# General Computer Science II (320102) Spring 2014

Michael Kohlhase
Jacobs University Bremen

August 9, 2014

## Contents

# Assignment 1: Graph Theory and Circuits (Given Feb. 5., Due Feb. 12.)

**Problem 1.1 (In-degrees in acyclic digraphs)**
Prove by induction or refute that any acyclic digraph with non-empty set of nodes has at least one node with in-degree 0. 30pt

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

**P.1.1 n=1**: trivial

**P.1.2 Step case:** $n \Longrightarrow n+1$:

**P.1.2.1** 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.

**P.1.2.2** So assume an acyclic digraph with n nodes and at least a node with in-degree 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.

**P.1.2.3.1 This new node has no incoming edge.**: This is the new 0 in-degree node. □

**P.1.2.3.2 $v_0$ still has no incoming edge.**: $v_0$ remains the 0 in-degree node. □

**P.1.2.3.3 $v_0$ has an incoming edge from $v_n$ and indeg$(v_n) > 0$**: Then there exists an edge from $v_i$ to $v_n$. If $v_i$ has in-degree 0, then this is a 0 in-degree node. 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. Continuing this process either leads to identifying a 0 in-degree node or to the situation where a last untouched node is reached but the graph is acyclic, so this node has to have in-degree 0. □

□

□

**Problem 1.2 (Depth of a Fully Balanced Binary Tree)**
Prove or refute that in a fully balanced binary tree with $n \geq 1$ nodes, the depth is $\lceil log(n) \rceil$. 15pt

**Solution:**
**Proof**: by induction over $n$

**P.1.1 $n = 1$ (base case)**: $\log_2(1) = 0$, the depth □

**P.1.2 $n \to n+1$ (induction step)**: $n+1$ leaves are added (because fully balanced), so we have $\log_2(2n+1)$.

$\log_2(2n) = (\log_2(n)) + 1$.

Since $2n+1$ is not a power of 2, $\log_2(2n+1) = \log_2(2n) = 1 + (\log_2(n))$, which by I.H is depth+1 . □

□

**Problem 1.3 (Generate Boolean Tree)**
Write an SML function MakeBTree which takes an odd integer $n$ and returns a balanced binary tree with $n$ nodes of the type datatype btree = leaf | parent of btree*btree 25pt

**Solution:**

```
datatype btree = leaf | parent of btree*btree;

exception WrongInput;
Compiler.Control.Print.printDepth := 20;

(* a. Makes a fully balanced binary tree with n nodes. n must of form (2^n)−1, n>0 *)

fun MakeFBTree 0 = raise WrongInput |
        MakeFBTree 1 = leaf |
    MakeFBTree n = let
        val temp = MakeFBTree ((n−1) div 2)
        in
         parent(temp,temp)
        end;

(* b. Count nodes of a binary tree. *)

fun CountNodes leaf = 1 |
    CountNodes (parent(x,y)) = CountNodes x + CountNodes y + 1;

(* c. Makes a balanced binary tree with n nodes. n must be odd number >0. *)

 fun tlog2 1 = 0 |
     tlog2 n = 1 + tlog2(n div 2);
 fun pow2 0 = 1 |
     pow2 n = 2*pow2(n−1);

 fun make(1, 0) = (leaf,0) |
     make(1, m) = (parent(leaf, leaf), m−2) |
     make(n, m) = let
       val (t1, l1) = make(n−1,m)
       val (t2, l2) = make(n−1,l1)
      in (parent(t1,t2), l2)
     end

fun MakeBBTree 1 = leaf |
        MakeBBTree n = let
         val lvl = tlog2 n (* lvl > 0 *)
         val lf = n − pow2 lvl + 1
         val (tree, left) = make(lvl, lf)
         in if left = 0 then tree
          else raise WrongInput
        end
```

**Problem 1.4:** Using Quine McCluskey and shared monomials make a circuit for calcu- 30pt lating the values of $f_1$ and $f_2$, given by the following table, for input $(x_1, x_2, x_3)$.

| $x_1$ | $x_2$ | $x_3$ | $f_1$ | $f_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |

**Solution:**

# Assignment 2: Combinatorial Circuits and PNS (Given Feb. 12., Due Feb. 19.)

**Problem 2.1 (Converting to decimal in SML)** 20pt

Write an SML function to_int = fn : string −> int that takes a string in binary, octal or hexadecimal notation and converts it to a decimal integer. If the string represents a binary number, it begins with 'b' (e.g. b1011), if it is an octal number - with '0' (e.g. 075) and if it is a hexadecimal number it begins with '0x' (e.g. 0x3A).

If the input does not represent an integer in one of these three forms raise the InvalidInput exception.

For example we have to_int("b101010") −> 42

---

**Solution:**

```
exception InvalidInput;

fun reverse nil = nil
| reverse (h::t) = reverse(t)@[h];

fun find_int(#"A") = 10
|find_int(#"B") = 11
    |find_int(#"C") = 12
    |find_int(#"D") = 13
    |find_int(#"E") = 14
    |find_int(#"F") = 15
    |find_int(c) = if 0<= ord(c)− 48 andalso ord(c)−48 <=9 then ord(c)−48 else raise InvalidInput;

fun from_binary (nil) = 0
| from_binary (h::l) = if 0<= ord(h)− 48 andalso ord(h)−48 <=1
     then ord(h) − 48 + 2 * from_binary(l) else raise InvalidInput;

fun from_octal (nil) = 0
|from_octal (h::l) = if 0<= ord(h)− 48 andalso ord(h)−48 <=8
    then ord(h) − 48 + 8 * from_octal(l) else raise InvalidInput;

fun from_hexa (nil) = 0
|from_hexa (h::l) = find_int(h) + 16 * from_hexa(l);

fun selector (#"0"::(#"x"::l)) = from_hexa(reverse(l))
| selector (#"0"::l) = from_octal(reverse(l))
    | selector (#"b"::l) = from_binary(reverse(l))
    | selector _ = raise InvalidInput;

fun to_int number = selector(explode(number));

(*TEST CASES*)
val test1 = to_int("b101010")=42;
val test2 = to_int("052")=42;
val test3 = to_int("0x2A")=42;
val test4 = to_int("0x11A")=282;
val test5 = to_int("b101101")=45;
val test6 = to_int("12") = 0 handle InvalidInput => true| other => false;
val test7 = to_int("b12") = 0 handle InvalidInput => true| other => false;
val test8 = to_int("0x12H") = 0 handle InvalidInput => true| other => false;
val test9 = to_int("0129") = 0 handle InvalidInput => true| other => false;
val test10 = to_int("0−") = 0 handle InvalidInput => true| other => false;
```

**Problem 2.2:** Design a two bit multiplier using only NOT, OR, AND and XOR gates. 10pt

**Solution:** The solution has 8 gates.

**Problem 2.3 (6-ary AND gate)**

Using only normal AND gates draw the inside of a 6-ary AND gate. What is the minimal depth of such a circuit and why? 5pt

**Solution:**

**Problem 2.4 (Conditional circuit)**

Design a 4-bit conditional circuit that implements the following operation: 20pt

if $x \leq y$ then $x + y$ else $2x$

**Problem 2.5 (Periodic Number Representation)**

Prove that if the base-$b$ representation of a real number $x = x_0.x_1x_2x_3\ldots_b$ is periodic then x is a rational number. The representation is periodic if there are positive integers $n$ and $p$ such that $x_k = x_{k+p}$ for all integers $k > n$. For example, $10.\overline{11}_2 = 2\frac{3}{4}$, $0.\overline{112}_3 = 0.112112112\ldots_3 = \frac{7}{13}$, and $2.01\overline{2}_8 = 2.012222222\ldots_8 = 2\frac{9}{448}$. 35pt

# Assignment 3: Combinatorial Circuits and PNS (Given Feb. 20., Due Feb. 26.)

**Problem 3.1 (Incremental Adder)**

25pt

Design a circuit that recursively adds consecutive numbers, i.e. it first adds 1, then 2, then 3 etc. The starting number is zero and all the numbers are represented with 4 bits. There is no need to deal with overflows.
The outputs should be 1, 3, 6, 10, 15, 5 . . . etc.

**Problem 3.2 (Detecting overflow in TCN addition)**

10pt

1. Convert the following pairs of decimal numbers into (4-bit) two's complement notation and add them (to obtain a 4-bit result). Convert the sum back to decimal and check whether the answer you obtained is correct. Thus state whether the addition was proper or an overflow/underflow occured.
   - $-1$ and $7$
   - $-5$ and $-4$
2. According to the TCN Main Theorem, when adding two $n$-bit TCN numbers, a proper sum is obtained when the last two carry bits are equal. Otherwise an overflow or an underflow occurred. There are a number of other ways to detect that, however. Consider the full adder used for the addition of the most significant bits of the two numbers. Look at the inputs and outputs of this full adder, and determine when an overflow occurs depending on the carry-in and carry-out bit values. Write down the logic equation for this and draw its corresponding combinatorial circuit.

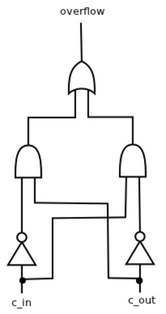
---

**Solution:**

1. 4-bit TCN additions:
   - $-1_{10} = 1111_{TCN}$, $7_{10} = 0111_{TCN}$, $1111_{TCN} + 0111_{TCN} = 0110_{TCN} = 6_{10}$, thus the addition was proper
   - $-5_{10} = 1011_{TCN}$, $-4_{10} = 1100_{TCN}$, $1011_{TCN} + 1100_{TCN} = 0111_{TCN} = 7_{10} \neq -9_{10}$, thus an overflow occured
2. We write down all the possibilities for what is happening at the last full adder for two $n$-bit TCN numbers:

| $a_{n-1}$ | $b_{n-1}$ | $c_{in}$ | $c_{out}$ | $s_{n-1}$ | overflow? |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

From the table we see that an overwflow occurs when $c_{out} \neq c_{in}$.
Thus the corresponding expression is $c_{in} \cdot \overline{c_{out}} + \overline{c_{in}} \cdot c_{out}$.
And the corresponding circuit is below:



**Problem 3.3 (TCN in SML)**

Given the datatype: type tcn = int list write the following SML functions 20pt

- extend = fn : tcn −> int −> tcn which takes a number in Two's complement representation and an integer $n$ and makes the tcn number $n$ bits wide. Here $n$ should always be bigger or equal to the current width of the number.
- int2tcn = fn : int −> int −> tcn which converts an integer number to a tcn number given the width.
- tcn2int = fn : tcn −> int which converts a tcn number to an integer number.

If in any of the functions a number can't fit into the requested number of bits raise the NumberDoesNotFit exception.

**Note:** A number's width is simply the number of bits that are used to represent that number.

As an example conisder

int2tcn 10 8 −> [0,0,0,0,1,0,1,0]

**Solution:**

```
type tcn = int list;
exception NumberDoesNotFit;

fun int2bin 0 = [0]
| int2bin n = (int2bin (n div 2) ) @ [n mod 2];
```

```
fun negate bin = foldr (fn(c,p)=> (1−c)::p ) nil bin;

fun extend (num:tcn) (bits:int) :tcn =
if bits < (length num)
then raise NumberDoesNotFit
else List.tabulate(bits−(length num), (fn _ => hd(num))) @ num;

fun int2tcn n (bits) : tcn =
if n >= 0
then extend ( int2bin n ) bits
else extend ( negate( int2bin ((~n)−1) ) ) bits;

fun bin2int num = foldl (fn (c,p) => p*2+c ) 0 num;
fun tcn2int (num:tcn) =
if hd(num) = 0
then bin2int num
else ~(bin2int (negate num) ) − 1;

(*TEST CASES*)
val test1 = (int2tcn 0 4) = [0,0,0,0];
val test2 = (int2tcn 1 4) = [0,0,0,1];
val test3 = (int2tcn 2 4) = [0,0,1,0];
val test4 = (int2tcn 3 4) = [0,0,1,1];
val test5 = (int2tcn 4 4) = [0,1,0,0];
val test6 = (int2tcn 5 4) = [0,1,0,1];
val test7 = (int2tcn 6 4) = [0,1,1,0];
val test8 = (int2tcn 7 4) = [0,1,1,1];
val test9 = (int2tcn ~1 4) = [1,1,1,1];
val test10 = (int2tcn ~2 4) = [1,1,1,0];
val test11 = (int2tcn ~3 4) = [1,1,0,1];
val test12 = (int2tcn ~4 4) = [1,1,0,0];
val test13 = (int2tcn ~5 4) = [1,0,1,1];
val test14 = (int2tcn ~6 4) = [1,0,1,0];
val test15 = (int2tcn ~7 4) = [1,0,0,1];
val test16 = (int2tcn ~8 4) = [1,0,0,0];
val test17 = (int2tcn ~10 4) = [1,0,1,0] handle NumberDoesNotFit => true| other => false;
val test18 = (int2tcn 16 4) = [1,0,1,0] handle NumberDoesNotFit => true| other => false;

val test19 = (extend [0] 3) = [0,0,0];
val test20 = (extend [1,0] 3) = [1,1,0];
val test21 = (extend [0,1,0] 3) = [0,1,0];
val test22 = (extend [1] 3) = [1,1,1];
val test23 = (extend [1,0,0] 3) = [1,0,0];
val test24 = (extend [1,0,1,0] 3) = [0] handle NumberDoesNotFit => true| other => false;

val test25 = tcn2int [0] = 0;
val test26 = tcn2int [0,0,0] = 0;
val test27 = tcn2int [0,0,1] = 1;
val test28 = tcn2int [1] = ~1;
val test29 = tcn2int [1,1,1,1] = ~1;
val test30 = tcn2int [1,1,1,1,1,1,1,1,0] = ˜2;
val test31 = tcn2int [0,0,0,0,1,1,0,0] = 12;
val test32 = tcn2int [1,0,1,0] = ˜6;
```

---

## Problem 3.4 (Accumulator Circuit)

To finish off your work on designing circuits, here is an all-encompassing assignment. You have to design an accumulator. 35pt

It should be implemented as a circuit which takes an 8-bit input value, together with a 2-bit control input. When this control input is "off" (i.e. its value is 00), the accumulator should store the 8-bit input value. When the control input is 10, the accumulator should add the 8-bit input value to the value previously stored in it, and save the new result. Similarly, when the control input is 11, the accumulator should subtract the 8-bit value from its old value. You are supposed to use a CSA for this problem.

**Note:** Since this is a problem that is supposed to summarize all your knowledge on circuits from GenCS, please create your circuit out of gates. You will need building blocks like flip-flops and adders, but please draw their detailed circuits (you can copy from the slides), and then use some short notation for them.

**Solution:** The required building blocks are a 3 : 1 (4 : 1) 8-bit multiplexer that uses the control bits to choose between the value previously stored in the accumulator, and the new sum or difference that has to be stored.

Next, a D-flip-flop is needed. Then a register consisting of 8 flip-flops will be used to store the ACC value.

In addition, a full-adder and an 8-bit adder are needed. The 8-bit adder will be used to compute the sum of the old ACC value plus the new input. It will also be used to execute the subtraction: the input bits will be inverted, and this number will be added to 1 plus the old ACC value.

# Assignment 4: ASM Language (Given Feb. 27., Due Mar. 6.)

**Problem 4.1:** Write an `ASM` program that multiplies numbers at addresses 0 and 1 and stores the product at address 2. Show intermediate states of registers/memory. 10pt

**Solution:**

**Problem 4.2 (Fibonacci Numbers)**

Write an `ASM` program that computes the $n^{\text{th}}$ Fibonacci number in the accumulator, where $n$ is the value in cell 0. You do not have to worry about sizes of memory cells here. Just assume that there is enough space. 20pt

**Problem 4.3 (Longest equal sublist)**

Write an `ASM` program which determines the length of the longest sequence of consecutive equal numbers from a given list of $n$ numbers and the number itself. 30pt

For example, the answer for this list: $1, 1, 2, 5, 2, 2, 2, 6, 6$ is length 3 and number 2.

$D(0)$ contains the number of elements in the list. It is guarantied that the list contains at least one element. $D(10)$ through $D(10 + n - 1)$ contain the elements of the list. At the end of the execution, the length must be saved in $D(1)$ and the number in $D(2)$

Note that you are only allowed to use labels in jump statements. You are very much encouraged to write comments and even a description of your approach, which can help the grader award partial points.

You can find a list of `ASM` commands (only for reference) on the right.

| | |
|---|---|
| load i | jump(=) i |
| store i | jump(>) i |
| loadi i | jump(<) i |
| addi i | jump(≥) i |
| subi i | jump(≤) i |
| add i | sub i |
| move S T | jump(≠) i |
| loadin i j | storein i j |
| jump i | stop 0 |

**Solution:** Algorithm: you determine the longest such sequence by calculating the answer for the first $1, 2, 3, \ldots, n$ numbers in the list. The base case is clear. The answer is 1. How do we determine for the step cases? It can easily be done by keeping also the length of the equal sublist ending at the last position. So at the next step, we just check if the new number is different from the previous one, then a sequence has ended. After each step, we check if the current maximum has been exceeded and update if necessary.

The following solution is not written with labels so that it can be run by the online tool.

In $D(2)$ is stored the position of the current number we are looking at. In $D(3)$ is stored the length of the nondecreasing sequence ending at the current number, in $D(4)$ is stored the previous number and in $D(5)$ - the current number. Finally, int $D(6)$ we store the result that we later move to $D(2)$.

```
loadi 1
store 1
store 2
store 3
load 10
store 4
load 3
addi 1
```

```
store 3
load 0
sub 2
jump(=) 22
load 2
move acc in1
loadin 1 10
store 5
sub 4
jump(=) 3
loadi 1
store 3
load 3
sub 1
jump(<=) 5
load 3
store 1
load 5
store 6
load 5
store 4
load 2
addi 1
store 2
jump −26
load 6
store 2
stop 0
```

**Problem 4.4 (Binary to decimal)**

Let $D(0) = n$ contain the number of bits of a binary number stored in $D(2) \ldots D(2 + n - 1)$. 40pt
Each memory cell represents one bit of the number where $D(2)$ is the least significant bit and $D(2 + n - 1)$ is the most significant bit. Write a program that stores the corresponding decimal number in $D(1)$.

**Solution:**

| label | instruction | comment |
|---|---|---|
| | `LOAD` 0 | |
| | `MOVE` $ACC\ IN$ | The recursion index. Initializing $IN := n$ |
| | `LOADI` 0 | |
| | `STORE` 1 | Initialize $D(1) := 0$ |
| | | |
| $\langle loop\rangle$ | `MOVE` $IN\ ACC$ | |
| | `JUMP`$_=$ $= \langle end\rangle$ | if $IN$ becomes 0 we are done. |
| | `LOAD` 1 | |
| | `ADD` 1 | |
| | `STORE` 1 | $D(1) := 2 \cdot D(1)$ |
| | `LOADIN` 1 1 | |
| | `ADD` 1 | |
| | `STORE` 1 | $D(1) := D(1) + D(IN+1)$ |
| | `MOVE` $IN\ ACC$ | |
| | `ADDI` $-1$ | |
| | `MOVE` $ACC\ IN$ | $IN--$ |
| | `JUMP` $\langle loop\rangle$ | go to next iteration. |
| | | |
| $\langle end\rangle$ | `STOP` 0 | |

# Assignment 5: ASM Language (Given Mar. 5., Due Mar. 12.)

**Problem 5.1** (Convert Highlevel Code to $\mathcal{L}$(VM) Code)

Given is an array A[0..10] and the following piece of imperative code: 20pt

```
for j := 1 to 5 do
 for i := j to 10−j do
  A[i] := A[i−j] + A[i+j];
```

Suppose the array is loaded on stack (top value being A[10]). Convert the code into $\mathcal{L}$(VM) code.

**Problem 5.2** (Simulating REMA in SML)

40pt

Given the following declarations:

```
datatype register = acc | in1 | in2;
datatype instr = load of int | loadi of int | loadin1 of int | loadin2 of int |
                 store of int | storein1 of int | storein2 of int |
                 add of int | addi of int | sub of int | subi of int |
                 move of register*register| nop of int | stop of int |
                 jump of int | jumpe of int | jumpne of int |
                 jumpl of int | jumple of int | jumpg of int | jumpge of int;
type program = instr list;
type memory = int list;

(* This is the state of the machine. From left to right the values mean:
   PC register; ACC register; IN1 register; IN2 resigter; Memory cells*)
type state = int*int*int*int*(int list);
```

Write two SML functions:

- execute_instr : instr −> state −> state
- run : program −> memory −> memory

The first function takes an ASM instruction and the current state of the REMA as arguments and returns the new state after the instruction is executed. The second function takes a program and the initial configuration of the memory. It then simulates the program until a STOP 0 instruction is reached and returns the memory at that point. In both functions 'memory' is just a list of integers that represent the current state of the memory of the REMA. Once the initial list is supplied, during simulation its length shoudn't change.

**Note:** For this problem and the next it will be very helpful to use built-in SML functions. Make sure to check the forums for more info.

**Solution:**

```
(* Needed in order not to truncate output. *)
Control.Print.printDepth := 100;
Control.Print.printLength := 100;
Control.Print.stringDepth := 100;
```

```
datatype register = acc | in1 | in2;
datatype instr = load of int | loadi of int | loadin1 of int | loadin2 of int |
                 store of int | storein1 of int | storein2 of int |
                 add of int | addi of int | sub of int | subi of int |
                 move of register*register| nop of int | stop of int |
                 jump of int | jumpe of int | jumpne of int |
                 jumpl of int | jumple of int | jumpg of int | jumpge of int;
type program = instr list;
type memory = int list;

(* This is the state of the machine. From left to right the values mean:
   PC register; ACC register; IN1 register; IN2 resigter; Memory cells*)
type state = int*int*int*int*(int list);

(* returns a list identical to mem, where the element index is replaced with new_val *)
fun modify mem index new_val = List.take(mem,index) @ [new_val] @ List.drop(mem,index+1);

(* Data LOAD and STORE instructions *)
fun execute_instr (load(i)) ((pc_r,acc_r,in1_r,in2_r,mem):state) :state = (pc_r+1,List.nth(mem,i),in1_r,in2_r,mem)
  | execute_instr (loadi(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,i,in1_r,in2_r,mem)
  | execute_instr (loadin1(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,List.nth(mem,i+in1_r),in1_r,in2_r,mem)
  | execute_instr (loadin2(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,List.nth(mem,i+in2_r),in1_r,in2_r,mem)
| execute_instr (store(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,modify mem i acc_r)
| execute_instr (storein1(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,modify mem (i+in1_r) acc_r)
| execute_instr (storein2(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,modify mem (i+in2_r) acc_r)

(* Arithmetic instructions *)
| execute_instr (add(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r+List.nth(mem,i),in1_r,in2_r,mem)
| execute_instr (addi(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r+i,in1_r,in2_r,mem)
| execute_instr (sub(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r−List.nth(mem,i),in1_r,in2_r,mem)
| execute_instr (subi(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r−i,in1_r,in2_r,mem)

  (* The MOVE instruction *)
| execute_instr (move(acc,in1)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,acc_r,in2_r,mem)
| execute_instr (move(acc,in2)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,acc_r,mem)
| execute_instr (move(in1,acc)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,in1_r,in1_r,in2_r,mem)
| execute_instr (move(in1,in2)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in1_r,mem)
| execute_instr (move(in2,acc)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,in2_r,in1_r,in2_r,mem)
| execute_instr (move(in2,in1)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in2_r,in2_r,mem)
(* Just for match completeness. *)
| execute_instr (move(acc,acc)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (move(in1,in1)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (move(in2,in2)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,mem)

  (* The STOP and NOP instructions. *)
| execute_instr (stop(_)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r,acc_r,in1_r,in2_r,mem)
| execute_instr (nop(_)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+1,acc_r,in1_r,in2_r,mem)
```

**Solution:**

```
  (* The JUMP instructions. *)
| execute_instr (jump(i)) (pc_r,acc_r,in1_r,in2_r,mem) = (pc_r+i,acc_r,in1_r,in2_r,mem)
| execute_instr (jumpe(i)) (pc_r,acc_r,in1_r,in2_r,mem) = if acc_r = 0
                                                          then (pc_r+i,acc_r,in1_r,in2_r,mem)
                                                          else (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (jumpne(i)) (pc_r,acc_r,in1_r,in2_r,mem) = if acc_r <> 0
                                                          then (pc_r+i,acc_r,in1_r,in2_r,mem)
                                                          else (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (jumpl(i)) (pc_r,acc_r,in1_r,in2_r,mem) = if acc_r < 0
                                                          then (pc_r+i,acc_r,in1_r,in2_r,mem)
                                                          else (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (jumple(i)) (pc_r,acc_r,in1_r,in2_r,mem) = if acc_r <= 0
                                                          then (pc_r+i,acc_r,in1_r,in2_r,mem)
                                                          else (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (jumpg(i)) (pc_r,acc_r,in1_r,in2_r,mem) = if acc_r > 0
                                                          then (pc_r+i,acc_r,in1_r,in2_r,mem)
                                                          else (pc_r+1,acc_r,in1_r,in2_r,mem)
| execute_instr (jumpge(i)) (pc_r,acc_r,in1_r,in2_r,mem) = if acc_r >= 0
                                                          then (pc_r+i,acc_r,in1_r,in2_r,mem)
                                                          else (pc_r+1,acc_r,in1_r,in2_r,mem);


fun run_helper (p:program) (s:state) =
  let
    val ins = List.nth(p, #1 s);
  in
    if ins = stop 0 then s else run_helper p (execute_instr ins s)
  end;

fun run (nil:program) (mem:memory) :memory = mem
  | run p mem = #5 (run_helper p (0,0,0,0,mem));

(* Test Cases - From slides *)
val p1 = [load 0, store 2, load 1, store 0, load 2, store 1, stop 0] : program;
val mem1 = [4, ~10, 0] : memory;
val res1 = [~10, 4, 4] : memory;

val p2 = [load 1, add 2, add 3, store 4, stop 0] : program ;
val mem2 = [0,4,6,~2,10] : memory;
val res2 = [0,4,6,~2,8] : memory;

val p3 = [load 0, move (acc,in1) , load 1, storein1 0, stop 0] : program;
val mem3 = [5,10,0,0,0,0] : memory;
val res3 = [5,10,0,0,0,10] : memory;

val p4 = [load 1, move (acc,in1), load 2, move (acc,in2), load 0, jumpe 13,
          loadin1 0, storein2 0, move (in1,acc), addi 1, move (acc,in1), move (in2,acc),
          addi 1, move (acc,in2), load 0, subi 1, store 0, jump ~12, stop 0] : program;
val mem4 = [5,3,10,~1,~2,~3,~4,~5,~6,0,0,0,0,0,0] : memory;
val res4 = [0,3,10,~1,~2,~3,~4,~5,~6,0,~1,~2,~3,~4,~5] : memory;

val test1 = res1 = run p1 mem1;
val test2 = res2 = run p2 mem2;
val test3 = res3 = run p3 mem3;
val test4 = res4 = run p4 mem4;
```

---

**Problem 5.3 (SW TCN Converter)**

Write a Simple While program that converts TCN to decimal. You will be given the TCN number $t$ (consider it a variable set at the beginning of your program). You can consider that the number $t$ is an integer formed only from 1s and 0s and the most significant bit is represented by the first digit; remember that the first bit also tells you whether the number is negative or positive. 25pt

Your program should start, for example, with:

```
var t=11001;
```

That is, $t = 11001$, which corresponds to $-7$.

---

**Solution:** The student is expected to loop while $t$ is larger than 1 and use the current last digit for binary to decimal conversion. Then, based on the test above, to add $-2^{nrbits}$ to the number or not.

---

**Problem 5.4 (Simple While program on Fibonacci)**

15pt

Write a Simple While Program that takes a number $N$ and computes the $N^{\text{th}}$ Fibonacci number. Then provide the Abstract Syntax for your code.

Show how the $\mathcal{L}$(VM) version of it looks like by compiling it.

---

**Solution:**

```
var n := N; var a := 0;
var b := 1; var c=b; ([ ("n", Con N), ("a", Con 0), ("b", Con 1), ("a", Con 1)],
while 2<= n do While(Leq(Con 2, Var"n"),
c=b+a; Seq [Assign("c", Add(Var"b", Var"a")),
        a=b; Assign("a", "b"),
        b=c; Assign("b", "c"),
n:=n−1; Assign("n", Sub(Var"n", Con 1))]
end ),
return b; Var"c")
```

VM code:

```
con N con 0 con 1; con 1
peek 0 con 2 leq cjp 22
peek 1 peek 2 add poke 3
peek 2 poke 1
peek 3 poke 2
con 1 peek 0 sub poke 0
jp −27
peek 3 halt
```

# Assignment 6: L(VMP) and $\mu$ML (Given Mar. 12., Due Mar. 19.)

**Problem 6.1 (Simple While with procedures)**
Let SWP (Simple While with procedures) be an imperative language that supports function calls. Each of the functions/procedures is defined like a standard `SW` program: a list of variable declarations, a statement or sequence of statements and a return value. The only difference is that you now also have a list of arguments. Here is a program, written in concrete syntax that has one procedure which computes the factorial function and the main program uses it to calculate $10! + 1$. 40pt

```
fun fact(n)
    var i:=1;
    var f:=1;
    while i<=n do
        f:=f*i;
        i:=i+1;
    end
    return f;

main
    var r:=0;
    r:=fact(10)+1;
    return r;
```

Let the abstract syntax of this language be defined with the following types/datatypes:

```
type id = string (* identifier *)

datatype exp = (* expression *)
  Con of int (* constant *)
| Id of id (* variable or argument *)
| Add of exp * exp (* addition *)
| Sub of exp * exp (* subtraction *)
| Mul of exp * exp (* multiplication *)
| Leq of exp * exp (* less or equal test *)
| App of id * exp list (* application *)

datatype sta = (* statement *)
  Assign of id * exp (* assignment *)
| If of exp * sta * sta (* conditional *)
| While of exp * sta (* while loop *)
| Seq of sta list (* sequentialization *)

type declaration = id * exp (* variable declaration *)

type procedure = id * id list * declaration list * sta * exp (* procedure declaration *)

type programBody = declaration list * sta * exp (* main procedure *)

type program = procedure list * programBody
```

The same example written in abstract syntax:

```
val factprog = ([("fact",["n"],[ ("i", Con(1)), ("f", Con(1)) ],
                              While(Leq(Id("i"), Id("n")),
                                       Seq([Assign("f", Mul(Id("f"), Id("i"))),
                                             Assign("i", Add(Id("i"), Con(1))) ])),
                              Id("f")) ],
 ( [("r", Con(0))],
       Assign("r", Add( App("fact", [Con (10)]), Con(1) )),
 Id("r") ) )
```

This language can be compiled using a combination of $\mathcal{L}(\mathtt{VM})$ and $\mathcal{L}(\mathtt{VMP})$. The problem with `peek` and `poke` inside procedures can be solved in the same way `SW` is compiled - first make the list of initially declared values and then use `peek` for variable reference, `poke` for variable assignment and `arg` for argument reference. The arguments of `peek` and `poke` will now be referencing indexes in the stack with respect to the current frame, i.e. form 1 to $n$, where $n$ is the number of variable declarations. The only exception is with the main program where the is no return address stored in the current frame, so the numbering will start from 0. `return` will delete the variables along with the arguments and return address.

Your task is to write an SML function compileSWP which takes a SWP program in abstract syntax as an argument and compiles it into $\mathcal{L}(\mathtt{VM})$ + $\mathcal{L}(\mathtt{VMP})$ (with the abstract syntax defined on the slides).
Signature and example:

```
val compileSWP = fn : program −> code

− compileSWP(factprog);
val it =
  [proc (1,33),con 1,con 1,arg 1,peek 1,leq,cjp 18,peek 1,peek 2,mul,poke 2,
   con 1,peek 1,add,poke 1,jp ~21,peek 2,return,con 0,con 1,con 10,call 0,add,
   poke 0,peek 0,halt] : code
```

**Solution:**

**Problem 6.2 (μML Mystery)**

You are given the following piece of μML code: 30pt

```
(
[
("helper1", ["x", "y"],
 If(Leq(Id"y", Con 0), Con 0,
  If(Leq(Id"x", Id"y"), App("helper1", [Id"x", Sub(Id"y", Id"x")]), Con 1))),

("helper2", ["x", "n"],
 If(Leq(Id"n", Con 2), App("helper1", [Con 2, Id"x"]),
  If(App("helper1", [Id"n", Id"x"]),
     If(App("helper2", [Id"x", Sub(Id"n", Con 1)]), Con 1, Con 0),
     Con 0)
  )
),
("myproc", ["x"],
  App("helper2", [Id"x", Sub(Id"x", Con 1)])
)
],

App("myproc", [Con 6])
);
```

1. State what the program *myproc*($x$) is doing. (Assume $x > 1$.)
2. Write another $\mu$ML program that does the same thing in a different way. Remember that a $\mu$ML program should be a pair of a well defined list of function declarations, and a single App call to the main function. Of course, that function will be calling the helping function(s) in its body, and the helping functions may call themselves.

---

**Solution:**

1. The program checks whether the number $x$ is prime. It starts by checking whether $x$ is divisible by $x - 1$ with a call to $helper2$, and then recursively continues checking the divisibikity by $x-2$, $x-3$, etc., until it finds a number by which $x$ is divisible, and returns 0, or if it reaches $helper2(x, 1)$, stops and returns 1.
2. Here is a long, standard solution

```
(
[

("Equal", ["a","b"],
 If(Leq(Id"a",Id"b"),
     If(Leq(Id"b",Id"a"), Con 1, Con 0),
     Con 0
    )
),

("Less", ["a","b"],
 If(Leq(Id"a",Id"b"),
     If(Leq(Id"b",Id"a"), Con 0, Con 1),
     Con 0
    )
),

("Mod", ["a","b"],
 If(App("Less",[Id"a",Id"b"]),
     Id"a",
     App("Mod", [Sub(Id"a", Id"b"), Id"b"])
    )
),

("Div", ["a","b"],
 If(App("Less",[Id"a",Id"b"]),
     Con 0,
     Add(Con 1, App("Div", [Sub(Id"a",Id"b"), Id"b"]))
    )
),

("FindRoot", ["n","i"],
 If(Leq(Mul(Id"i", Id"i"), Id"n"),
     App("FindRoot", [Id "n", Add(Id "i",Con 1)]),
     Sub(Id "i",Con 1)
    )
),

("FindDv", ["n","i","b"],
 If(Leq(Id"i", Id"b"),
     If(App("Equal", [App("Mod", [Id"n",Id"i"]),Con 0]),
         Con 0,
         App("FindDv", [Id"n", Add(Id"i", Con 2), Id"b"])
         ),
     Con 1
    )
),

("IsPrime", ["n"],
   If(App("Equal", [Id"n",Con 2]),
       Con 1,
       If(App("Equal", [App("Mod", [Id"n",Con 2]),Con 0]),
           Con 0,
           App("FindDv", [Id"n", Con 3, App("FindRoot", [Id"n", Con 1]) ])
           )
     )
)

],

App("IsPrime", [Con 6]) );
```

---

**Problem 6.3 (SML simulator for $\mathcal{L}$(VM) with static procedures)**
Write an SML function vm_simulator that simulates a program written in Virtual Machine Language. It should take as input: 40pt

- a string list, a list of $\mathcal{L}$(VM) commands including `con` $i$, `add`, `sub`, `mul`, `leq`, `jp` $i$, `cjp` $i$, `peek` $i$, `poke` $i$, `proc` $i$ $j$, `arg` $i$, `call` $i$, `return`, `halt` and numbers. Note that, for example, "`con` " and "5" count as 2 instructions.
- an int list describing the initial state of the stack; the first element in the list represents the top of the stack while the last element represents bottom of the stack. Of course it can also be initially empty.

```
val vm_simulator = fn : string list * int list −> int list
```

As there are a few things that could possibly go wrong, make sure to use the following exceptions:

- NumberExpected, if no number follows after `con` , `jp` , `cjp` , `peek` , `poke` .
- InvalidPeekPoke, if the number following `peek` or `poke` is smaller than 0 or bigger than the size of the stack.
- InvalidJump, if the number following `jp` , `cjp` , `call` or `proc` makes the virtual program counter point to an unexisting instruction.
- InvalidStackSize, if the stack is not big enough so that the current instruction can be normally executed. (`add`, `sub`, `leq` etc.)
- UnknownCommand, if the input string list contains strings that are not one of the VML commands described in class.

Signature and examples:

```
val vm_simulator = fn : string list * int list −> int list

vm_simulator(["con","~2","con","~7","mul","con","3","con","4","mul","sub"],[2,9,7]);
val it = [~2,2,9,7] : int list


vm_simulator(["proc","2","9","arg","1","arg","2","add","return",
              "con","1","con","2","con","3","con","4",
              "call","0","call","0","call","0","halt"], []);
val it = [10] : int list
```

---

**Note:** You can safely assume that in each test case only 1 version of $\mathcal{L}$(`VM`) will be used, i.e. either $\mathcal{L}$(`VM`) with `peek` $i$ and `poke` $i$ or $\mathcal{L}$(`VM`) with `proc` $i$ $j$ and `call` $i$.

---

**Solution:**

```
exception NumberExpected;
exception InvalidPeekPoke;
exception InvalidJump;
exception UnknownCommand;
exception InvalidStackState;

fun int_option_to_int(SOME (x)) = x |
int_option_to_int(NONE) = raise NumberExpected;

fun peek(i, nil) = raise InvalidPeekPoke |
    peek(i, x::stack) = if length(x::stack) <= i then raise InvalidPeekPoke else
    if length(stack) = i then x else peek(i, stack);

fun poke(x, i, nil) = raise InvalidPeekPoke |
    poke(x, i, y::stack) = if length(x::stack) <= i then raise InvalidPeekPoke else
    if length(stack) = i then x::stack else y::poke(x, i, stack);

fun cut_first_i(i, nil) = if not (i = 0) then raise InvalidJump else nil |
    cut_first_i(0, stack) = stack |
    cut_first_i(i, x::stack) = cut_first_i(i−1, stack);

fun ith(0, x::ls) = x |
    ith(i, x::ls) = ith(i−1, ls);

fun get_args(0, ls) = nil |
    get_args(i, x::ls) = x::get_args(i−1, ls);


fun step(args, all, vpc, "add"::prog, x::y::stack) = step(args, all, vpc+1, prog, (x+y)::stack) |
    step(args, all, vpc, "add"::prog, x::nil) = raise InvalidStackState |
    step(args, all, vpc, "add"::prog, nil) = raise InvalidStackState |
    step(args, all, vpc, "sub"::prog, x::y::stack) = step(args, all, vpc+1, prog, (x−y)::stack) |
    step(args, all, vpc, "sub"::prog, x::nil) = raise InvalidStackState |
    step(args, all, vpc, "sub"::prog, nil) = raise InvalidStackState |
    step(args, all, vpc, "mul"::prog, x::y::stack) = step(args, all, vpc+1, prog, (x*y)::stack) |
    step(args, all, vpc, "mul"::prog, x::nil) = raise InvalidStackState |
    step(args, all, vpc, "mul"::prog, nil) = raise InvalidStackState |
    step(args, all, vpc, "leq"::prog, x::y::stack) = if x <= y then step(args, all, vpc+1, prog, 1::stack)
```

```
            else step(args, all, vpc+1, prog, 0::stack) |
      step(args, all, vpc, "leq"::prog, x::nil) = raise InvalidStackState |
      step(args, all, vpc, "leq"::prog, nil) = raise InvalidStackState |
      step(args, all, vpc, "halt"::prog, stack) = stack |
      step(args, all, vpc, "con"::i::prog, stack) = step(args, all, vpc+2, prog,
    int_option_to_int(Int.fromString(i))::stack) |
      step(args, all, vpc, "con"::nil, stack) = raise NumberExpected |
      step(args, all, vpc, "peek"::i::prog, stack) = step(args, all, vpc+2, prog,
    peek(int_option_to_int(Int.fromString(i)), stack)::stack) |
      step(args, all, vpc, "peek"::nil, stack) = raise NumberExpected |
      step(args, all, vpc, "poke"::i::prog, x::stack) = step(args, all, vpc+2, prog,
    poke(x, int_option_to_int(Int.fromString(i)), stack)) |
      step(args, all, vpc, "poke"::nil, stack) = raise NumberExpected |
      step(args, all, vpc, "jp"::i::prog, stack) = step(args, all,
       vpc + int_option_to_int(Int.fromString(i)),
                                                          cut_first_i(vpc + int_option_to_int(Int.fromString(i)), all),
                                                          stack) |
      step(args, all, vpc, "jp"::nil, stack) = raise NumberExpected |
      step(args, all, vpc, "cjp"::i::prog, x::stack) = if x = 0 then step(args, all,
         vpc + int_option_to_int(Int.fromString(i)),
         cut_first_i(vpc + int_option_to_int(Int.fromString(i)), all),
                        stack)
                                                              else step(args, all, vpc+2, prog, stack) |
      step(args, all, vpc, "cjp"::nil, stack) = raise NumberExpected |
      step(args, all, vpc, nil, stack) = stack |

      (*proc*)
      step(args, all, vpc, "proc"::arg::len::prog, stack) = step(args, all, vpc + int_option_to_int(Int.fromString(len)),
                                                            cut_first_i(vpc + int_option_to_int(Int.fromString(len)), all),
                                                     stack) |
      step(args, all, vpc, "proc"::arg::nil, stack) = raise NumberExpected |
      step(args, all, vpc, "proc"::nil, stack) = raise NumberExpected |

      (*call*)
      step(args, all, vpc, "call"::add::prog, stack) =
      let
         val addr = int_option_to_int(Int.fromString(add))
         val no_arg = int_option_to_int(Int.fromString(ith(addr+1, all)))
      in
         step(args, all, vpc + 2, prog,
     step(get_args(no_arg,stack),
all,
addr + 3,
           cut_first_i(addr + 3, all), nil)
              @cut_first_i(no_arg,stack))
      end |
      step(args, all, vpc, "call"::nil, stack) = raise NumberExpected |

      (*return*)
      step(args, all, vpc, "return"::prog, x::stack) = [x] |

      (*arg*)
      step(args, all, vpc, "arg"::no::prog, stack) =
      let
         val argument = ith(int_option_to_int(Int.fromString(no))−1, args)
            step(args, all, vpc+2, prog, argument::stack)
      end |
      step(args, all, vpc, "arg"::nil, stack) = raise NumberExpected |

      step(args, all, vpc, x, stack) = raise UnknownCommand;


fun vm_simulator(program, initial_stack) = step(nil, program, 0, program, initial_stack);




(* test cases *)


val test0 = vm_simulator(["proc","2","9","arg","1","arg","2","add","return",
   "con","1","con","2","con","3","con","4",
                                "call","0","call","0","call","0","halt"], []) = [10];
val test1 = vm_simulator(["con","12","con","1","peek","0","con","2","leq","cjp","18","peek","0",
"peek","1","mul","poke","1","con","1","peek","0","sub","poke","0","jp","~21","peek","1","halt"],[]) =
[479001600,479001600,1];
val test2 = vm_simulator(["proc", "1", "6", "con", "3", "return", "con", "4", "call", "0", "halt"],[]) = [3];
val test3 = vm_simulator(["proc", "1", "6", "arg", "1", "return", "con", "4", "call", "0", "halt"],[]) = [4];
val test4 = vm_simulator(["proc", "1", "9", "con", "3", "arg","1","mul","return",
   "con", "4", "call", "0", "halt"],[]) = [12];
val test5 = vm_simulator(["proc","2","26",
   "con", "0", "arg","2","leq","cjp","5",
                                "con", "1", "return",
                                "con", "1", "arg", "2", "sub", "arg", "1",
                                "call", "0", "arg", "1", "mul",
                                "return",
                                "con", "5", "con", "10", "call", "0", "halt"], []) = [100000];
val test6 = vm_simulator(["proc", "1" ,"23",
    "arg", "1",
   ”con”, ”1”, ”arg”, ”1”, ”leq”, ”cjp”, ”5”,
                              ”arg”, ”1”, ”return”,
                              ”con”, ”2”, ”arg”, ”1”, ”sub”, ”call”, ”0”,
                              ”return”,
                              ”proc”, ”1”, ”24”,
                              ”arg”, ”1”, ”con”, ”2”, ”leq”, ”cjp”, ”13”,
                              ”con”, ”2”, ”arg”, ”1”, ”sub”, ”call”, ”23” , ”con”, ”1”, ”add”, ”return”,
                              ”con”, ”0”, ”return”,
                              ”con”, ”13”, ”call”, ”0”,
                              ”con”, ”13”, ”call”, ”23”, ”halt”], []) = [6, 1];
val test7 = vm_simulator([”proc”, ”1” ,”23”,
   ”arg”, ”1”,
   ”con”, ”1”, ”arg”, ”1”, ”leq”, ”cjp”, ”5”,
                              ”arg”, ”1”, ”return”,
                              ”con”, ”2”, ”arg”, ”1”, ”sub”, ”call”, ”0”,
                              ”return”,
                              ”proc”, ”1”, ”24”,
                              ”arg”, ”1”, ”con”, ”2”, ”leq”, ”cjp”, ”13”,
                              ”con”, ”2”, ”arg”, ”1”, ”sub”, ”call”, ”23” , ”con”, ”1”, ”add”, ”return”,
                              ”con”, ”0”, ”return”,
                              ”proc”, ”1”, ”28”,
                              ”con”,”1”,”arg”,”1”, ”leq”, ”cjp”, ”5”,
                              ”arg”, ”1”,”return”,
                              ”arg”,”1”,”call”,”23”,”call”,”47”,
                              ”con”, ”10”, ”mul”,
                              ”arg”,”1”,”call”,”0”,”add”,
                              ”return”,
                              ”con”, ”3”, ”call”,”47”,
                              ”con”, ”13”, ”call”, ”47”,
                              ”con”, ”123”, ”call”, ”47”,
                              ”halt”], []) = [1111011,1101,11];
```

# Assignment 7: Turing Machines (Given Mar. 19., Due Mar. 26.)

**Problem 7.1 (Divisible by three in Turing Machine)**
Design a Turing Machine that determines whether a number is divisible by 3. Your TM is over alphabet $0, 1, \#, Y, N$, and you can assume that your head is initially positioned on the first digit of a binary representation of a non-negative number which is written between two hash symbols (#). Upon termination, your program should replace the second hash with $Y$ or $N$ indicating that the number whose binary representation is written on tape is divisible by three or not. So if the tape was 30pt

$$\#\overbrace{1}^{Hd}010\#$$

then after execution we should have:

$$\#1010\overbrace{Y}^{Hd}$$

It is mandatory that you explain your approach in plain English.

**Solution:**

First add a third # at the end. This way you divide the tape into 2 areas: the input on the left (finite) and the working tape on the right (right infinite).

Compute the number in unary form (e.g. $4 = 1111$) and write it after the second hash. You can do that by the following algorithm:

1. Double the number of ones on the working tape
2. Go do the first bit of the input. If it is 1, add a 1 to the working tape. Replace the bit by a #.
3. Go to step 1.

Then, using three additional states, check if the length of the string of ones you created is a multiple of 3.

OR:

Use the fact that a binary number is divisible by three if the number of even bits minus the number of odd bits is a multiple of 3.

Start converting ones to zeros in the following way: for every even positioned one you make zero, find an odd position one and do the same. Check if the number of ones left is divisible by 3.

**Problem 7.2 (Palindrome Detector Turing Machine)**
Create a Turing machine that detects whether the word $w \in \{0, 1\}^*$ on the tape is a palindrome, i.e. $w = w^R$, where $w^R$ is $w$ reversed. If so, leave a 1 at the final position of the head, which can be anywhere on the tape; otherwise leave a 0. Assume that, initially, 25pt

the head is over the first character of $w$, and that $w$ is surrounded by hash marks as delimiters, i.e. you have the alphabet $\{0, 1, \#\}$.

**Note:** Admissible moves are $L$ (*left*), $R$ (*right*), and $N$ (*none*) with the obvious meaning.

**Solution:** Idea: Define states that encode one memorized character, as well as the information whether the beginning or the end of the word is currently being examined. In each step, we compare the first character of the word to the last one, erase them and proceed.

1. Read a character at one end of the word.
   - If it is a digit, memorize whether it was 0 or 1 by using an appropriate state.
   - If it is a hash mark, write a 1 and halt.
2. Overwrite the character with a hash mark.
3. Walk to the other end of the word, i.e. move until a hash mark is read and then move one cell back to the beginning/end of the word (using the "motion" information encoded in the current state).
4. - If the character under the head does not match the memorized one, write a 0 and halt.
   - If it does match, overwrite it with a hash mark, move one cell further to the new beginning/end of the word and continue from the beginning.

**Problem 7.3 (Various Bracket structure)**

Design the following Turing Machine. Its input is a word $w \in \{(,),[,]\}^*$, surrounded by hash marks as delimiters. You can assume that your head is initially positioned right after the first hash. Upon termination, your program should replace the second hash with Y or N indicating if the bracket structure is correct (Y - for correct, N otherwise). Thus the overall alphabet is $\{(,),[,],\#,Y,N\}$. 30pt

**Note:** Correct bracket structure is a sequence of the same amount of opening and closing brackets, where the amount of opening brackets is not less than the amount of closing brackets in any prefix of such a sequence and they are positioned in the proper order.

Example:

| | |
|---|---|
| initial | #[[(]])# |
| result | $\# - something - N$ |
| initial | #[()[]]# |
| result | $\# - something - Y$ |

**Solution:** Go through the input until you find the first close bracket and replace it by $Y$. Go backward and try to find a matching open bracket. If there is one, replace it with $Y$ and continue the algorithm otherwise stop and answer $N$. If at the end there are only $Y$'s then answer $Y$, but if anywhere along the way we have discovered a bracket mismatch the answer would be $N$.

```
1,( 1,(,>
1,) 2,Y,<
1,Y 1,Y,>
1,# 4,#,<

2,) 2,),<
2,( 1,Y,>
2,Y 2,Y,<
2,# 3,#,>

3,) 3,),>
3,Y 3,Y,>
3,( 3,(,>
3,# H,N,>

4,( 3,(,>
4,) 3,),>
4,Y 4,Y,<
4,# 5,#,>

5,Y 5,Y,>
5,# H,Y,>
```

**Problem 7.4 (Counting Characters)**

Given a string of a's and b's delimited by # (e. g. #$\underline{a}baaabbabba$#) with the head (underlined) initially over the leftmost letter, write a Turing machine program that finds out whether there are more a's than b's or vice versa. Return the result as one character under the head when the machine stops: An "a" for "more a's", a "b" for "more b's", or a "#" for an equal number. 15pt

**Solution:** Idea: in every step, delete one a and one b – roaming the whole tape within the delimiters –, overwriting a's with 0's and b's with 1's, until that's not possible any more.

# Assignment 8: Internet and Mail Services (Given Apr. 13., Due Apr. 19.)

**Problem 8.1 (SMTP Mail Writer)**

You have recently learned in the lecture about how you can connect to a SMTP server via Telnet and send a simple email message. Your first task is now to automate the pocess by creating an SML function sendMail that will: 35pt

1. open a connection (socket) to the server
2. issue a 'HELO' command for self-identification
3. start a mail message using 'MAIL FROM'
4. set the recipient ('RCPT') and mail contents ('DATA')
5. close the connection using 'QUIT'

Remember that the server will understand a single point on a line as a mark of mail ending!

Your function should have the following signature:

```
val sendMail = fn : string * int * string list * string -> bool
```

The parameters, in order, are: the hostname, the port number (usually 25), a list of mail recipients, and the message body. The function should return true if everything was OK or false if there were problems. You are not required to treat any exceptions raised by SML library functions in your implementation.

**Solution:**

**Problem 8.2 (POP Mail Reader)**

Write the following SML functions: 35pt

1. countMails, which returns the number of emails available on the server. Consider that you have to first login using your username and password, so you will first to send the username and password. The function must have the following signature:

   ```
   val countMails = fn : string * int * string * string -> int
   ```

   The arguments are, in order: the string identifying the host, the port number (usually 110), the user name and the password. The return value is the number of emails available on the server.

2. readMail, which returns the contents of one of the emails available on the server. The function must have the following signature:

   ```
   val readMail = fn : string * int * string * string * int -> string
   ```

The arguments are, in order: the string identifying the host, the port number, the user name and password and the number of the email that you are going to request. The return value is the content of the email you requested.

The email contains a set of meta-data containing, for example, the sender, the subject, and date. You can simply leave them in the output string for now.

---

**Solution:** The protocol for POP asks for the following order of events:

```
> USER [username]
> PASS [password]
> STAT
< +OK [nr of mails] [total size (?)]
> RETR [number of mail]
< [email contents ending in one single "." per line]
```

Therefore, in SML, our strategy would be:

1. establish working connection
2. send "user X" as string
3. read and check for "+OK" at beginning
4. send "pass Y" as string
5. read and check again for "+OK"
6. and so on...

---

**Problem 8.3 (Email relay server)** 35pt

Now that you have implemented mail sending and retrieving, consider writing an email relay server. Theoretically, it should be contacted by an email client in order to implement routing of the messages between the sending client to the destination server.

In your case, you have to write an SML function routeMails that will accept incoming connections and requests of email sending. Your function will emulate an SMTP server for a limited set of commands (the ones that you implemented in the SMTP client are enough). After getting the contents of the email, it will simply re-transmit it further to the destination email address (that you have to read from the message you receive).

The function will have the following signature:

```
val routeMails = fn : int * string -> bool;
```

The int represents the port number the server will open to, while the string represents the address of the server the email is going to be forwarded to. The function return value should be true in case everything went OK, and false in case of error. Alternatively, you can leave thrown exceptions uncaught to signal an error. Here is an example run:

```
(* you run this function first *)
routeMails(2525, "exchange.jacobs-university.de");

(* then, in another terminal, you run this *)
sendMail("localhost", 2525, ["youraddress@jacobs-university.de"],
"Test");
```

You should wait for a short period of time and then check your Jacobs mail. You should see the email there!

**Note:** The server should wait in a loop for the next message to come - that means you should not finish execution after the first email has been forwarded.

**Solution:**

# Assignment 9: WWW
# (Given Apr. 9., Due Apr. 16.)

### Problem 9.1 (Final practice)

40pt

The final exam is not more than a month away so it is good to start practicing. To do this you decide to make quizzes and exchange them with your classmates. Please have a first page that contains a form where the student must fill in their name, age and e-mail address. You must validate that the data here makes sense with PHP (i.e. they are strings/integers and the e-mail address is a proper e-mail address). Then students should be redirected to a short 5 question quiz to practice for the final exam and in the end their results shall be reported back to them. Make sure to display the correct answers to the questions they got wrong.

The web form must:

1. Include multiple choice and 'fill in the blanks' questions.
2. Include all of the following: button, radio button, check box, drop down box, text input.
3. The overall style should be professional. Put a bit of effort into appearance and aesthetics.
4. In the end, the scoring system should work. Nothing too fancy, but it should be an operational exam from start to finish.

# Assignment 10: Encryption and Search (Given Apr. 30., Due May. 7.)

**Problem 10.1 (SML Web Crawler)** 30pt

A web crawler is a program that will store a copy (mirror) of a web site. Generally, crawlers access a given web page and, after retrieving the HTML source, they extract the links and also download those pages (or images or scripts). This will provide the user the possibility to access these pages even when they are not connected to the internet or to perform different measurements on the pages.

Your task is to write your own SML Web Crawler, following these steps:

1. Make sure that you downloaded and understood the SML sockets example file used in the last assignment. Use the following updated socketReceive function:

   ```
   (* Receives maxbytes bytes from the socket. Returns the string message. *)
   fun socketReceive(sock, maxbytes) =
           Byte.bytesToString(
                   Socket.recvVecNB(sock, maxbytes)
           );
   ```

   The problem with this function is that, if the server sends a message longer than maxbytes, all the remaining bytes will be queued on the socket, but not processed. Write your own fullMessage function that overcomes this problem by reading the whole reply from the server (you can use socketReceive, it will return a string of length 0 if the message from the server is finished). Your function should have the following type:

   ```
   val fullMessage= fn : ('a,Socket.active Socket.stream) Socket.sock −> string
   ```

2. Now, write a method that, given a *host* and *page*, will make a HTTP **GET** request to the server for the given *page* on that *host*, and will return the HTTP response. Your function should have the following signature:

   ```
   val getPage = fn : string * string −> string
   ```

   For example, you should be able to run getPage("en.wikipedia.org","/wiki/Main_Page") and retrieve the home page of Wikipedia.

3. Now that you have the HTTP response, check it closely and you will discover that it contains the HTML web page, but also some headers. In order to be sure that you will only store the HTML page, write a function extractHTML that scans the string and discards everything that **is not** between <html> and </html>. Of course, your function will have the signature:

   ```
   val extractHTML : string −> string
   − extractHTML("Discard me! <html><head><title>Hello!</title></head></html>");
   val it = "<html><head><title>Hello!</title></head></html>" : string;
   ```

4. Write a function extractLinks that will go through your HTML source code and will return all the links that it contains. Feel free to look into the HTML or RegExp library of SML, but making your function only going through the string and extracting sequences like the following will suffice:

   ```
   <a href="extract me!">...
   <img src="extract me!"> ...
   ```

   You are not required to handle links other than the ones found in anchors and images. Your function will have the following signature (get a string and return a list of strings which are the links found):

   ```
   val extractLinks = fn : string -> string list;
   ```

5. Mind the fact that these links might contain the protocol ("http://"), might be relative to the root of the host ("/img/happy.png"), or might be relative to the current page ("next/index.html"). Your getPage function requires a *host* and a *page* as arguments, and the *page* should be relative to the host root (i.e. absolute path). Write an SML function normalizeLinks that, given a host, page and list of strings, will return a list of pairs (*host*, *page*) that can be used by the getPage:

   ```
   val normalizeLinks : string * string * string list -> (string * string) list;
   normalizeLinks("www.example.com", "/en/test.html",
           ["http://www.google.com/something/x", "/img/happy.png", "next/index.html"]
   );
   val it = [
           ("www.google.com", "/something/x"),
           ("www.example.com", "/img/happy.png"),
           ("www.example.com", "/en/next/index.html")
   ] : (string * string) list;
   ```

6. This sub-task will be to write the wrapping crawler function.

   Have a look at the following SML function that writes a string to a file:

   ```
   fun writeToFile(file, content) =
           let
                   val os = TextIO.openOut(file)
                   val vc = String.toString(content) (* we need an SML vector *)
                   val _ = TextIO.output(os, vc)
                   val _ = TextIO.flushOut(os)
           in
                   TextIO.closeOut(os)
           end;
   ```

   This function will be used in storing the HTML page to disk. Your crawler will have the following signature:

   ```
   val crawler : string * string * int -> unit;
   ```

The first two parameters are the host and the starting page (i.e. "`www.example.com`" and "`/test/index.html`"). The third parameter is an integer representing the maximum depth you should go into. You will follow the following steps:

(a) use getPage to retrieve the HTTP response

(b) use extractHTML to extract only the HTML part of the response

(c) write the HTML part to a file (see the note below!)

(d) use extractLinks and normalizeLinks to get the list of links to follow further

(e) recursively call the crawler method; remember to decrease the depth and not proceed with a negative depth!

---

**Note:** There might be problems with storing images. We will not grade this problem based on the output, but rather on how well you managed to follow the instructions and on your intermediary results. Please think about what the problem with images is and write a short comment at the end of your sml file!

---

**Solution:**

## Problem 10.2 (Parsing XML)

We are going to consider XML files with a special, more restricted format: 25pt

- Each line contains a Tag or a Content String.
- A Tag line is either of form "<NAME>" (for opening tags) or "</NAME>" (for closing tags). The lines between the opening and the closing tags can contain a Content String (in which case the Tag is called "leaf") or several other tags (and then the Tag is named "node").
- The Content String is a valid SML string.
- The file always starts with a Tag and ends with the corresponding closing Tag (this tag is called the "root" of the document).
- A Tag "B" that is contained in a node "A" is called "son" of "A". Conversely, "A" is the "father" of "B". "B" can have multiple "brothers" (which are Tags that also have "A" as a father).

An example of a valid XML file:

```
<first>
 <a2>
  <b3>
   text
  </b3>
  <b1>
   text3
  </b1>
 </a2>
</first>
```

The root is “first”, we have node “a2”, with leaves “b3” and “b1”.

We can “normalize” an XML file by alphabetically sorting the children of every node after the tag name. In case there are several children with the same name, we want them to keep the relative order they had in the input file (in other words, the sort is stable).

You are required to write an SML function sortTags that takes two arguments (the input file name and the output file name), reads the input file and then writes the normalized XML file to the output.

Example:

| File “input.xml”: | SML: | File “output.xml”: |
|---|---|---|
| `<r>`<br>` <a3>`<br>`  <b2>`<br>`   Father`<br>`  </b2>`<br>`  <b3>`<br>`   Mother`<br>`  </b3>`<br>` </a3>`<br>` <a1>`<br>`  <ffx>`<br>`   AAA`<br>`  </ffx>`<br>`  <ff>`<br>`   BBB`<br>`  </ff>`<br>` </a1>`<br>`</r>` | sortTags(“input.xml”, “output.xml”); | `<r>`<br>` <a1>`<br>`  <ff>`<br>`   BBB`<br>`  </ff>`<br>`  <ffx>`<br>`   AAA`<br>`  </ffx>`<br>` </a1>`<br>` <a3>`<br>`  <b2>`<br>`   Father`<br>`  </b2>`<br>`  <b3>`<br>`   Mother`<br>`  </b3>`<br>` </a3>`<br>`</r>` |

**Note:** The whitespace at the beginning of each line is ignored, but it should be preserved to the output file. Consider the input file to be well formatted (no closing tag / opening tag without match inside their parent tag).

**Solution:**

```
 (*
 * read the contents of a file
 * returns list of strings - each line of the file
 *)
fun loadFile(name) =
let
val fin = TextIO.openIn(name)
fun read() =
let
val x = TextIO.inputLine(fin)
in
if x = NONE then [] else (valOf x) :: read()
end
in
read()
end;

(*
 * write a list of vectors to a file
 *)
fun dumpFile(name, contents) =
let
val fout = TextIO.openOut(name)
fun write([]) = (TextIO.flushOut(fout); true)
  | write(a::l) = (TextIO.output(fout, a); write(l))
in
write(contents)
end;

(*
 * for a string, return it without
 * whitespace at the beginning
 *)
fun dropSpace(s) =
let
val ss = explode(s)
fun f(#" "::l) = f(l)
  | f(#"\t"::l) = f(l)
  | f(a::l) = a::l
in
implode(f(ss))
end;

(* datatype for xml file *)
datatype xml = leaf of string | tag of string * (xml list);

(*
 * dumb test to see if a line is a tag:
 * it should contain a "<"
 *)
fun isTag(s) =
if List.exists (fn x=>(x= #"<")) (explode s)=false then
0
else
if List.exists (fn x=>(x= #"/")) (explode s) then
2
else
1;

(*
 * creates a list of (xml, level)
 * where the level will correspond to the level
 * of the xml element from the file
 *)
fun toXML([], level) = []
  | toXML(a::l, level) =
let
val s = dropSpace(a)
val ist = isTag(s)
in
```

```
if ist>0 then
if ist=1 then
(tag(a, []), level)::toXML(l, level+1)
else
toXML(l, level)
else
(leaf(a), level)::toXML(l, level−1)
end;

(*
 * splits the list after a criterion f
 *)
fun takeWhile([], f) = ([], [])
  | takeWhile(a::l, f) =
if f(a) then
let
val (x, y) = takeWhile(l, f)
in
(a::x, y)
end
else
([], a::l);

(*
 * reconstructs the xml in the datatype
 *)
fun parseXML((tag(t,_),lvl)::l) =
let
val (x,rst) = takeWhile (l, (fn (cx,cy)=>(cy>lvl)))
val d = parseXML(x)
in
tag(t,d)::parseXML(rst)
end
  | parseXML((leaf x, lvl)::l) =
(leaf x) :: parseXML(l)
  | parseXML([]) = [];

(*
 * sort for the tags
 *)
fun mergesort(x) =
if List.length x = 1 then
x
else
let
fun merge([], []) = []
  | merge(la, []) = la
  | merge([], lb) = lb
  | merge(tag(a,ca)::la, tag(b,cb)::lb) =
if a<b then
tag(a,ca)::merge(la,tag(b,cb)::lb)
else
```

```
tag(b,cb)::merge(tag(a,ca)::la,lb)
   | merge(leaf(a)::la, leaf(b)::lb) =
leaf(a) :: merge(la, leaf(b)::lb)
   | merge(tag(a,ca)::la, leaf(b)::lb) =
tag(a,ca) :: merge(la, leaf(b)::lb)
   | merge(leaf(a)::la, tag(b,cb)::lb) =
tag(b,cb) :: merge(leaf(a)::la, cb)
val l = List.length x
val a = mergesort(List.take(x, l div 2))
val b = mergesort(List.drop(x, l div 2))
in
merge(a,b)
end;

(*
 * recursively sort down the tags
 *)
fun sortall(tag(s, list)) =
let
val x = List.map sortall list
val sx = mergesort(x)
in
tag(s, sx)
end
   | sortall(leaf(s)) = leaf(s);


(*
 * given a tag, return the matching ending tag
 *)
fun closeTag(a) =
let
fun recursive(#"<"::lx) = #"<" :: #"/" :: lx
   | recursive(x::lx) = x::recursive(lx)
in
implode(recursive(explode(a)))
end;

fun mymap([]) = []
   | mymap(t1::l1) = XMLtoString(t1) @ mymap(l1)
and XMLtoString(tag(a,lst)):(string list) =
[a] @ mymap(lst) @ [closeTag(a)]
   | XMLtoString(leaf a) = [a];

(*
 * wrapper
 *)
fun sortTags(fin, fout) =
let
val lines = loadFile(fin)
val [process] = parseXML(toXML(lines,0))
val sorted = sortall(process)
```

```
in
dumpFile(fout, XMLtoString(sorted))
end;
```

---

**Problem 10.3 (Transport Layer Secutrity)**

15pt

- What is Transport-Layer Security (TLS)? Specify the steps which are executed by a client and a server in a TLS handshake.
- Explain the difference between public and private key encryption?
- What is an asymmetric key?
- Name 4 current commonly used encryption algorithms and explain one in detail.

---

**Solution:** Transport layer security (TLS) is a cryptographic protocol that encrypts the segments of network connections at the transport layer, using asymmetric cryptography for key exchange, symmetric encryption for privacy, and message authentication codes for message integrity.

1. Client presents a list of supported encryption methods
2. Server picks the strongest and tells client (C/S agree on method)
3. Server sends back its public key certificate (name and public key)
4. Client confirms certificate with CA (authenticates Server if successful)
5. Client picks a random number, encrypts that (with servers public key) and sends it to server.
6. Only server can decrypt it (using its private key)
7. Now they both have a shared secret (the random number)
8. From the random number, both parties generate key material

---

**Problem 10.4 (Color Island)**

On Color Island there are 13 blue chameleons, 15 yellow chameleons and 17 red chameleons. 20pt
When two chameleons of different colors meet, they change color into the third color. So for example, a blue and yellow chameleon meeting will result in two red chameleons. Is it possible that at some point all chameleons on the island have the same color? Write a formal description of this problem, as explained on the slides. What is one possible solution?

---

**Solution:**

Denote by $a$, $b$ and $c$ the number of blue, yellow and red chameleons respectively. Also, use the tuple $\langle a,b,c\rangle$ for a state $\mathcal{S}$ in our problem, This means that our initial state, $\mathcal{G}$ is given by $\langle 13,15,17\rangle$. Operators $\mathcal{O}$ between states can be described as transitions from a state $\langle a,b,c\rangle$ to either $\langle a-1,b-1,c+2\rangle$, $\langle a-1,b+2,c-1\rangle$ or $\langle a+2,b-1,c-1\rangle$.

The difference between the number of chameleons either does not change or changes by 3. Which means that the modulo-division by 3 produces an invariant result. Initially $a - b = -2$, but for all chameleons to have the same color, we need to have either $a - b = 0$ or $a - b = \pm 45$ in the $\mathcal{G}$ states, as we should have either 0 or 45 chameleons of each color (two of them are of course 0 and the remaining type has 45). Numbers 0 and $-2$ have different remainders when dividing by 3, hence the problem has no solution.

## Assignment 11: Astar-search
## (Given May 7., Due May. 14.)