** on **P.1** and **P.4**)

Problem 6.2 (Natural deduction) 10pt

Prove the following theorem of Natural Deduction (using ND Calculus rules only!!! - give their short abbreviation too when applying them)

$$(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)$$

Solution:

$$[P \Rightarrow Q]^1$$

$$\dfrac{\dfrac{\dfrac{}{\neg P \vee P}TND \qquad \dfrac{\begin{array}{c}[\neg P]^2\\ \neg P\end{array}}{\neg P \vee Q}\vee I_l \qquad \dfrac{\dfrac{P \Rightarrow Q \qquad \begin{array}{c}[P]^2\\ P\end{array}}{Q}\Rightarrow E}{\neg P \vee Q}\vee I_r}{\neg P \vee Q}\vee E^2}{(P \Rightarrow Q) \Rightarrow (\neg P \vee Q)}\Rightarrow I^1$$
