General Computer Science II (320201) Spring 2012

Michael Kohlhase
Jacobs University Bremen

April 8, 2013

Contents

Assignment 1: Graph Theory(Given Feb. 10., Due Feb. 16.)

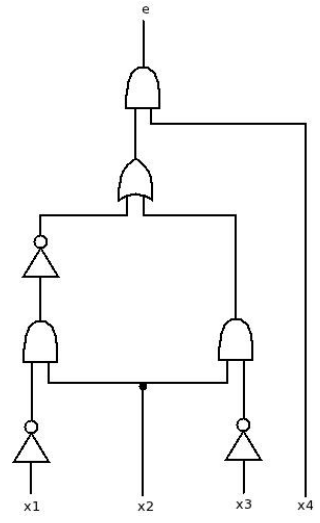
20pt

Problem 1.1 (Parse trees and isomorphism)
Let P_e be the parse-tree of $e := (\overline{\overline{x_1} * x_2} + x_2 * \overline{x_3}) * x_4$

1. Design a combinational circuit that represents P_e.
2. Write the mathematical representation of a graph G that is different but equivalent to P_e.

Solution:

1.



2. $G := \langle \{A, B, C, D, E, F, G, 1, 2, 3, 4\}, \{\langle A, B\rangle, \langle A, 4\rangle, \langle B, C\rangle, \langle B, E\rangle, \langle C, D\rangle, \langle D, F\rangle, \langle D, 2\rangle, \langle E, 2\rangle, \langle E, G\rangle, \langle F, 1$

Problem 1.2 (Tree Equivalences) 30pt

Let G be a graph with v vertices. Prove or refute that the following statements are equivalent:

- G is connected and it does not contain cycles.
- G contains no cycles, but adding one edge e will create exactly one cycle.
- Any two vertices of G are connected by strictly one path.

Solution:

- We can first prove the equivalence of the first and third statement. To prove: An acyclic connected graph has a unique path between any two of its vertices u and v, with $u, v \in V$. If G is connected then each pair of vertices can be connected by at least one path. If some pair is connected by more than one path, we choose the shortest pair $\langle P, Q \rangle$ of distinct paths with the same endpoints. Due to this choice, no internal vertex of P or Q will belong to the other path. Hence $P \cup Q$ is a cycle and contradicts the hypothesis.

  Conversely, if there exists one path between any u and v, then G is connected. If G has a cycle C, G would have at least two different paths between points in C, thus contradicting the initial assumption.

- Now we can prove the second claim using the others. As proven, a tree has a unique path linking each pair of vertices. This means that joining any two vertices u and v by edge e will create another path P between them, second to the initial one. So one can just consider $P \cup e$ and we have a cycle. More than one cycle could be formed only if there were at least two paths between u and v. But this would mean that there would already be a cycle in G, which is excluded.

  From the other direction, the argument is similar. If adding one edge to G creates a unique cycle, then we need to prove that initially G is acyclic and connected. If by adding any edge e to a graph we obtain a cycle C, then $\exists$ path P from any u to any v. Otherwise we could simply connect the isolated vertex to G via e. Also, the uniqueness of the cycles translates to G having not more than one path from any u to any v, so we have shown that G is connected and acyclic.

Problem 1.3 (In-degrees in acyclic digraphs) 25pt

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

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 1.4 (Graphs and SML) 30pt

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

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

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

Your tasks are:

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

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

3. Write the SML functions size and depth which return the size and the depth of a tree represented with an adjacency list.

Function signatures and example:

```
val getNeighbors = fn : (int * int list) list * int -> int list;
val isTree = fn : (int * int list) list -> bool;
val size = fn : (int * int list) list -> int;
val depth = fn : (int * int list) list -> int;

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

Solution:

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

Assignment 2: Number Systems and Combinatorial Circuits(Given Feb. 17., Due Feb. 23.)

25pt

Problem 2.1 (Gray Code)
A Gray code is a number encoding in which every two consecutive numbers differ by a single bit only. Below there is an example of a 2-bit Gray code:

Gray encodng	binary equivalent	decimal equivalent
00	00	0
01	01	1
11	10	2
10	11	3

- Design a 4-bit Gray code. You can provide your answer in a form similar to the table above. Keep in mind that every two consecutive numbers must differ by the value of only one of the four bits.
- Create a combinatorial circuit that takes a 4-bit number in Gray code as input, and outputs its 4-bit binary equivalent. You are only allowed to use XOR gates for this task.

Solution:

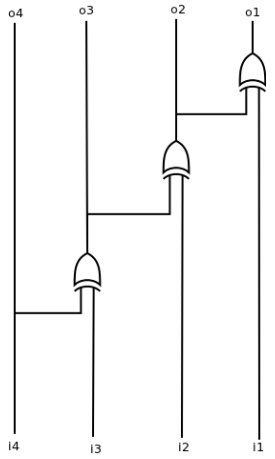
- There are various solutions for a 4-bit Gray code. One of them is the following:

Gray encodng	binary equivalent	decimal equivalent
0000	0000	0
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0010	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15

- Let us name the input bits i_1, i_2, i_3 and i_4, with i_1 being the least significant bit. Similarly, the output bits would be o_1, o_2, o_3 and o_4.
  By comparing the first two columns of the table above, we note the following:

- $o_4 = i_4$
- $o_3 = i_3$ if $i_4 = 0$ and $o_3 = \overline{i_3}$ if $i_4 = 1$. Thus $o_3 = XOR(i_3, i_4)$
- $o_2 = i_2$ if $i_3 = i_4$, i.e. if $o_3 = 0$, and $o_2 = \overline{i_2}$ if $i_3 = \overline{i_4}$, i.e. if $o_3 = 1$. Thus $o_2 = XOR(i_2, o_3)$
- $o_1 = i_1$ if there is an odd number of 0's among i_2, i_3 and i_4, and $o_1 = \overline{i_1}$ otherwise. Thus $o_1 = XOR(o_2, i_1)$

So the corresponding circuit is:



Problem 2.2 (Converting to decimal in SML) 30pt

Write an SML functions:

change_base n = fn : int -> string list

that takes a natural number and converts it into binary, octal or hexadecimal notations. Output all results as a list of strings: first decimal, next octal, last hexadecimal.

In your implementation, follow these steps:

1. First convert the number to base two by creating a function base2 n = fn : int -> string
2. Then, using this result, create 2 functions base2to16 n = fn : string -> string list and base2to8 n = fn : string -> string list that take the number in binary and convert it to octal and hexadecimal.

Please note that these steps are mandatory!

As an example consider

change_base 10 -> ["1010","12","A"]

Solution:

```
exception InvalidInput;
exception SomeError;


fun tobase2 0 = []
| tobase2 n = (n mod 2)::(tobase2 (n div 2));

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

fun make_char_list [] = []
|make_char_list(c::l) = if c=0 then (#"0")::make_char_list(l)
                                else (#"1")::make_char_list(l);

fun base2 n = implode(make_char_list(reverse (tobase2 n)));

fun find_int [] = []
|find_int (c::lc) = if 0 <= ord(c)-48 andalso ord(c)-48<=9 then (ord(c) - 48)::find_int(lc)
                                                          else raise InvalidInput;

fun resize8 l = if length(l) mod 3 = 1 then 0::(0::l)
                                       else if length(l) mod 3 = 2 then 0::l
                                            else l;

fun tobase8 [] = []
|tobase8 (a::b::c::l)= (a*4+b*2+c)::tobase8(l)
|tobase8 l = raise SomeError;

fun base2to8 l = tobase8(resize8(find_int (explode l)));

fun resize16 l = if length(l) mod 4 = 1 then 0::0::0::l
```

```
                                    else if length(l) mod 4 = 2 then 0::0::l
                                         else if length(l) mod 4 = 3 then 0::l
                                              else l;

fun make_special_char 10 = "A"
| make_special_char 11 = "B"
| make_special_char 12 = "C"
| make_special_char 13 = "D"
| make_special_char 14 = "E"
| make_special_char 15 = "F"
| make_special_char n = raise SomeError;

fun make_char(x:int) = if x div 10 = 0 then Int.toString x
                       else make_special_char x;

fun tobase16 [] = []
|tobase16 (a::b::c::d::l)= make_char(a*8+b*4+c*2+d)::tobase16(l)
|tobase16 l = raise SomeError;

fun base2to16 l = tobase16(resize16(find_int (explode l)));
```

Problem 2.3 (DNF Circuit with Quine McClusky) 20pt

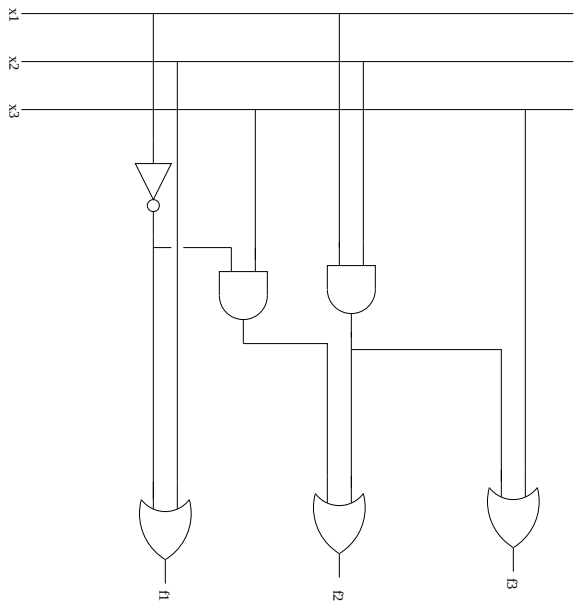
Use the technique shown in class to design a combinational circuit for the following Boolean function:

X_1	X_2	X_3	$f_1(X)$	$f_2(X)$	$f_3(X)$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	1	1	1

Solution: After using Quine-McCluskey and checking the prime implicants for their essentialness we conclude that the given functions are

$$\begin{aligned} f_1(X) &= \neg X_1 \vee X_2 \\ f_2(X) &= \neg X_1 \wedge X_3 \vee X_1 \wedge X_2 \\ f_3(X) &= \neg X_1 \end{aligned}$$

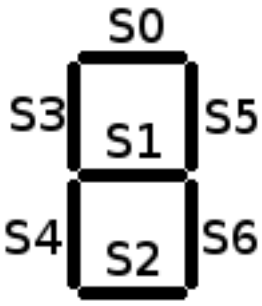
Hence the circuit for these functions can be designed as the following:



Problem 2.4 (Digit Display) 25pt

Suppose that you are given a set of 4 binary inputs that represent a digit (0-9) in binary (from '0000' for 0 to '1001' for 9). Design a circuit that will turn on (set output value to 1) the outputs that correspond to the digit.

Consider the segments in the order described in the image below:



For example, one could, for the digit 1, turn on segments $S3$ and $S4$, so only the outputs corresponding to these segments will be 1 if the input is '0001'.

Note: Write down in your solution the segments that should be on for each digit. Also, give a reasoning why your circuit is correct.

Solution: For each digit, one has to determine what segment wil be on. For example, the digit 2 should turn on segments $S0$, $S5$, $S1$, $S4$ and $S2$. The final implementation is dependant on the choice of digit-display, but for example one can use segment $S1$ in digits 2 (0010), 3 (0011), 4 (0100), 5 (0101), 6 (0110), 8 (1000), 9 (1001). Thus we can create the following truth table:

i_1	i_2	i_3	i_4	$S1$
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1

The next steps would be applying QMC to obtain the minimum polynomial and then implement it in a circuit:

$$\overline{i_1}\,\overline{i_2}\,i_3 + \overline{i_1}\,i_2\,\overline{i_3} + i_1\,\overline{i_2}\,\overline{i_3} + \overline{i_1}\,i_3\,\overline{i_4}$$

The same procedure should be done for every segment.

Assignment 3: TCN and Combinatorial Circuits(Given Feb. 24., Due Mar. 1.)

30pt

Problem 3.1 (TCN in SML)

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

- 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;
```

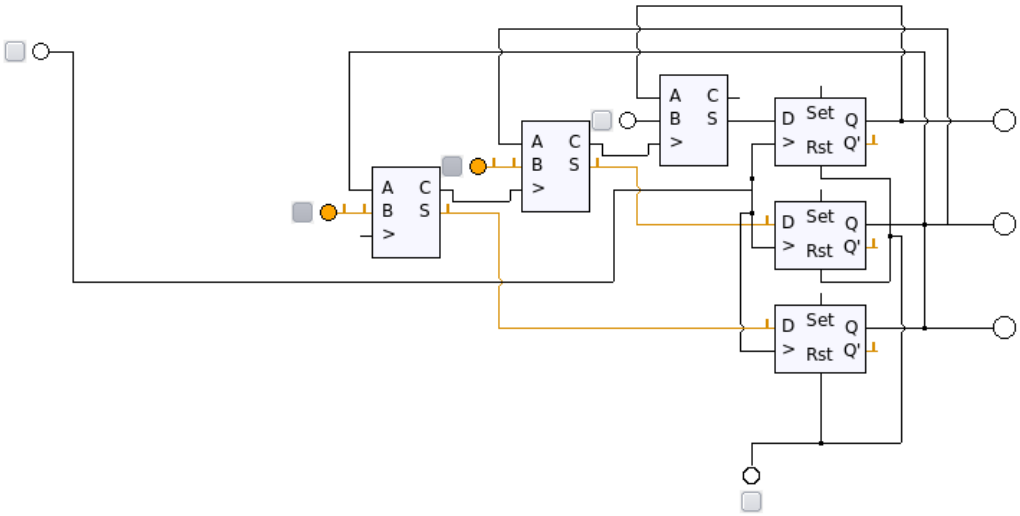
Problem 3.2 (3-Counter) 30pt

A digital counter is a device which, given a logical input, will output the binary representation of the number of times the input was turned on.

Your task is to design such a counter, which counts 3 times for every time when the input is on (i.e. it produces the sequence $0, 3, 6, 9, \ldots$ in binary). In order to keep the complexity of the circuit low, use only 3 outputs (i.e. produce the sequence $0, 3, 6, 1, 4, 7, 2, 5, 0...$, which is the previous sequence modulo 8).

Also include a short description of how your circuit works and why does it accomplish its task.

Solution: Below is my implementation in KTechLab:



The operational principle there is that we store in the D-latches the current number, which we feed as one of the terms of a 3-bit adder (built as a CCA); the other term is the number 3, which is represented as 011 in binary (note that the upper bit is the most significant in the outputs/inputs). The sequence 011 is fixed (should not be changed by the user), i.e. the only input required is the logical input on the left. There is alo a logical input on the lower side of the diagram, which is used as a universal "reset button".

Problem 3.3 (Cost and depth of adders) 15pt

What is the cost and depth of an n-bit CCA? What about the n-bit CSA (for cost, big-O is enough)? Now what if we construct a new adder, that computes the two cases for the first half of the input just like CSAs do (and of course uses a multiplexer), but only does this once, and the $\frac{n}{2}$-bit adders are not also CSAs, but CCAs (so only one multiplexer is used overall) - what would the cost and depth of this adder be?

Solution: The CCA has depth $3n$ and cost $5n$, as shown in the slides. The CSA has depth $3\log(n) + 3$ and cost of complexity order n^{log3}. For the new adder, the depth is the one of the $\frac{n}{2}$-bit CCA, plus the $\frac{n}{2}$-bit multiplexer, which is 3. Thus the depth is $\frac{3n}{2} + 3$. The cost is that of three CCAs and one multiplexer, so $\frac{5n}{2} \cdot 3 + \frac{3n}{2} + 1$.

Problem 3.4 (Detecting overflow in TCN addition) 30pt

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:

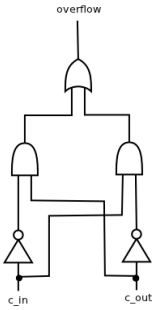
   - $-1_{10} = 1111_{TCN}$, $7_{10} = 0111_{TCN}$, $1111_{TCN} + 0111_{TCN} = 0110_{TCN} = 6_{10}$, thus the addition was proper
   - $-5_{10} = 1011_{TCN}$, $-4_{10} = 1100_{TCN}$, $1011_{TCN} + 1100_{TCN} = 0111_{TCN} = 7_{10} \neq -9_{10}$, thus an overflow occured

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:



Assignment 4: Machine Languages(Given Mar. 2., Due Mar. 8.)

35pt

Problem 4.1 (Accumulator Circuit)

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

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.

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.

Problem 4.2 (Greatest Common Divisor) 30pt

Euclid's method to determine the GCD of two numbers can easily be expressed in pseudocode in the following way:

```
function gcd(a,b):
  if ( a<b )
    swap(a, b);
  while ( b!=0 ) do
    r = a mod b;
    a = b;
    b = r;
  end while
  return a;
end function
```

Of course, there is no such thing as mod or div in ASM, but you can simulate it. You are asked to output the GCD of N numbers in position $P(1)$. You are given N in $P(0)$ and the N numbers on positions $P(11..N+10)$. You can use $P(1..10)$ freely (pay attention, of course, that the result should appear in $P(1)$).

Note: Be sure to thoroughly explain your code, and briefly explain it in plain English.

Solution: My solution is based on the following steps written in a C-like language. Moreover, I am using the hint for emulating function calls; this might produce inefficient code, but this is not a matter of concern right now. I have mentioned in a comment where I am placing the temporary variables or arguments of each function:

```
main() { /* i=D(1) */
    i = N;
    do {
        a[i+9] = gcd(a[i+9], a[i+8]);
        i--;
    } while (i>1);
}


gcd(a, b) { /* a=D(2), b=D(3), r=D(4) */
    if ( a<b )
        swap(a, b);
    do {
        r = mod(a, b);
        a = b;
        b = r;
    } while ( b>0 )
    return a;
}

swap(a, b) { /* a=D(4), b=D(5), r=D(6) */
    r = a;
    a = b;
    b = r;
}
```

```
mod(a, b) { /* a=D(4), b=D(5) */
    while ( a>=b ) {
        a -= b;
    }
    return a;
}

div(a, b) { /* a=D(5), b=D(6), x=D(7) */
    x = 0;
    while ( a>=b ) {
        x++;
        a -= b;
    }
    return x;
}
```

Now, translated to ASM, this would look like:

```
LOAD 0 ; prepare jump to gcd
MOVE ACC IN1
LOADIN1 9
STORE 2
LOADIN1 8
STORE 3 ; ready for jump to gcd
LOAD 2
SUB 3
JUMP(>) 7 ; no need for swap
LOAD 2 ; swap
STORE 4
LOAD 3
STORE 2
LOAD 4
STORE 3
LOAD 2 ; prepare jump to mod
STORE 4
LOAD 3
STORE 5
LOAD 4 ; jump to mod
SUB 5
STORE 4
LOAD 4
JUMP(>=) -4
LOAD 4
ADD 5
STORE 4 ; return from mod to main
LOAD 2 ; prepare jump to div
STORE 5
LOAD 3
STORE 6
LOAD 5 ; jump to div
STORE 7
LOAD 7 ; I had an error in my algorithm. div is just bypassed now.
STORE 2 ; return from div to main
LOAD 4
```

```
STORE 3
LOAD 3
JUMP(>) −41 ; while in the gcd
LOAD 1
MOVE ACC IN1
LOAD 2
STOREIN1 8
LOAD 1
SUBI 1
STORE 1
LOAD 1
SUBI 1
JUMP(>) −57
LOAD 11
STORE 1
STOP 0
```

Problem 4.3 (Prime numbers) 20pt

You are given a number X in $\mathcal{S}(0)$. Write a $\mathcal{L}(\texttt{VM})$ program that will output 0 in $\mathcal{S}(1)$ if X is prime or 1 otherwise.

Solution: A theorem states that a number X is prime if it does not have any divisor between 2 and $\sqrt{X}$. It is less efficient, but easier to test for divisor using each integer between 2 and $X-1$. Therefore, we could develop the following C-like program:

```
main() { /* i=D(1); */
    i = 2;
    while (i<=N-1) {
        if ( mod(N, i) == 0 )
            return 1;
        ++ i;
    }
    return 0;
}

mod(a, b) { /* a=D(2); b=D(3) */
    while ( a>=b ) {
        a -= b;
    }
    return a;
}
```

The $\mathcal{L}(\texttt{VM})$ implementation is:

```
con 2
poke 1
peek 0 ; prepare mod
poke 2
peek 1
poke 3
peek 3 ; jumped in mod
peek 2
sub
poke 2 ; a -= b
peek 3
peek 2
leq ; b<=a
con 1
sub ; this part is equivalent to !(b<=a), or (a>b)
cjp -9 ; while loop in mod
peek 2 ; back in main
con 1
sub ; !mod(a,b)
cjp 15 ; we have a non prime number
peek 1
con 1 ; increase iterator
add
poke 1
peek 0
peek 1
sub ; i-N
```

```
cjp 2 ; if i==N exit while
jp -27 ; while loop in main
con 0 ; we exit the main loop, we have prime number
poke 1
halt
con 1 ; we were interrupted, we have non-prime number
poke 1
halt
```

Problem 4.4 (SW TCN Converter) 20pt

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.

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.

Assignment 5: Machine Languages(Given Mar. 2., Due Mar. 8.)

35pt

Problem 5.1 (Simulating REMA in SML)

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
        | 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

        (* 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 5.2 (SW simulator in SML) 25pt

Using the definitions that appear on slide 314, write an SML function run : program -> int that, given a program, will return the output value that results after running the simulated SW program.

For example, the following is a valid run:

```
run(([(''n'', Con 12), (''m'', Con 15)],
    While( Leq(Con 1, Var ''n''), Seq([
      Assign(''m'', Add(Var ''m'', Con 2)),
      Assign(''n'', Sub(Var ''n'', Con 1))
      ])),
    Var ''m'');
val it = 39;
```

You may consider the input program to be sintactically correct.

Solution: The following clean solution was written by Alexandra Zayets in 2011:

```
type id = string;
datatype exp = Con of int
             | Var of id
             | Add of exp*exp
             | Sub of exp*exp
             | Mul of exp*exp
             | Leq of exp*exp;

datatype sta = Assign of id*exp
             | If of exp*sta*sta
             | While of exp*sta
             | Seq of sta list ;

type declaration = id * exp;

type program = declaration list * sta * exp;

(*finds the value of a variable, if variable not declared initializes to 0*)
fun find_value (x :id, []: declaration list) = 0
  | find_value (x, (y,m)::(l:declaration list)) = if x = y then evaluate(m, (y,m)::l) else find_value (x,l)

(*evaluates an expression*)
and evaluate (Con (a), l:declaration list) = a
  | evaluate (Var (x), l) = find_value (x,l)
  | evaluate (Add (x, y),l) = evaluate(x,l) + evaluate(y,l)
  | evaluate (Sub (x, y),l) = evaluate(x,l) - evaluate(y,l)
  | evaluate (Mul (x, y),l) = evaluate(x,l) * evaluate(y,l)
  | evaluate (Leq (x, y),l) = if evaluate(x,l) <= evaluate(y,l)then 1 else 0;

(* assignes a new value to a variable if the variable is not declared no change is made*)
fun assign ( x: id, y:exp, []:declaration list) = ([]:declaration list)
  | assign ( x: id, y:exp, (z,m)::l) = if x = z then (z, y)::l else (z,m)::assign(x, y, l);

(*given a declaration list, executes a statement*)
fun execute (Assign (x, e), l) = assign (x, Con (evaluate(e, l)), l)
```

```
  | execute (If(x:exp, s:sta , p:sta),l) = if( evaluate (x, l) = 1) then execute(s, l) else execute (p, l)
  | execute (While(x:exp, s:sta), l) = if evaluate(x,l) = 0 then l else execute (While(x,s), execute (s, l))
  | execute (Seq([]), l) = l
  | execute (Seq(a::b), l) = execute (Seq(b), execute(a,l));

fun run ((a,b,c):program) = evaluate (c, execute(b, a));
```

Problem 5.3 (Exponential function in $\mathcal{L}(\texttt{VMP})$) 20pt

The exponential function has the following power series representation:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{(n)!} = 1 + x + \frac{x^2}{(2)!} + \frac{x^3}{(3)!} + \dots$$

You are required to write an $\mathcal{L}(\texttt{VMP})$ procedure that computes this sum up to rank k:

$$s(x,k) = \sum_{n=0}^{k} \frac{x^n}{(n)!} = 1 + x + \frac{x^2}{(2)!} + \dots + \frac{x^k}{(k)!}$$

Both arguments of the procedure are integers. Also, consider all the divisions to be integer divisions (that is, the result of the division is an integer).

Solution: Here is the solution:

```
<pow>   proc 2 (length)
        arg 2 con 1 leq cjp 5
        arg 1 return
        con 1 arg 2 sub arg 1 call <pow>
        arg 1 mul return
<fact>  proc 1 (length)
        arg 1 con 1 leq cjp 5
        con 1 return
        con 1 arg 1 sub call <fact>
<div>   proc 2 (length)
        arg 1 arg 2 leq cjp 5
        con 0 return
        arg 2 con 1 arg 1 sub call <div>
        con 1 add return
<exp>   proc 2 (length)
        arg 1 con 0 leq cjp 5
        con 1 return
        arg 2 call <fact>
        arg 2 arg 1 call <power>
        call <div>
        con 1 arg 2 sub call <exp>
        add return
```

Problem 5.4 (μML Mystery) 25pt

You are given the following piece of μML code:

```
(
[
("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 μML program that does the same thing in a different way. Remember that a μML 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]) );
```

Assignment 6: Machine Languages(Given Mar. 16., Due Mar. 29.)

20pt

Problem 6.1 (TM for logarithm)

The logarithm of a number $n \geq 1$, denoted by $\log_2(n)$, is the maximum natural number p such that $2^p \leq n$. You are required to describe a Turing Machine that receives a string of n characters '1' surrounded by a hash at each end, and halts after outputting a string of $\log_2(n) + 1$ characters '1' after the last hash. The tape is right-infinite and filled with special characters (blanks).

Example run:

initial	#1111111111111111#
final	#1111111111111111#11111

You are allowed to use any alphabet. You are **not required** to write the transition table for each state, but **shortly describe** what is the purpose of each state you use.

Note: Your explanation should be clear enough and cover all the cases that might arise. In some situations, writing the transition table might be more clear than using plain English.

Solution: The general idea is to follow the following steps:

1. append a 1 after the last hash
2. proceed to divide the 1's between hashes in two by marking one 1 from the beginning with 0 and a matching 1 from the end with 2
3. revert all 0's back to 1's, go to the beginning of the tape and start again

The clearest solution is a table with explanations:

1	#	1	#	>
1	1	2	1	>
2	1	2	1	>
2	#	3	#	>
3		4	1	<
4	1	4	1	<
4	#	5	#	<
5	2	5	2	<
5	1	6	2	<
6	1	6	1	<
6	#	7	#	>
6	0	7	0	>
7	1	8	0	>
8	1	8	1	>
8	2	5	2	<
5	0	5	1	<
5	#	1	#	>
2	2	2	2	>
3	1	3	1	>
7	2	5	2	<

Problem 6.2 (Turing Machine for Binary to Octal) 20pt

You have to design a Turing Machine that converts a binary number to octal for this problem. You are given an infinite tape with a sequence of binary digits (0s and 1s) surrounded by # (see example). You are requested to output, after the second #, the equivalent number written in base 8. The tape has an infinite number of spaces (blank cells) after the second #.

Note: Both the binary number and the resulting octal representation have the least significant digit first!

Note: You are granted that the number of binary digits is divisible by 3.

Example:

initial	#010110111011#
possible result	#010110111011#2376#

Solution: Roughly, the student can design 8 states to identify the octal digit that is represented by the sequence of bits (and also mark these bits as "seen"). Then, the machine should navigate right until the first blank cell and fill there the corresponding digit. Another state would navigate back to the first unmarked bit.

Problem 6.3 (Halting Reductions) 25pt

The fact that a TM cannot decide if another TM halts on a given input is not the only limit of computation. There are a lot of other things TM's cannot do, and the halting problem can be used to prove this. This process is called "reduction to the halting problem": for proving that a TM cannot decide a certain a property P, assume that it could and then use it to construct another TM that can decide the halting problem (i.e. to decide if some TM halts on some given input).

For the following statements, provide a proof by reduction to the halting problem or a counterexample:

- No TM can decide in general whether another TM halts on all inputs.
- TM can decide in general whether another TM uses all its states in the computation on a given input x.

Solution:

- Construct K such that it halts on every input but x, and on x it simulates N. Then use M on K, and if the output is yes, then it means N halted on x, otherwise no.
- Construct K such that it simulates N on x and if it halts, then it goes through all the states of N. This means that N halting on x is equivalent to K uses all its states on x (since if N doesn't halt, then the halting state will not be used). Then run M on K.

Problem 6.4 (SML Implementation of a Turing Machine Simulator) 35pt

Let us try to simulate a Turing Machine in SML. The states and the move directions shall be given as

```
datatype State = q of int
datatype Direction = L | R
```

Introduce the type Alphabet according to the problem you wish to test. Write an SML function TM: TransTrable -> Tape -> Tape that simulates a Turing Machine, where

```
type Tape = Int -> Alphabet
type TransTable = ((State * Alphabet) * (State * Alphabet * Direction)) list
```

You can use the other tasks from this homework, problems for the slides or come up with simple own examples to test your implementation.

Solution:

```
datatype State = q of int
datatype Alphabet = zero | one
datatype Direction = L | R

type Tape = int -> Alphabet
type TransTable = ((State * Alphabet) * (State * Alphabet * Direction)) list

(* takes table, current state and symbol read and returns the appropriate row from transition table *)
fun find_row(nil, st, sym) = (0,zero,~1,zero, L) |
    find_row(hd::tl, st, sym) = let
                val (ost,rdsym,_,_,_) = hd
        in
                if (ost = st) andalso (rdsym = sym) then hd
            else find_row(tl, st, sym)
        end

(* the main guy *)
fun myTM(table, tape, head_pos, state) = let
        val (_,_,new_state,new_symbol,dir) = find_row(table, state, tape(head_pos))
        val new_head_pos = if dir = R then head_pos+1 else head_pos-1
        fun new_tape(n) = if n = head_pos then new_symbol else tape(n)
in
        if new_state = ~1 then tape (*if there is no appropriate row in the table, it halts*)
        else myTM(table, new_tape, new_head_pos, new_state)
end

(* assumed: initial head position = 1, initial state of the head = 1, states non-negative numbers*)

fun TM table tp = myTM(table, tp, 1, 1);

(* Test 1: half bit adder *)

val MYTAPE_11 = fn 1 => one | 2 => one | n => zero;
val MYTAPE_10 = fn 1 => one | 2 => zero | n => zero;
val MYTAPE_01 = fn 1 => zero | 2 => one | n => zero;
val MYTAPE_00 = fn 1 => zero | 2 => zero | n => zero;
```

```
val MYTABLE = [(1, zero, 2, zero, R), (1, one, 3, one, R),
               (2, zero, 4, zero, R), (2, one, 5, one, R),
                   (3, zero, 5, zero, R), (3, one, 6, one, R),
                            (4, zero, 7, zero, R), (4, one, 7, zero, R),
                            (7, zero, 9, zero, L), (7, one, 9, zero, L),
                            (5, zero, 8, zero, R), (5, one, 8, zero, R),
                            (8, zero, 9, one, L), (8, one, 9, one, L),
                            (6, zero, 7, one, R), (6, one, 7, one, R)];

val result_11 = TM MYTABLE MYTAPE_11; (* Inspect result_11 3 to see carry, result_11 4 to see sum *)
val result_10 = TM MYTABLE MYTAPE_10;
val result_01 = TM MYTABLE MYTAPE_01;
val result_00 = TM MYTABLE MYTAPE_00;

(* Test 2: one's duplicator *)

val MYTAPE2 = fn n => if n<=6 andalso n>=1 then one else zero; (*...000111111000...*)

val MYTABLE2 = [(1, one, 2, zero, R),
                (2, one, 2, one, R),
                                (2, zero, 3, zero, R),
                                (3, one, 3, one, R),
                                (3, zero, 4, one, L),
                                (4, one, 4, one, L),
                                (4, zero, 5, zero, L),
                                (5, one, 5, one, L),
                                (5, zero, 1, one, R)];

val result2 = TM MYTABLE2 MYTAPE2;

(* another *)

val TAPE = fn 1 => zero |
                            2 => zero |
                            3 => zero |
                            _ => one;

val TABLE = [(1,zero,1,one,R),(1,one,2, one, R)];
val RESULT = TM TABLE TAPE;

(* another2 :) *)

val TAPE2 = fn 1 => one |
                            2 => one |
                3 => one |
                            _ => zero;

val TABLE2 = [(1,one,2,zero,R), (2,one,2,one,R), (2, zero, 3, zero, L), (3, one, 4, zero, L),
(4, one, 4, one, L), (4, zero, 1, zero, R), (3, zero, 5, one, R)];

val RESULT2 = TM TABLE2 TAPE2;
```

Assignment 7: Internet Introduction(Given Mar. 30., Due Apr. 12.)

20pt

Problem 7.1 (HTML basics)
Answer the following questions about HTML:

1. What does HTML stand for?
2. Who is making the Web standards?
3. What is HTML tag for the largest heading?
4. What is the correct HTML tag for inserting a line break?
5. What is the correct HTML for adding a background color?
6. What is the correct HTML tag to make a text bold?
7. What is the correct HTML tag to make a text italic?
8. What is the correct HTML for creating a hyperlink?
9. How can you create an e-mail link?
10. How can you open a link in a new browser window?
11. Which of these tags are all <table> tags?
    - <thead><body><tr>
    - <table><head><tfoot>
    - <table><tr><tt>
    - <table><tr><td>
12. What is the correct HTML to left-align the content inside a tablecell?
13. How can you make a list that lists the items with numbers?
14. How can you make a list that lists the items with bullets?
15. What is the correct HTML for making a checkbox?
16. What is the correct HTML for making a text input field?
17. What is the correct HTML for making a drop-down list?
18. What is the correct HTML for making a text area?
19. What is the correct HTML for inserting an image?

20. What is the correct HTML for inserting a background image?

Solution:

1. Hyper Text Markup Language
2. The World Wide Web Consortium
3. <h1>
4.
5. <body style="background−color:yellow">
6. <b>
7. <i>
8. WordToBeLinked
9. <a href="mailto:xxx@yyy">
10. <a href="url" target="_blank">
11. <table><tr><td>
12. <td align="left">
13.
14.
15. <input type="checkbox" />
16. <input type="text" />
17. <select>
18. <textarea>
19.
20. <body background="background.gif">

Problem 7.2 (Web browsers) 20pt

- What is the difference between a web page and a web site?
- What is a web browser? Name at least 5 practical web browser tools.

Solution:

- A web page is a document on the Web that can include multimedia data. A web site is a collection of related Web pages usually designed or controlled by the same individual or company.
- A web Browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web, enabling users to view Web pages and to jump from one page to another.

  Practical web browser tools: Status Bar, Bookmarks, View Source, history, temporary Internet files, home page, auto complete, security settings, programs, etc.

Problem 7.3 (Quiz for the TAs) 45pt

Your last assignment this semester is to give your TAs a quiz. We hope you will enjoy this :)

You need to create a form in HTML that contains the following:

1. Include at least 5 multiple choice questions.
2. All following concepts: button, radio button, check box, drop down box, text input.
3. At least one image and one working link.
4. Tables, lists.
5. Make it look nice overall (styles, colors ...)

You can provide a fictive action attribute.

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

35pt

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:

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) 35pt

Now that you managed to send your first email, you have to write the following SML functions:

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: Web technologies(Given Apr. 22., Due May. 3.)

75pt

Problem 9.1 (SML Web Crawler)

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 9.2 (Ranking pages) 25pt

In this task you will gain some practical experience with a real-world web crawler and you will come up with your own page ranking procedure!

Look into the man pages of wget (available on `linux`, use the tlab machines if you don't have `linux` already on your laptop; you might also find `Windows` ports of the program). wget has the ability to follow links while saving the pages to disk, and also to keep the directory structure consistent with the server.

Choose a web page of your preference (we recommend using a wikipedia page) and run wget with a depth limit of your choice. Now inspect the output directory and observe items that might help you in ranking your web pages (for example, number of links pointing to a web page, number of images, length of the content or its age might be starting points!). Do not reinvent the wheel, or reverse-engineer the Google PageRank algorithm! Be creative and make a good use of the features that your starting page has (wikipedia has, for example, the links between related topics). Also, do not take into consideration whether the features are (easily) computable.

You will have to supply a PDF document reporting your actions. Describe how you used wget to mirror the site (do include the commands used!). Describe your ranking function (what items you consider, how they influence the page score). Compile a table which contain these items, the score of each item for each page and the final score of the page.

Finally, write down your observations and comments about the method that you employed.

Solution:

Assignment 10: Web technologies(Given May. 4., Due May. 10.)

20pt

Problem 10.1 (Color Island)

On Color Island there are 13 blue chameleons, 15 yellow chameleons and 17 red chameleons. 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.

Problem 10.2 (Missionaries and cannibals) 30pt

Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. The final goal is to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

1. Formulate the problem precisely. When defining the operators, it is not necessary that you write every possible *state* $\rightarrow$ *state* combination, but you should make it clear how one would derive the next state from the current one.

2. Suppose the next-function for depth first search (DFS) and breadth first search (BFS) expands a state to its successor states using the operators you have defined in 1. in the order you have defined them. Operators that leave more cannibals than missionaries on one side will not be considered. Likewise, operators that lead to the immediate previous state will not be considered (e.g., after moving a cannibal from left to right, the next-function for this state will not include a state where a cannibal moves from right to left). Draw the search tree till depth 3. What are the first 5 nodes explored by DFS? What are the first 5 nodes explored by BFS?

3. If you would implement this problem, would you rather use BFS or DFS to find the solution? Briefly explain why?

Solution:

1. Here is one possible representation:

   The State Space is a six-tuple of integers listing the number of missionaries, cannibals, and boats on the first side, and then the second side of the river. The goal is a state with 3 missionaries and 3 cannibals on the second side. The cost function is one per action, and the successors of a state are all the states that move 1 or 2 people and 1 boat from one side to another.

   The **Initial State** is $(3, 3, 1, 0, 0, 0)$

   The **Final State** is $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{3}, \mathbf{3}, \mathbf{1})$

   Operators: Unless the next state leaves more cannibals then missionaries on one side, and unless the transition is impossible (eg: a movement always occurs from the side where the boat is to the other) rules are as follows:

   (a) Move a missionary to the other side.

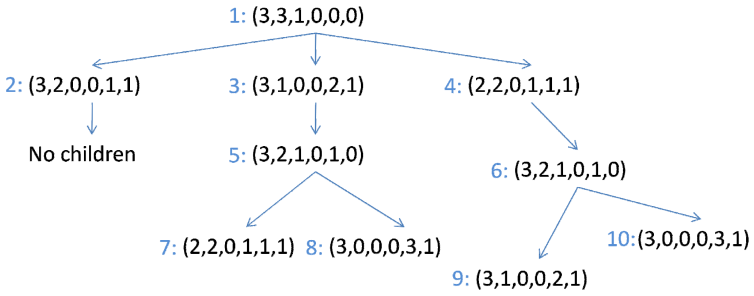
   (b) Move a cannibal to the other side.

   (c) Move 2 missionaries to the other side.

   (d) Move 2 cannibals to the other side.

   (e) Move a missionary and a cannibal to the other side.

2. Depends in which order the operators are defined. In the case above, it will look as follows:



3. BFS, since the branching factor is not big (we won't have memory problems), and DFS may get stuck in loops.

Problem 10.3 (Implementing Search) 30pt

Implement the depth-first and breadth-first search algorithms in SML. The corresponding functions dfs and bfs take three arguments that make up the problem description:

1. the initial state
2. a function next that given a state x in the state tree returns at set of pairs (action,state): the next states (i.e. the child nodes in the search tree) together with the actions that reach them.
3. a predicate (i.e. a function that returns a Boolean value) goal that returns true if a state is a goal state and false else.

The result of the functions should be a pair of two elements:

- a list of actions that reaches the goal state from the initial state
- the goal state

The signatures of the two functions should be:

```
dfs : 'a -> ('a -> ('b * 'a) list) -> ('a -> bool) -> 'b list * 'a
bfs : 'a -> ('a -> ('b * 'a) list) -> ('a -> bool) -> 'b list * 'a
```

where 'a is the type of states and 'b is the type of actions.

In case of an error or no solution found raise an InvalidSearch exception.

Solution:

```
exception InvalidSearch;
val tick = false; (* used for debugging *)

local

    fun add_actions x nil = nil
      | add_actions x ((a,s)::l) = (x @ [a],s)::(add_actions x l);

    fun depthFirst_strategy nil next = raise InvalidSearch
      | depthFirst_strategy ((a,s)::l) next = ( add_actions a (next s) ) @ l;

    fun breadthFirst_strategy nil next = raise InvalidSearch
      | breadthFirst_strategy ((a,s)::l) next = l @ ( add_actions a (next s) );

    fun sl strategy nil next goal = raise InvalidSearch
      | sl strategy ((a,s)::l) next goal =
        let
        val _ = if tick then print "#" else print "";
in
                    if goal(s)
                then (a,s)
                else
                    let
                    val new_fringe = strategy ((a,s)::l) next;
```

```
                    in
                        sl strategy new_fringe next goal
                    end
        end;

        fun search strategy i next goal =
                if goal(i)
                then (nil,i)
        else sl strategy (add_actions nil (next i)) next goal;
in
        fun dfs i next goal = search depthFirst_strategy i next goal;
        fun bfs i next goal = search breadthFirst_strategy i next goal;

end;
```

Solution:

```
(* TEST CASES *)

datatype action = a1to2 | a1to4 | a1to5 | a2to3 | a4to5 | a4to6 | a5to1 | a5to7 | a3to6;
datatype state = one | two | three | four | five | six | seven;

fun next1(one) = [(a1to2,two),(a1to4,four),(a1to5,five)]
  | next1(two) = [(a2to3,three)]
  | next1(three) = [(a3to6,six)]
  | next1(four) = [(a4to5,five),(a4to6,six)]
  | next1(five) = [(a5to1,one),(a5to7,seven)]
  | next1(six) = []
  | next1(seven) = [];

fun next2(one) = [(a1to2,two),(a1to4,four),(a1to5,five)]
  | next2(two) = [(a2to3,three)]
  | next2(three) = [(a3to6,six)]
  | next2(four) = [(a4to5,five),(a4to6,six)]
  | next2(five) = [(a5to7,seven),(a5to1,one)]
  | next2(six) = []
  | next2(seven) = [];

fun goal1(six) = true
  | goal1(_) = false;

fun goal2(four) = true
  | goal2(three) = true
  | goal2(_) = false;

fun goal3(seven) = true
  | goal3(_) = false;

val test4 = bfs one next1 goal1 = ([a1to4,a4to6],six);
val test5 = dfs one next1 goal1 = ([a1to2,a2to3,a3to6],six);
val test6 = bfs one next1 goal2 = ([a1to4],four);
val test7 = dfs one next1 goal2 = ([a1to2,a2to3],three);
val test8 = bfs one next2 goal3 = ([a1to5,a5to7],seven);
```

```
val test9 = dfs one next2 goal3 = ([a1to4,a4to5,a5to7],seven);
val test10 = bfs one next1 goal3 = ([a1to5,a5to7],seven);
val test11 = dfs one next1 goal3; (*should run endlessly*)
```

Problem 10.4: Write the next function, goal predicate and initial_state variable for the 8-puzzle presented on the slides (please check the slides for the description). Then use these to test your breadth-first and depth-first search algorithms from the previous problem. 30pt

Use the following :

```
datatype action = left|right|up|down;
type state = int list;(*9 elements, in order, 0 for the empty cell*)
```

Refer to the slides for the initial_state variable. Make sure that if an action is illegal for a certain state, it does not appear in the output of next.

Sample testcase:

```
test call : next(initial_state);

output: [(left,[7,2,4,0,5,6,8,3,1]),(right,[7,2,4,5,6,0,8,3,1]),
   (up,[7,0,4,5,2,6,8,3,1]),(down,[7,2,4,5,3,6,8,0,1])];
```

Solution:

```
datatype action = left|right|up|down;
type state = int list;(*9 elements, in order, 0 for the empty cell*)

val initial_state=[7,2,4,5,0,6,8,3,1];

fun goal(state)=if state=[1,2,3,4,5,6,7,8,0]
                then true
                else false;


(*invert a list*)
fun invert(a::l)=invert(l)@[a]
|invert(nil)=nil;


(*compute for left action*)
fun next_left(0::l)=nil
|next_left(a::b::c::0::e::f::g::h::i::nil)=nil
|next_left(a::b::c::d::e::f::0::h::i::nil)=nil
|next_left(a::0::l)=0::a::l
|next_left(hd::l)=hd::next_left(l);


(*to compute for right, just invert the list and call left*)
fun next_right(state)=invert(next_left(invert(state)));


(*this rearranges the list so that left can be used also for up and down*)
fun rearrange(a::b::c::d::e::f::g::h::i::nil)=[a,d,g,b,e,h,c,f,i]
|rearrange(nil)=nil;


(*to compute the up, rearrange the list and then call left*)
fun next_up(state)=rearrange(next_left(rearrange(state)));
```

```
(*to compuet down, invert and rearrange the list and then call left*)
fun next_down(state)=invert(rearrange(next_left(rearrange(invert(state)))));


(*gets rid of the nil ones, for which the action cannot be performed*)
fun make_list((act,l)::tl)=if (l=nil)then make_list(tl)
                           else (act,l)::make_list(tl)
|make_list(nil)=nil;


fun next(state)=let val x=next_left(state);
                    val y=next_right(state);
                    val z=next_up(state);
                    val t=next_down(state);
                in
                    make_list([(left,x),(right,y),(up,z),(down,t)])
                end;


(*Testcases*)

next(initial_state)= [(left,[7,2,4,0,5,6,8,3,1]),(right,[7,2,4,5,6,0,8,3,1]),
   (up,[7,0,4,5,2,6,8,3,1]),(down,[7,2,4,5,3,6,8,0,1])];

next([7,2,4,0,3,6,8,5,1])=[(right,[7,2,4,3,0,6,8,5,1]),(up,[0,2,4,7,3,6,8,5,1]),
   (down,[7,2,4,8,3,6,0,5,1])];(*cannot do left*)

next([7,2,4,8,3,6,0,5,1])=[(right,[7,2,4,8,3,6,5,0,1]),(up,[7,2,4,0,3,6,8,5,1])];(*connot do left or down*)

next([7,2,0,8,3,6,4,5,1])=[(left,[7,0,2,8,3,6,4,5,1]),(down,[7,2,6,8,3,0,4,5,1])]; (*cannot do right or up*)
```

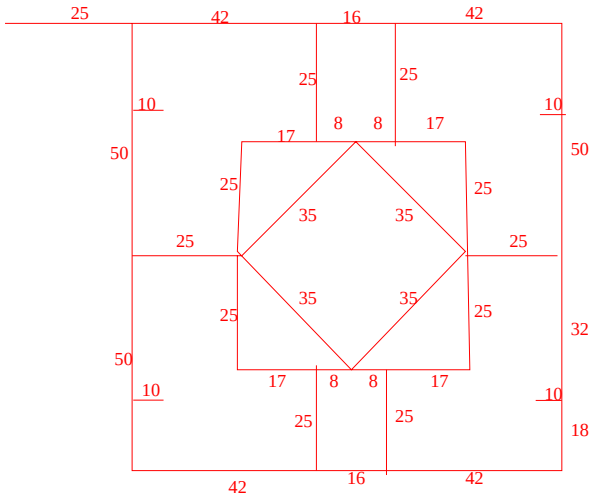
Assignment 11: A* Search and Prolog (bonus assignment)(Given May. 14., Due May. 21.)

35pt

Problem 11.1 (A^* **search on Jacobs campus**)

Implement the A^* search algorithm in SML and test it on the problem of walking from the main gate to the entrance of Research 3 with linear distance as heuristic. The length of line segments are annotated in the map below.

No function signature is provided, instead at the end of your program call your function so that it prints the actions needed to reach the entrance and the associated cost.



Solution:

```
val it = (["E","E","S","E","SE","E","S","W"],202) (* The states here are directions e.g. SE means Southeast. *)

fun coor 1 = (0,0) | coor 2 = (25,0) | coor 3 = (67,0) | coor 4 = (83,0) | coor 5 = (125,0) |
    coor 6 = (25,18) | coor 7 = (35,18) | coor 8 = (115,18) | coor 9 = (125,18) | coor 10 = (50,25) |
        coor 11 = (67,25) | coor 12 = (75,25) | coor 13 = (83,25) | coor 14 = (100,25) | coor 15 = (25,50) |
        coor 16 = (50,50) | coor 17 = (100,50) | coor 18 = (125,50) | coor 19 = (50,75) | coor 20 = (67,75) |
        coor 21 = (75,75) | coor 22 = (83,75) | coor 23 = (100,75) | coor 24 = (25,82) | coor 25 = (35,82) |
        coor 26 = (115,82) | coor 27 = (125,82) | coor 28 = (25,100) | coor 29 = (67,100) | coor 30 = (83,100) |
        coor 31 = (125,100);

val edges = [(1,2), (2,3), (3,4), (4,5), (6,7), (8,9), (10,11), (11,12), (12,13), (13,14), (15,16), (17,18),
                        (19,20), (20,21), (21,22), (22,23), (24,25), (26,27), (28,29), (29,30), (30,31),
                        (2,6), (6,15), (15,24), (24,28), (10,16), (16,19), (3,11), (20, 29), (4,13), (22,30),
```

```
                        (14,17), (17,23),
                        (5,9), (9,18), (18,27), (27,31),
                        (12,16), (16,21), (21,17), (17,12) ];

fun heuristic(n,m) = let
 val (x1,y1) = coor(n);
 val (x2,y2) = coor(m);
in Real.round(Math.sqrt(Real.fromInt((x1−x2)∗(x1−x2)+(y1−y2)∗(y1−y2))))
end;

fun next(n) = let
        fun successors(_,nil) = nil |
            successors(n,(a,b)::tl) = if n=a then b::successors(n,tl)
                                                                    else if n=b then a::successors(n,tl)
                                                                            else successors(n,tl);
        fun hlist(_, nil) = nil |
                hlist(n, hd::tl) = heuristic(n, hd) :: hlist(n, tl);
        fun tie(nil,nil) = nil |
                tie(h1::t1, h2::t2) = (h1,h2) :: tie(t1,t2);
        val succ = successors(n,edges)
        val cost = hlist(n, succ)
in
        tie(succ,cost)
end;

exception NoSolution;

(∗ASearch takes and initial node, next function and goal node and returns
the optimal path between initial and goal node ∗)

fun AStarSearch(initial, next, goal) = let
        fun putCheapestInFront(hd::tl, nil) = putCheapestInFront(tl,[hd]) |
                putCheapestInFront(nil, x) = x |
                putCheapestInFront((a,b,c,d)::tl1, (xa,xb,xc,xd)::tl2 ) =
                        if c < xc then putCheapestInFront(tl1, (a,b,c,d)::((xa,xb,xc,xd)::tl2))
                        else putCheapestInFront(tl1, ((xa,xb,xc,xd)::tl2)@[(a,b,c,d)]);
        fun addActionsCosts(_,_,nil) = nil |
                addActionsCosts(pcost, pactions, (node, cost)::tl) =
                        ( node, pcost + cost, pcost + cost + heuristic(node, goal), pactions@[node] ) ::
                        addActionsCosts(pcost, pactions, tl);
        fun asearch(nil) = raise NoSolution |
                asearch((node, pathcost, totalcost, actions)::rfringe) =
                        if node = goal then actions
                        else let
                                val expansion = next(node); (∗ next(20) = [(19,17), (21,8), (29,25)] ∗)
                                val newFringeEl = addActionsCosts(pathcost, actions, expansion);
                        in
                                asearch(putCheapestInFront(newFringeEl@rfringe, nil))
                        end
in
        asearch([(initial, 0, heuristic(initial, goal), [])])
end

(∗ The nodes are labeled starting from the upper−left corner of the map to right/down direction ∗)
val result = AStarSearch(1, next, 26);
```

Problem 11.2 (Girls are witches) 10pt

Consider the following logic argument (after Monty Python):

- A witch is a female who burns.
- Things burn - because they're made of wood.
- Wood floats.
- What else floats on water? A duck.
- if something has the same weight as a duck it must float.
- A duck and scales are fetched. A girl and the duck balance perfectly.

Write the given statements as `ProLog` predicates. Check wether a girl is a witch.

Solution:

```
#!/usr/bin/swipl -s

witch(X):-burns(X),female(X).
burns(X):-wooden(X).
wooden(X):-floats(X).
floats(duck).
floats(X):-sameweight(duck,X).
female(girl).
sameweight(duck,girl).

?- witch(girl).
true.
```

Problem 11.3 (Basic `ProLog` Functions) 10pt

Your task is to implement the functions listed below in `ProLog`. Note that many of them are built-in, but we ask you create your own functions.

- a fucntion removing multiple occurrances of elements in a list

  ?− removeDuplicates([1,1,1,1,2,2,3,4,1,2,7],A).
  A = [1, 2, 3, 4, 7].

- a function reversing a list

  ?− myReverse([1,2,3,4,2,5],R).
  R = [5, 2, 4, 3, 2, 1].

- a function outputting all the permutations of the elements in a list

  ?− myPermutations([1,2,3],Z).
  Z = [1, 2, 3] ;
  Z = [2, 1, 3] ;
  Z = [2, 3, 1] ;
  Z = [1, 3, 2] ;
  Z = [3, 1, 2] ;
  Z = [3, 2, 1].

Feel free to implement any helper functions.

Solution:

```
% remove duplicates function
delete(_,[],[]).
delete(X,[X|T],R) :− delete(X,T,R).
delete(X,[H|T],[H|R]) :− not(X=H), delete(X,T,R).
removeDuplicates([],[]).
removeDuplicates([H|T],[H|R]) :− delete(H,T,S), removeDuplicates(S,R).

% reverse function
preReverse([],X,X).
preReverse([X|Y],Z,W) :− preReverse(Y,[X|Z],W).
myReverse(A,R) :− preReverse(A,[],R).

% permute function
takeout(X,[X|T],T).
takeout(X,[H|T1],[H|T2]) :− takeout(X,T1,T2).
myPermutations([],[]).
myPermutations([X|Y],Z) :− myPermutations(Y,W), takeout(X,Z,W).
```

Problem 11.4 (Prolog "MU" Puzzle) 20pt

In one of the first chapters of "Godel, Escher and Bach: An Eternal Golden Braid" the following puzzle is given: In lack of a better name we'll call it the "MU" puzzle. In simplified terms, you start with a string of words, in this case, we get to start with, "MIU" and with four different rules and we want to try to change the string, "MIU", to "MU". Here are the rules:

1. If any string ends in I, you can append U
2. If any string begins with M, you can duplicate the string after M
3. If any string contains III, you can replace the III with U
4. If any string contains UU, you can delete the UU

Beginning with, "MUI", we can use any of these rules, in any order, for as many times as we like to transform "MUI" into "MU". Well, can it be done? Implement it in `ProLog`.

Solution:

```
/* first rule */

/* appends an element to a list */
append([], E, [E]).
append([X|L], E, [X|Y]) :- append(L,E,Y).

/* concatenates two lists */
con2([], E, E).
con2([X|L], E, [X|Y]) :- con2(L,E,Y).

/* concatenates three lists */
con3([], E, L, Q) :- con2(E, L, Q).
con3([X|T], E, L, [X|Q]) :- con3(T,E,L,Q).

reverse([],[]).
reverse([X|R],L):-reverse(R,S),append(S,X,L).

doesItEndWithI(L) :- reverse(L,['I'|_]).

ruleX( T, W) :- doesItEndWithI(T), append(T, 'U', W).

/* second rule */

ruleX( T, W) :- con2(T,T,W).

/* third rule */

ruleX( L, O) :-
  con3(FH, ['I','I','I'], SH, L),
  con3(FH, ['U'], SH, O).

/* fourth rule */
```

```
ruleX( L, O) :-
  con3(FH, ['U','U'], SH, L),
  con2(FH, SH, O).

/* The Rule */

rule(X,M) :- nm(N,M), ruleL(['I'], X, N).

ruleL(['I'],['I'],0).
ruleL(['I'],X,N) :- N>0, N1 is N-1, ruleL(['I'],J,N1), ruleX(J,X).

nm(0, M) :- M>=0.
nm(N, M) :- M>=0, M1 is M-1, nm(N1, M1), N is N1+1.
```

Problem 11.5 (Unicorn fun) 25pt

You have a rectangular map of N rows and M columns. Some of the cells contain lakes. Knowing N, M and the positions of the lakes, you want to place X unicorns around the lakes. Each unicorn has to be around at least one lake, and no two unicorns can be placed next to each other. When talking about 2 items being next to each other, the (potentially) eight neighbors of a cell should be considered.

Write a `ProLog` rule that determines whether a list of unicorn placements is a correct one.

Use the $setof(Things, GoalExpression, Bag)$ predicate which computes all Things which satisfy the GoalExpresssion and collect them in the list Bag. The list will not contain duplicates and it will be sorted. For the input, consider that facts about the rows, columns and lake placements are written down before your program.

Example:

```
rows(3).
cols(3).

lake(1,1).
lake(2,3).
lake(3,2).

?- solution(3,Sol).
Sol = [[unicorn(3, 3), unicorn(2, 1), unicorn(1, 3)],[unicorn(3, 3), unicorn(3, 1), unicorn(1, 2)],[unicorn(3, 3), unicorn(
```

Solution:

```
/*test1

rows(3).
cols(3).

lake(1,1).
lake(2,3).
lake(3,2).

Sol = [[unicorn(3, 3), unicorn(2, 1), unicorn(1, 3)],
       [unicorn(3, 3), unicorn(3, 1), unicorn(1, 2)],
       [unicorn(3, 3), unicorn(3, 1), unicorn(1, 3)]].
*/

/*test2

rows(3).
cols(3).

lake(3,3).

?- solution(1,Sol).
Sol = [[unicorn(2, 2)], [unicorn(2, 3)], [unicorn(3, 2)]].
```

```
*/


/*test3
rows(5).
cols(4).

lake(1,1).
lake(1,3).
lake(1,4).
lake(3,3).
lake(4,2).
lake(5,1).
lake(5,4).

?- solution(5,S).
S = [[unicorn(5, 2), unicorn(4, 4), unicorn(3, 1), unicorn(2, 4), unicorn(1, 2)],
     [unicorn(5, 2), unicorn(4, 4), unicorn(3, 2), unicorn(2, 4), unicorn(1, 2)]].
*/


solution(N,Sol):-setof(X, place_unicorns(N,X), Sol).

% Returns one posible arrangement of given number of unicorns.
place_unicorns(0,[]).
place_unicorns(N,[unicorn(X,Y)|T]):-N>0, M is N-1,
                                    place_unicorns(M,T),
                                    valid_unicorn_position(X,Y,T).

% Returns a valid unicorn position, given a list of already place unicorns.
% Also, the returned position is on the lower, right side w.r.t. the
% previously placed unicorn.
valid_unicorn_position(X,Y,[]):-valid_row(X), valid_col(Y), not(lake(X,Y)),
    has_lake_around(X,Y),
    not(has_unicorn_around(X,Y,[])).
valid_unicorn_position(X,Y,[unicorn(A,B)|T]):-valid_row(X),
    X=A,valid_col(Y),Y>B,
    not(lake(X,Y)),
    has_lake_around(X,Y),
    not(has_unicorn_around(X,Y,[unicorn(A,B)|T])).
valid_unicorn_position(X,Y,[unicorn(A,B)|T]):-valid_row(X),
    X>A,valid_col(Y),
    not(lake(X,Y)),
    has_lake_around(X,Y),
    not(has_unicorn_around(X,Y,[unicorn(A,B)|T])).


%checks if x is a member of the given list.
member(X,[X|_]).
member(X,[_|R]):-member(X,R).

% Checks if X,Y has a unicorn around it.
```

```
has_unicorn_around(X,Y,L):-member(unicorn(X,Y),L).
has_unicorn_around(X,Y,L):- W is X-1, valid_row(W),
                            member(unicorn(W,Y),L).
has_unicorn_around(X,Y,L):- Z is Y-1, valid_col(Z),
                            member(unicorn(X,Z),L).
has_unicorn_around(X,Y,L):- W is X+1, valid_row(W),
                            member(unicorn(W,Y),L).
has_unicorn_around(X,Y,L):- Z is Y+1, valid_col(Z),
                            member(unicorn(X,Z),L).
has_unicorn_around(X,Y,L):- W is X-1, valid_row(W),
                            Z is Y-1, valid_col(Z),
                            member(unicorn(W,Z),L).
has_unicorn_around(X,Y,L):- W is X-1, valid_row(W),
                            Z is Y+1, valid_col(Z),
                            member(unicorn(W,Z),L).
has_unicorn_around(X,Y,L):- W is X+1, valid_row(W),
                            Z is Y-1, valid_col(Z),
                            member(unicorn(W,Z),L).
has_unicorn_around(X,Y,L):- W is X+1, valid_row(W),
                            Z is Y+1, valid_col(Z),
                            member(unicorn(W,Z),L).

% Checks if X,Y has a lake around it.
has_lake_around(X,Y):- W is X-1, valid_row(W), lake(W,Y).
has_lake_around(X,Y):- Z is Y-1, valid_col(Z), lake(X,Z).
has_lake_around(X,Y):- W is X+1, valid_row(W), lake(W,Y).
has_lake_around(X,Y):- Z is Y+1, valid_col(Z), lake(X,Z).
has_lake_around(X,Y):- W is X-1, valid_row(W), Z is Y-1, valid_col(Z), lake(W,Z).
has_lake_around(X,Y):- W is X-1, valid_row(W), Z is Y+1, valid_col(Z), lake(W,Z).
has_lake_around(X,Y):- W is X+1, valid_row(W), Z is Y-1, valid_col(Z), lake(W,Z).
has_lake_around(X,Y):- W is X+1, valid_row(W), Z is Y+1, valid_col(Z), lake(W,Z).

% Check if a row or column is in the valid range.
valid_row(X):-rows(A), between(1, A, X).
valid_col(X):-cols(A), between(1, A, X).
```
