General Computer Science II (320201) Spring 2011

Michael Kohlhase
Jacobs University Bremen
FOR COURSE PURPOSES ONLY

April 8, 2013

Contents

Assignment 1: Semester revision
(Given Feb. 2., Due Feb. 9.)

15pt

Problem 1.1 (ADT for sentences)

A *sentence* is a (possibly empty) list of **non-empty** words over the alphabet $\{a, b, \ldots, z\}$

Create an Abstract Data Type for *sentences*. Start with the constructor declarations $[a\colon Letter], [b\colon Letter], \ldots, [z\colon Letter]$. Make sure that any sentence has a *unique* representation in your ADT.

Don't forget that the words are non-empty, but the sentence can be empty.

Solution:

$\langle\{Letter, Word, Sentence\}, \{[a\colon Letter], [z\colon Letter], [baseW\colon Letter \to Word], [stepW\colon Letter \times Word \to Word],$

Problem 1.2 (Divisibility by 1225) 15pt

Prove or refute that for every natural number $n \geq 1$ it holds that

$$6^{2n} - 35n - 1 \bmod 1225 = 0$$

Solution:

Proof:

P.1 We have two cases

P.1.1 $n = 1$:

P.1.1.1 $6^{2n} - 35n - 1 = 36 - 35 - 1 = 0$ is divisible by 1225. □

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

P.1.2.1 Assume the statement holds for some n.

P.1.2.2 Then,

$$\begin{aligned} 6^{2(n+1)} - 35(n+1) - 1 &= 36 * 36^n - 35n - 35 - 1 \\ &= 36 * 36^n - 35n - 36 \\ &= 36 * (36^n - 1) - 35n \\ &= 36 * (36^n - 35n - 1) - 35n + 36 * 35n \\ &= 36 * (36^n - 35n - 1) + 35 * 35n \\ &= 36 * (36^n - 35n - 1) + 1225n \end{aligned}$$

By induction hypothesis $(36n - 35n - 1)$ is divisible by 1225, which means that $35 * (36n - 35n - 1)$ is also divisible by 1225. $1225n$ is obviously divisible by 1225. Therefore $6^{2(n+1)} - 35(n + 1) - 1$ is divisible by 1225, so we proved the statement for all n. □

□

Problem 1.3: Consider the Hilbert-style calculus given by the following axiom schemata 15pt

1. $(\mathbf{A} \vee \mathbf{A}) \Rightarrow \mathbf{A}$ (idempotence of disjunction)
2. $\mathbf{A} \Rightarrow \mathbf{A} \vee \mathbf{B}$ (weakening)
3. $(\mathbf{A} \vee \mathbf{B}) \Rightarrow \mathbf{B} \vee \mathbf{A}$ (commutativity)
4. $((\mathbf{A} \Rightarrow \mathbf{B})) \Rightarrow (((\mathbf{C} \vee \mathbf{A})) \Rightarrow (\mathbf{C} \vee \mathbf{B}))$

together with the inference rule $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$ MP

In this calculus, the symbols $\Rightarrow$, $\wedge$, and $\Leftrightarrow$ are defined by the following identities:

$$\mathbf{A} \Rightarrow \mathbf{B} = \neg\mathbf{A} \vee \mathbf{B} \qquad \mathbf{A} \wedge \mathbf{B} = \neg(\neg\mathbf{A} \vee \neg\mathbf{B}) \qquad \mathbf{A} \Leftrightarrow \mathbf{B} = (\mathbf{A} \Rightarrow \mathbf{B}) \wedge (\mathbf{B} \Rightarrow \mathbf{A})$$

Prove or refute the formulae

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

Solution: We prove the first formula:

1	$((P \vee P) \Rightarrow P) \Rightarrow \neg P \vee (P \vee P) \neg P \vee P$	axiom scheme 4 with $\mathbf{A} := P \vee P$, $\mathbf{B} := P$, and $\mathbf{C} := \neg P$
2	$(P \vee P) \Rightarrow P$	axiom scheme 1 with $\mathbf{A} = P$
3	$(\neg P \vee (P \vee P)) \Rightarrow \neg P \vee P$	MP with 3 and 2
4	$(\neg P \vee (P \vee P)) \Rightarrow \neg P \vee P$	reverse abbreviation for $\Rightarrow$
5	$P \Rightarrow P \vee P$	axiom scheme 2 with $\mathbf{A} := P$ and $\mathbf{B} := P$.
6	$\neg P \vee P$	MP with 4,5

The second formula is very simple

1	$(((P \Rightarrow Q)) \Rightarrow (\neg R \vee P)) \Rightarrow (\neg R \vee P)$	axiom scheme 4 with $\mathbf{A} := P$, $\mathbf{B} := Q$, and $\mathbf{C} := \neg R$
4	$((P \Rightarrow Q)) \Rightarrow (((R \Rightarrow P)) \Rightarrow (R \Rightarrow P))$	reverse abbreviation for $\Rightarrow$

Problem 1.4 (Lexical ordering) 15pt

Let $A := \{x, :, +, R\}$ and $\prec$ be the ordering relation on A. If $\prec_{\text{lex}}$ is the lexical ordering induced by $\prec$ and

$$RRRR : \prec_{\text{lex}} RR + RRx \prec_{\text{lex}} RR : RR \prec_{\text{lex}} xRRRxR$$

write down one possibility for $\prec$.

Solution:

$R \prec + \prec : \prec x$ OR
$R \prec + \prec x \prec :$ OR
$R \prec x \prec + \prec :$

Assignment 2: Graphs and Circuits
(Given Feb. 9., Due Feb. 16.)

20pt

Problem 2.5 (Tree Equivalences)
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.

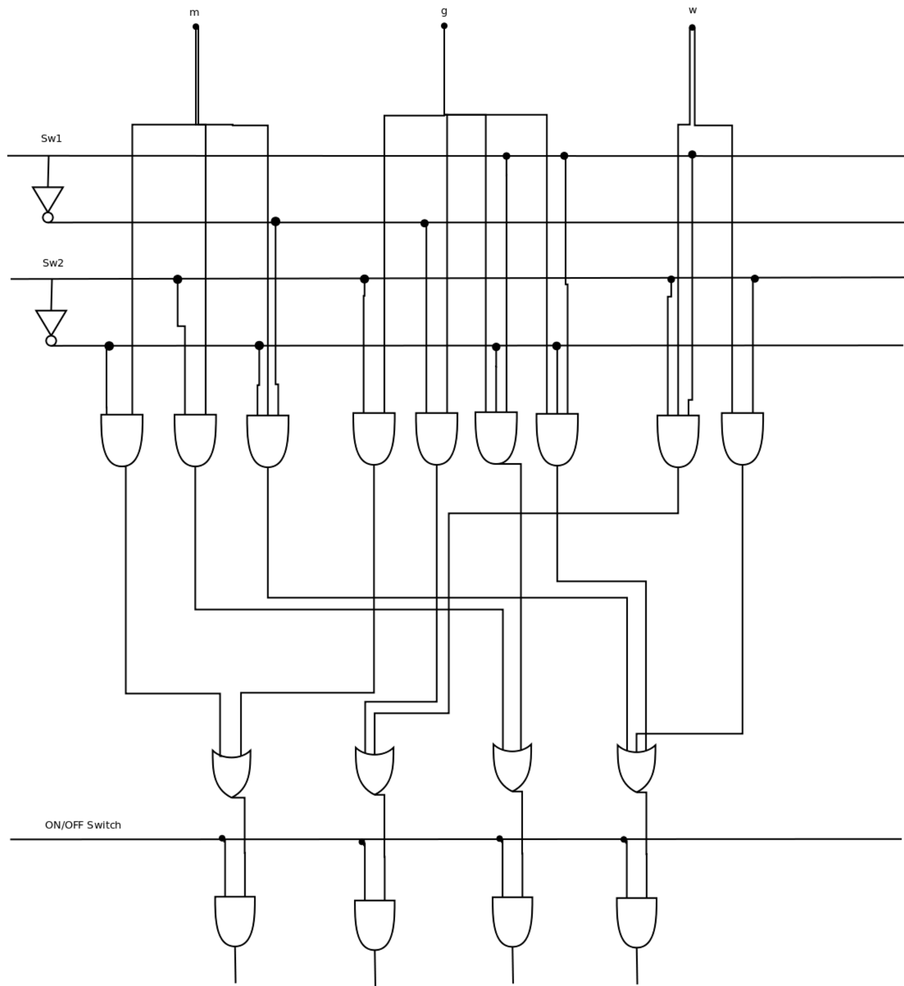
Problem 2.6 (Gnome Contraption) 30pt

Good news! The gnomes from Knoops Park have just released their new machine. It takes the 3 most important gnome resources, namely mushrooms, gold and wine and outputs suitable combinations of them as rations for the day on 4 different channels. The machine has of course an on/off switch and if turned off the output should be 0 on all channels. It also has two other switches to vary the output.

The gnomes are however forgetful creatures and they have misplaced the blueprint. Consider the following function below as a simplified representation of this device (in the case it is turned on). Design a combinational circuit (using only NOT, AND and OR gates) that implements this mixer function. You can use an additional input variable that checks if the mixer is on. $f_{\text{mix}} \colon \mathbb{B}^3 \times \mathbb{B} \times \mathbb{B} \to \mathbb{B}^4$ with

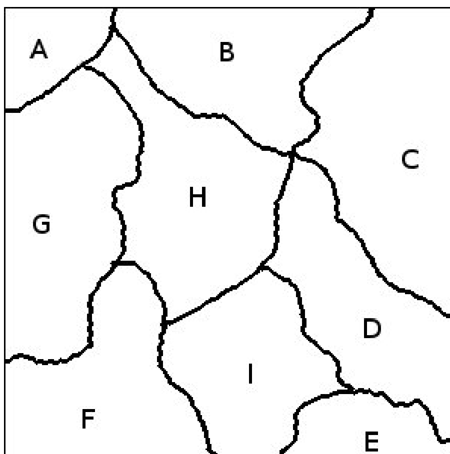
$$f_{\text{mix}}(\langle m, g, w \rangle, sw_1, sw_2) \begin{cases} \langle m, g, 0, m \rangle & \text{if } sw_1 = 0, sw_2 = 0 \\ \langle g, g, m, w \rangle & \text{if } sw_1 = 0, sw_2 = 1 \\ \langle g, w, m, w \rangle & \text{if } sw_1 = 1, sw_2 = 1 \\ \langle m, 0, g, g \rangle & \text{if } sw_1 = 1, sw_2 = 0 \end{cases}$$

Solution:


m
g
w
Sw1
Sw2
ON/OFF Switch

Problem 2.7 (Coloring graphs) 30pt

- You are given the map below. Model it as a graph; find a way to assign a color to each "country" (node) such that every two neighboring countries do not have the same color.



- Estimate the minimum number of colors that you need for the map.
- Such a way of assigning a color to each node is called a "coloring". The minimum number of colors that are needed for coloring a graph G is called "chromatic number" and is noted with $\chi(G)$. If we note the maximum degree of a node in G with δ, then prove that:

$$\chi(G) \leq \delta + 1$$

Solution:

- Any coloring that is valid (no two neighbors are colored the same) is OK.
- There is a (proven) theorem that states that every map can be colored with 4 colors. This map is one of those, so the expected answer is 4.
- Consider that the coloring gives a partition of the graph noted with $V_1, V_2, \dots V_{\chi(G)}$. Consider v to be a node with degree δ and, without loss of generality, that $v \in V_1$ and its neighbors to be included in $V_2, \dots V_{\delta+1}$.

  Consider **any** node w that is not neighbor with v and not included in any V_k, with $k \leq \delta+1$. We can prove by contradiction that w can actually be included in any partition V_k (with $k \leq \delta + 1$):

  Suppose that w cannot be included in any V_k ($k \leq \delta+1$). This is the case only when it has a neighbor in every V_k. Because we considered $\delta+1$ sets of the partition, this would mean that w has degree $\delta + 1 > \delta$, which contradicts the hypothesis that δ is the **maximum** degree.

Because we considered any node w, we can infer that **all** the nodes $w \in V_l$ with $l > \delta + 1$ can be included in some V_k with $k \leq \delta + 1$ (colored with one of the $\delta + 1$ colors needed instead). Therefore, $\chi(G) \leq \delta + 1$.

Problem 2.8 (DNF Circuit with Quine McCluskey) 20pt

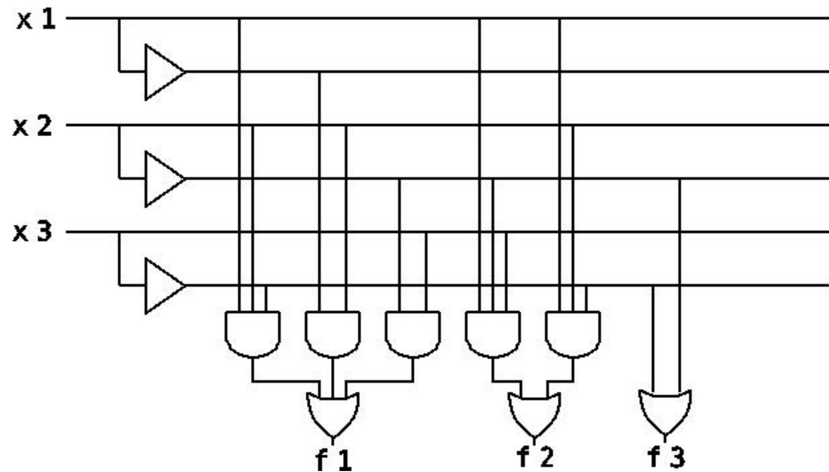
Design a combinational circuit for the following Boolean function:

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

Solution: After applying the QMC we get something like:

$$
\begin{aligned}
f_1(X) &= x_1\, x_2\, \overline{x_3} + \overline{x_1}\, x_2 + \overline{x_2}\, x_3 \\
f_2(X) &= x_1\, \overline{x_2}\, x_3 + x_1\, x_2\, \overline{x_3} \\
f_3(X) &= \overline{x_3} + x_2
\end{aligned}
$$

Hence the circuit:



Assignment 3: Graphs and Circuits

(Given Feb. 16., Due Feb. 23.)

15pt

Problem 3.9 (Cost and depth of adders)

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 3.10 (More on the Carry Chain Adder) 30pt

Given a binary string α let dec(α) be the nonnegative integer it represents when converted to decimal. So for example dec(1101) = 13. Now consider the $(n+1)$-bit carry chain adder studied in class. It takes two binary strings as input, namely $\alpha_n = a_n...a_1a_0$ and $\beta_n = b_n...b_1b_0$ and a binary digit c_0 as inputs and produces an $n+1$ binary string $\sigma_n = s_n...s_1s_0$ and a binary digit c_{n+1} as outputs.

It also satisfies the following specification:

$$dec(\alpha_n) + dec(\beta_n) + c_0 = 2^{n+1} \cdot c_{n+1} + dec(\sigma_n) \tag{1}$$

You are also given two identities regarding the output, namely

$$s_i = a_i \; XOR \; b_i \; XOR \; c_i \tag{2}$$

and

$$c_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i) \tag{3}$$

Your tasks are the following:

- Verify that equations (2) and (3) imply that $a_n + b_n + c_n = 2c_{n+1} + s_n$
- Prove by induction on n or refute that a $(n+1)$-CCA is indeed an $(n+1)$-bit adder, namely that it satisfies (1).

Solution:

- We first consider the fact that $s_n = (a_n \text{ XOR } b_n) \text{ XOR } c_n$, which translates to $(a_n\overline{b_n}+\overline{a_n}b_n)$ XOR c_n. From this the final form will be

  $$s_n = a_n\overline{b_n}\overline{c_n} + \overline{a_n}b_nc_n + \overline{a_n}b_n\overline{c_n} + a_n\overline{b_n}c_n \tag{4}$$

  From this point on there are two approaches. One would be applying associativity and the DeMorgan rule to reach the required result, while the simpler version is to simply test via truth tables. On one side we have the expression $a_n + b_n + c_n$, while on the other we have $s_n = a_n\overline{b_n}\overline{c_n} + \overline{a_n}b_nc_n + \overline{a_n}b_n\overline{c_n} + a_n\overline{b_n}c_n + a_nb_n + b_nc_n + a_nc_n$. The end result will be false when all a_n, b_n, c_n are false and true for every other configuration.

- For the second part, we will use the following very important identity which comes from the definition of binary representation of integers, namely $dec(\alpha_{n+1}) = a_{n+1}2^{n+1} + dec(\alpha_n)$. We shall consider $n = 0$ as base case, in which case we reach the identity $2^0a_0 + 2^0b_0 + c_0 = 2^1c_1 + s_0$. We know the validity of this from the first part of the problem. Now for the step case we naturally assume that the statement holds for n and try to prove it for $n + 1$.

  $dec(\alpha_{n+1}) + dec(\beta_{n+1}) + c_0 = 2^{n+2} \cdot c_{n+2} + dec(\sigma_{n+1})$. Using the identity, this reduces to,

  $$a_{n+1}2^{n+1} + b_{n+1}2^{n+1} + dec(\alpha_n) + dec(\beta_n) + c_0 = 2^{n+2} \cdot c_{n+2} + s_{n+1}2^{n+1} + dec(\sigma_n) \tag{5}$$

  By taking into account the induction hypthoseis, we can reach the form

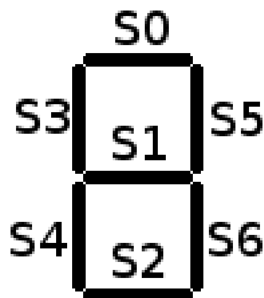
  $$a_{n+1}2^{n+1} + b_{n+1}2^{n+1} + c_{n+1}2^{n+1} = c_{n+2}2^{n+2} + s_{n+1}2^{n+1} \tag{6}$$

  Dividing by 2^{n+1} and noticing the similarity between this and the identity from the first part of the problem we get the final result.

Problem 3.11 (Digit Display) 20pt

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.

Problem 3.12 (TCN in SML) 35pt

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;
```

Assignment 4: Memory and Assembler
(Given Feb. 23., Due Mar. 2.)

25pt

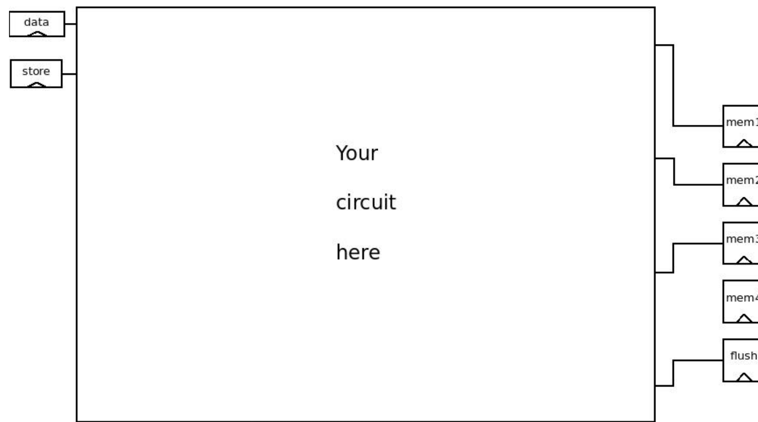
Problem 4.13 (Assembler summation)
You are asked to write an `ASM` program that adds n numbers that are in the memory. You can find the number n in $P(0)$ and the n numbers on positions $P(2..n+1)$. You should use $P(1)$ as the output of the final sum.

Solution:

```
LOAD 0
MOVE ACC IN1
LOADIN 1 1
ADD 1
STORE 1
MOVE IN1 ACC
SUBI 1
MOVE ACC IN1
SUBI 1
JUMP(>=) −7
STOP 0
```

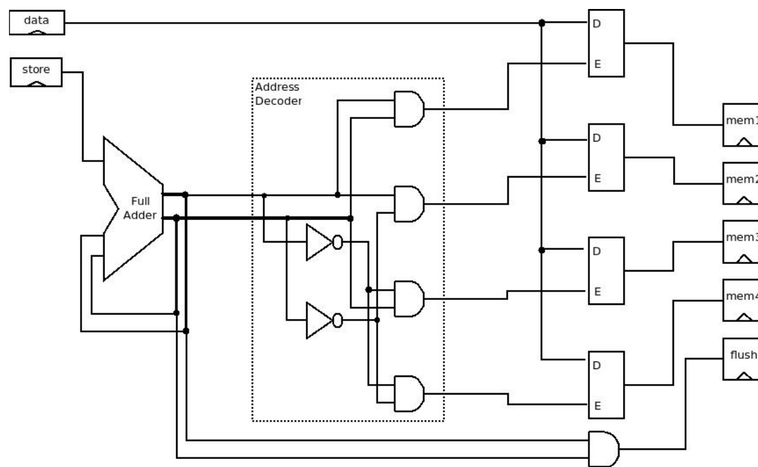
Problem 4.14 (4 Bit Buffer) 25pt

You are required to design a 4-bit buffer. Your circuit will have two inputs: a data input to be stored and a store flag that, when set to 1, will enable storing into the buffer. The four memory units will start as empty; once all four units are filled and new store operation is issued, the contents of the 4 bits is flushed to the output (which will have 4 output lines and a flush line, that will later be used by the main memory). You can use D-latches and the usual binary gates in your design. Briefly describe your circuit.



Note: Do not take timing issues into consideration. If the "flush" line is enabled, the contents of the buffer are first flushed and only then the new data is stored.

Solution: My solution uses an adder that will compute the address where the new data will be stored. Then, a 2-to-4 address decoder will enable the input of one of the 4 D-latches that will actually hold the buffer.



Problem 4.15 (Shift and Duplication on PNS) 15pt

Consider for this problem the signed bit number system and the two's complement number system. Given a binary string $b = a_n \ldots a_0$. We define

1. the duplication function *dupl* that duplicates the leading bit; i.e. it maps the $n+1$-bit number $a_n \ldots a_0$ to the $n+2$-bit number $a_n a_n \ldots a_0$ and
2. the shift function *shift* that maps the $n+1$-bit number $a_n \ldots a_0$ to the $n+2$-bit number $a_n \ldots a_0 0$

Prove or refute the following two statements

- The *shift* function has the same effect in both number systems; i.e. for any integer z:
$$(\langle\langle shift(B(z))\rangle\rangle^{-}) = \langle\langle shift(B_n^{2s}(z))\rangle\rangle_{n+1}^{2s}$$
- The *dupl* function has the same effect in both number systems; i.e. for any integer z:
$$(\langle\langle dupl(B(z))\rangle\rangle^{-}) = \langle\langle dupl(B_n^{2s}(z))\rangle\rangle_{n+1}^{2s}$$

Solution:

- $(\langle\langle dupl(B(z))\rangle\rangle^{-}) = z - 2^{n+1}$ if $z < 0$ else z.
- $(\langle\langle shift(B(z))\rangle\rangle^{-}) = 2 * z$
- $\langle\langle dupl(B_n^{2s}(z))\rangle\rangle_{n+1}^{2s} = z$
- $\langle\langle shift(B_n^{2s}(z))\rangle\rangle_{n+1}^{2s} = 2 * z$

Proof for the last equality:

$$shift(-a_n * 2^n + \sum_{k=0}^{n-1} a_k * 2^k) = -a_n * 2^{n+1} + \sum_{k=0}^{n-1} a_k * 2^{k+1} = 2 * (-a_n * 2^n + \sum_{k=0}^{n-1} a_k * 2^k)$$

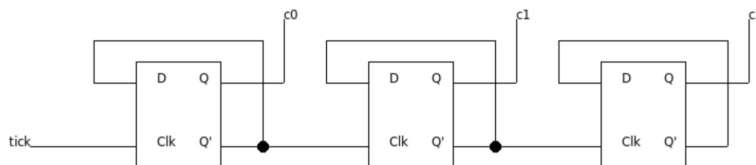
Problem 4.16 (Binary counters) 20pt

In the slides there is an implementation of a D-flipflop with an *enable* input. In practice a different version is more commonly used - the edge-trigerred D-flipflop. Here instead of an *enable* input there is a clock input (*clk*). The difference in operation is that the edge-trigerred D-flipflop only remembers the value of the D input at the one instant when the *clk* input switches from 0 to 1. If *clk* is constantly 0 or constantly 1 the flipflop will not change its state.

Using only such flipflops implement a 3-bit binary counter circuit. The circuit should have only one input 'tick' that will periodically change between 1 and 0. It should have three outputs that count the number of pulses on the input. After the counter counts to 111 it should continue from 000. You can assume the initial state of all flipflops is 0.

Note: Note that a real physical gate has a certain delay. When the input changes it takes some time (nanoseconds) for the output to react. As an upgrade, think of what you could design in order to increase the speed of your counter. There is a simple solution for doubling the efficiency that you might consider implementing.

Solution:



The addition may be performed by replacing the single tick with a frequency multiplier, as can be seen below:

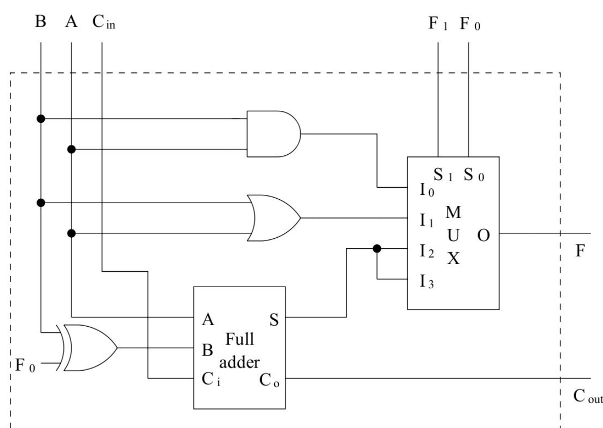
Solution:

Problem 4.17 (Arithmetic and Logic Units) 25pt

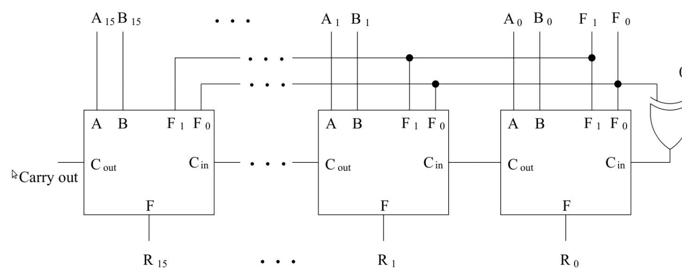
The Arithmetic and Logic Unit forms the computational core of any processor, being able to perform many basic operations. The ones we are interested in are addition, subtraction, AND and OR. Consider two 16-bit inputs A and B in TCN, as well as a carry bit, C_{in}. Your task is to build an ALU that allows us to compute the four operations mentioned above.

Also, since we are considering processors, it is very important that the cost and depth of your circuit are as low as possible.

Solution: The simple 1-bit version is found below:



This can be used in 16 instances in order to create the final solution:



Assignment 5: Memory and Assembler
(Given Mar. 2., Due Mar. 9.)

Problem 5.18 (Simulating REMA in SML) — 30pt

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 5.19 (VM Sum) 25pt

You are given a number n in $\mathcal{S}(0)$. Write a $\mathcal{L}(\mathtt{VM})$ program that will output the result of the following expression in $\mathcal{S}(1)$:

$$\sum_{i=1}^{n+1} i \cdot 2^i$$

Also, consider the following tasks:

- Draw the stack evolution for the first 3 iterations of your implementation.
- Is there a more efficient way of solving the problem than straightforward implementation of the sum? Prove your claim.

Solution: A solution sketch would be the following. We can take n in $\mathcal{S}(0)$ and place 1 in $\mathcal{S}(1)$ for the powers of two, 1 in $\mathcal{S}(2)$ for the iterations on i, 0 in $\mathcal{S}(3)$ for comparison and use $\mathcal{S}(4)$ for the sum initially.

```
 ; initialize
con n
con 1
con 0
con 0
con 0

; Compare 0 to 0th stack component to stop iterations at the right time
peek 0
peek 3
leq

; Here we start iterating
cjp _

;2^i here
peek 1
con 2
mul
poke 1

; i here
peek 2
con 1
add
poke 1

; multiply here
peek 1
peek 2
mul

; place new term in sum
peek 4
add
```

```
poke 4

;decrease n by 1
peek 0
con 1
sub
poke 0

jp −_

; in the end we store the sum in cell 1 as indicated
peek 4
poke 1
halt
```

Regarding the efficiency aspect, the formula can be proved by induction to be

$$\sum_{i=1}^{n+1} i \cdot 2^i = n \cdot 2^{n+2} + 2$$

Setting up this final expression is of course a lot simpler and cleaner.

Problem 5.20 (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 5.21 (Greatest Common Divisor) 25pt

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
```

Assignment 6: Machines and SW

(Given Mar. 9., Due Mar. 16.)

30pt

Problem 6.22 (VM Surprise)

From Tsarist times in Russia a special computation rule with result z and two non-negative integers x and y as input has been passed to future generations. Let x, y be given and z be initalized to 0. The following algorithm holds:

- If $y > 1$, then check if y is an odd number. In this case, add the value of x to the sum z. Irrespective of the parity of y, divide the value of y by 2 (integer division) and double the value of x.
- If $y = 1$, then add x to the sum and end the calculation. The sum z contains the final result.

Your tasks are:

- Explain in your own words what the algorithm computes, also using a concrete example.
- Write down the algorithm in pseudocode.
- Design the $\mathcal{L}$(VM) implementation using all the necessary static procedures.
- Provide a SW program equivalent to the above. (without procedures of course)

Solution:

- The algorithm computes the product of x and y and stores the result in z.
- The pseudocode can be written as:

```
while y>1
  if (y % 2 = 1) z = z + x
  y = y / 2
  x=2 * x
end
z = z + x
write z
```

- First we need two procedures for the operations mod 2 and div 2. These can be given either generally as a mod/div b or directly specialized for this case. One way to do it is:

```
<mod> proc 1 (length)
        arg 1 con 2 leq cjp 8
        con 2 arg 1 sub call <mod>
      arg 1 return

<div> proc 2 (length)
```

```
    arg 1 con 2 leq cjp 13
    arg 2 con 1 add
    con 2 arg 1 sub
    call div
  arg 2 return
```

Using the above we can now proceed to implementing the final function:

```
<mymult> proc 2 (length)
      con 1 arg 2
      leq cjp 5
      arg 1 return
      arg 2 call mod
      cjp 18
      arg 1 arg 2
      call div
      arg 1 con 2 mul
      call mymult add
      return
      arg 2 call div
      arg 1 con 2 mul
      call mymult
    return
```

- Finally here is the SW translation of the simple pseudocode:

```
([("x", con x),("y",con y), ("z", con 0), ("div", con 0), ("mod",con 0)],
  While(Leq(con 2, Var "y"),
    Seq[Assign("mod",Var "y"),
    While(Leq(con 2,Var "mod"),
      Seq[Assign("mod", Sub(Var "mod", con 2))]) ,
      If(Leq(con 0, Var "mod"),
        Assign("z", Add(Var "z",Var"x")),
      Assign("div", Var "div") ),
      Assign("x", Mul(con 2, Var "x")),
      Assign("div", con 0),
      While(Leq(con 1, Var "y"),
        Seq[Assign("div", Add(con 1, Var "div")),
        Assign("y", Sub(Var "y", con 2))]),
      Assign("y", Var "div")]), Var "z");
```

Problem 6.23 (Consecutive numbers) 20pt

You are given three integer numbers n_1, n_2 and n_3 in $\mathcal{S}(0)$, $\mathcal{S}(1)$ and $\mathcal{S}(2)$ respectively. Write a $\mathcal{L}(\mathtt{VM})$ static procedure that will output 1 in $\mathcal{S}(3)$ if the three numbers are consecutive and 0 if not. Note that the numbers need not be in order for the property to hold, so consider a proper handling of shuffled numbers. Also, use $\mathcal{S}(4)$ to output the parity of their product via another static procedure.

Solution: To check if the numbers are consecutive it would be a good idea to initially sort them in either increasing or decreasing order and then devise a procedure using that quality of the number sequence.

```
    peek 1 peek 0 leq cjp 47
    peek 0 peek 2 leq cjp 10
    peek 2 peek 0 peek 1 jp 68
    peek 1 peek 2 leq cjp 10
    peek 0 peek 2 peek 1 jp 53
    peek 2 peek 1 leq cjp 10
    peek 0 peek 1 peek 2 jp 38
    peek 1 peek 2 leq cjp 10
    peek 2 peek 1 peek 0 jp 23
    peek 0 peek 2 leq cjp 10
    peek 1 peek 2 peek 0 jp 8
    peek 1 peek 0 peek 2
//With everything sorted now we can write the procedure:
<cons> proc 3 55
    arg 2, arg 1, sub, con 1, leq,
    con 1, arg 2, arg 1, sub, leq,
    arg 3, arg 2, sub, con 1, leq,
    con 1, arg 3, arg 2, sub, leq,
    add, add, add, con 4, leq,
 return
    cjp 18, con 1, poke 3, jp 5,
    con 0, poke 3
```

Checking the parity is straightforward by multiplying the three numbers and then using a mod 2 procedure to determine the result

```
  <par> proc 1 19
        arg 1 con 2 leq
        cjp 10
          con 2 arg 1 sub
          call 0
        arg 1 return

    peek 0 peek 1 peek 2 mul mul
    call par
```

Problem 6.24 (Decimal to binary in Simple While) 20pt

Write a Simple While Program in Concrete Syntax that takes a decimal number N and computes the binary representation of it. This should be an *int*, and each digit should be either 0 or 1. Then provide the Abstract Syntax for your code. Disregard the possibility of overflowing, in case the int gets too big. For example for an input $varN := 123456$ the output should be an *int* equal to 11110001001000000.

Solution: First the concrete syntax:

```
var n := N; var pow := 1; bin := 0;
var mod := 0; var div := 0;
while 1 <= n do
        div = 0; mod = n;
        while 2 <= mod do
                div = div + 1;
                mod = mod − 2;
        end
        bin = pow ∗ mod + bin;
        pow = pow ∗ 10;
        n = div;
end
return bin;
```

And **then** the abstract syntax

```
([ ("n", Con N), ("pow", Con 1), ("bin", Con 0),
   ("mod", Con 0), ("div", Con 0) ],
While (Leq(Con 1, Var"n"),
       Seq [Assign("div", Con 0), Assign("mod", Var"n"),
            While (Leq(Con 2, Var"mod"),
                   Seq [Assign("div", Add(Var"div", Con 1)),
                        Assign("mod", Sub(Var"mod", Con 2))]
            ),
            Assign("bin", Add(Mul(Var"pow", Var"mod"), Var"bin")),
            Assing("pow", Mul(Var"pow", Con 10)),
            Assign("n", Var"div")]
),
Var"bin")
```

Problem 6.25 (Simple div in SW) 10pt

Write a SW program using Concrete Syntax, respective Abstract Syntax, which takes two decimal numbers a and b and computes the value of $a\mathrm{div}b$ (where div represents the decimal division). For example, for $a = 512$ and $b = 5$, the output should be 102.

Solution: Concrete syntax:

```
var a:=512; var b:=5; var div:=0;
while ( a>=b ) do
    a := a−b;
    div := div+1;
end;
return div;
```

Abstract syntax:

```
([("a", Con 512), ("b", Con 5), ("div", Con 0)],
While( Leq(Var "b", Var "a"),
    Seq [
        Assign(Var "a", Sub(Var "a", Var "b")),
        Assign(Var "div", Add(Var "div", Con 1))
    ]
),
Var "div")
```

Problem 6.26 (SW simulator in SML) 20pt

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));
```

Assignment 7: Turing Machines
(Given Mar. 16., Due Mar. 23.)

Problem 7.27 (SML Implementation of a Turing Machine Simulator) 30pt

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;

Problem 7.28 (Separation TM) 20pt

Design a Turing Machine that accepts an input of the form $w \in \{0, 1, *, +\}^*$. Its purpose is to relocate all $*$ on the left side of the tape and all $+$ on the right side of the tape. Initially on the tape we have randomly scattered elements of our alphabet, for example $010 + + * * +01 + * * + + * 1$. In this case the tape should be $* * *010011 + + + + ++$ at the end of the execution. There is no restriction on how you handle non-+ and non-$*$ characters in your alphabet.

Solution: The solution below basically follows the following algorithm. We start off from the leftmost non-void tape entry and go right until we reach the end. Then we start searching for $*$ and taking them one step at the time to the right of the tape. After all stars are gathered we move back to the left of the tape and follow the same procedure for +, but this time bringing it to the left side of the tape. Feel free to use `http://ironphoenix.org/tril/tm/` in order to test the implementation.

```
1,1 1,1, >
1,0 1,0, >
1,* 1,*, >
1,+ 1, +, >
1, _ 2,_, <

2, 1 2, 1, <
2, 0 2, 0, <
2, + 2, +, <
2, * 3, *, >
2, _ 7, _, >

3, * 4, *, <
3, _ 4, _, <
3,1 5, *, <
3,0 6, *, <
3,+ 12, *, <

4, * 2, *, <
5, * 2, 0, >
6, * 2, 1, >
12, * 2, +, >

7, 1 7, 1, >
7, 0 7, 0, >
7, * H, *
7, + 8, +, <
7, _ H, _

8, + 9, +, >
8, _ 9, _, >
8, 1 10, +, >
8, 0, 11, +, >
9, + 7, +, >
10, +, 7, 0, <
11, +, 7, 1, <
```

Problem 7.29 (Palindrome Detector Turing Machine) 20pt

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, 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 7.30 (Halting problem) 30pt

Show that if a Turing Machine M can decide in general if another Turing machine N halts on an empty input then we can use M and construct a Turing Machine machine that solves the halting problem.

Solution: From any turing machine A and an input for it I we can construct a turing machine B, that first prints I on the input tape and then simulates A on I. We can then use M to decide if B will halt or not and thus solve the halting problem.

Assignment 8: Problem Formulation and Graph Search

(Given Mar. 23., Due Mar. 30.)

20pt

Problem 8.31 (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 8.32 (Relations between search strategies) 20pt

Prove or refute each of the following statements:

1. Breadth-first search is a special case of uniform-cost search.
2. Breadth-first search, depth-first search, and uniform-cost search are special cases of best first searches.

Solution:

1. Let's consider a UCS on a search tree where the cost of each action is the same. Then the cost of a node will be proportional to its depth (distance from initial node). Therefore, the smallest-cost nodes in that way will actually be the shallowest nodes. So our UCS will expand first the shallowest nodes since they are cheaper. This is exactly the defining property of BFS, so our UCS will be equivalent to it, proving the required assertion.
2. Best first search with evaluation function $h(n)$ = "distance from n to the initial node" is indeed BFS since shallowest nodes will be the most desirable.

   Best first search with evaluation function $h(n) = -$"distance from the initial node" is indeed DFS since the deapest leaves will have the most negative (the smallest) $h(n)$.

   Best first search with evaluation function $h(n) =$"the cost of the path from n to the initial node" is indeed UCS since the cheapest nodes will be the most desirable.

   Note that we assume h to be a function from teh set of nodes to the integers, where a smaller value means greater desirability for exapansion of a certain node.

Problem 8.33 (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 8.34 (The Greedy Palindrome) 30pt

A word w is called **palindrome** when it is identical to its reversed representation ($w = reversed(w)$). For example, "abba" is a palindrome, while "abaa" is not.

You are requested to write an SML function that, given a string, will return the palindrome of **minimal length** that is obtained only by adding some (possibly zero) letters at the end of the word. The function should have the following signature:

```
solve = fn : string -> string
```

Examples:

```
- solve("abcxyzy");
val it = "abcxyzyxcba" : string
- solve("abba");
val it = "abba" : string
```

Note: Write a comprehensive comment where you explain why your algorithm is always correct!

Solution: The problem always has a solution because there is always a palindrome at the end of the argument string (for example, the last character is itself a palindrome). We can search for the longest palindrome that ends on the last character of the word, and then add at the end of the word the reversed preffix that ends before the palindrome. More clearly, if $w = w_1w_2 \dots w_n$, then we find the smallest i such that $w_i \dots w_n$ is palindrome and return the result $w1 \dots w_nw_{i-1}w_{i-2} \dots w_1$.

The solution is always correct because, by finding the maximum-length palindrome, the number of inserted characters is minimum. The lower bound of the result is the length of the input itself.

```
(* palindrome(x) checks wether x is a palindrome *)
fun palindrome([]) = true
  | palindrome([a]) = true
  | palindrome(a::lst) =
        let val b::rest = rev(lst) in
                if a=b then
                        palindrome(rest) (* could reverse again but doesn't matter *)
                else
                        false
        end;


(* y=current list, x=preffix already processed (in reversed order) *)
fun do_it(y, x) =
        if palindrome(y) then
                rev(x) @ (y) @ x
        else
                let val a::rest = y in
                        do_it(rest, a::x)
                end;

(* wrapper *)
fun solve(x) = implode( do_it(explode(x), []) );
```

Solution: Sample test cases (any string would do anyway):

```
- solve("abba");
val it = "abba" : string
- solve("abbab");
val it = "abbabba" : string
- solve("abcxyzy");
val it = "abcxyzyxcba" : string
- solve("mama");
val it = "mamam" : string
- solve("llla");
val it = "lllalll" : string
```

Assignment 9: Informed Search Algorithms
(Given Apr. 6., Due Apr. 13.)

20pt

Problem 9.35 (Monotone heuristics)
Let $c(n, a, n')$ be the cost for a step from node n to a successor node n' for an action a. A heuristic h is called *monotone* if $h(n) \leq h(n') + c(n, a, n')$. Prove or refute that if a heuristic is monotone, it must be admissible. Construct a search problem and a heuristic that is admissible but not monotone. Note: For the goal node g it holds $h(g) = 0$. Moreover we require that the goal must be reachable and that $h(n) \geq 0$.

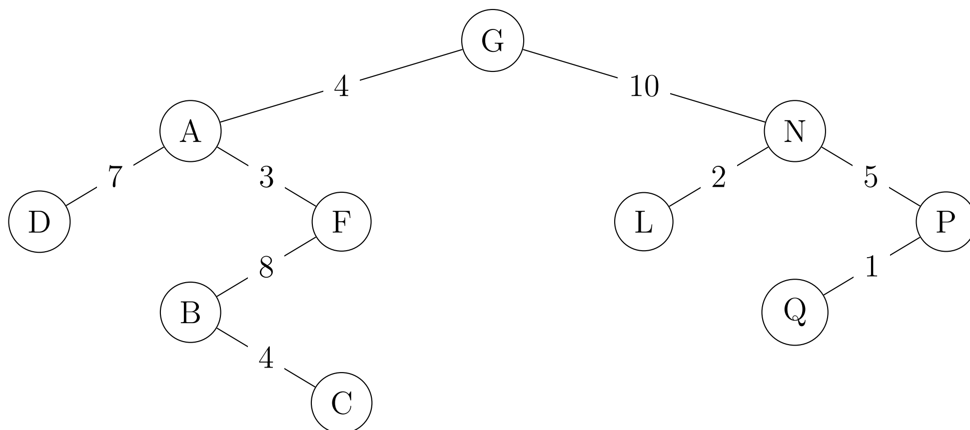
Solution: For the heuristic h to be admissible we have to show that $h(x)$ is less or equal the minimum coast to a goal state.

Let n_1 any node different from the goal node g. Suppose $< n_1, n_2, \ldots, n_p, g >$ is the minimum cost path from n_1 to g. Its cost is $C = c(n_1, a_1, n_2) + c(n_2, a_2, n_3) \ldots + c(n_p, a_p, g)$. Using $h(n) - h(n') \leq c(n, a, n')$ we get $C \geq h(n_1) - h(n_2) + h(n_2) - h(n_1) + \ldots + h(n_p) - h(g) = h(n_1) - h(g) = h(n_1)$. Hence we have proven that $h(n_1)$ is admissible.

We consider the minimum distance search problem with three cities A, B, G where G is the goal city and the distances are $dist(A, B) = 2$ and $dist(B, G) = 100$. The heuristic $h(A) = 6, h(B) = 3, h(G) = 0$ is admissble since $h(A) < dist(A, B) + dist(B, G)$. But is is not monotone since $h(A) > h(B) + dist(A, B)$.

Problem 9.36 (BFS vs. UCS) 15pt

- Compare and contrast BFS and UCS (i.e. highlight the differences and the common concepts.
- Apply BFS and UCS on the following tree:



Solution:

- BFS expands the shallowest unexpanded node in FIFO order, is complete but not optimal unless all the costs are 1. UCS expands the least-cost unexpanded node, thus the fringe is ordered increasingly by path cost; it is complete, and optimal.
- BFS: G, A, N, D, F, L, P, B, Q, C

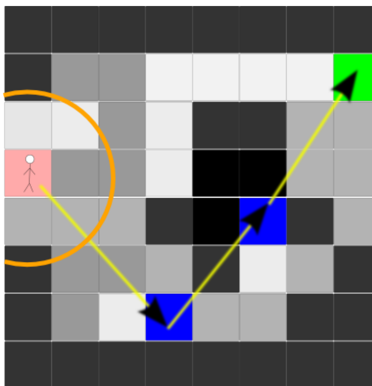
  UCS: G, A, F, N, D, L, B, P, Q, C (but E and P might be swapped)

Problem 9.37 (Let's play games!) 25pt

You are playing an RPG game where you have to control a character on a map. The map is internally represented as a matrix D of integer numbers between 0 and 10; the number represents how difficult is for your character to pass (therefore, if $D_{i,j} = 0$ then your character will pass really fast, while if $D_{i,j} = 10$ your character might not pass at all). Consider that the game assigns to your character a certain visibility radius r (that is, you cannot "see" further away than r units). You now want to move your character from one cell of the map to another.

- The programmer of the game decided to use A* for finding the shortest path (time-wise) between the two points. Design a heuristic that would work for the given problem.
- While playing the game, you observe that it consumes a lot of memory and is very slow while your character is moving. How would you improve the A* algorithm or the heuristic it uses in order to become more efficient?
- Consider that the programmer chose a really bad heuristic for the pathfinding algorithm, which takes your character through slow terrain. However, you are a skilled player and you already know the map by heart. You decide to use the "waypoint" feature of the game (defining a set of points the character has to reach in the order you input them in order to optimize the path). Design a heuristic function that would make use of the waypoints you set and/or of the lines connecting them.

For example, in the picture below, we encoded the difficulty of the terain in grayscale (white is equivalent to 0, up to black, which is equivalent to 10). The starting position is marked in red, the goal position in green. You already have set two waypoints marked in blue, which are connected by three lines in yellow. Your visible area is delimited by the orange circle around your starting position.



Note: The problem does not have a fixed solution, but rather is meant to make you think of varieties of A* and different heuristics. You are expected to give a rigurous description of your **assumptions** while solving it. You are not required to give precise formulas for the heuristics, but remember that they can make your solution more clear.

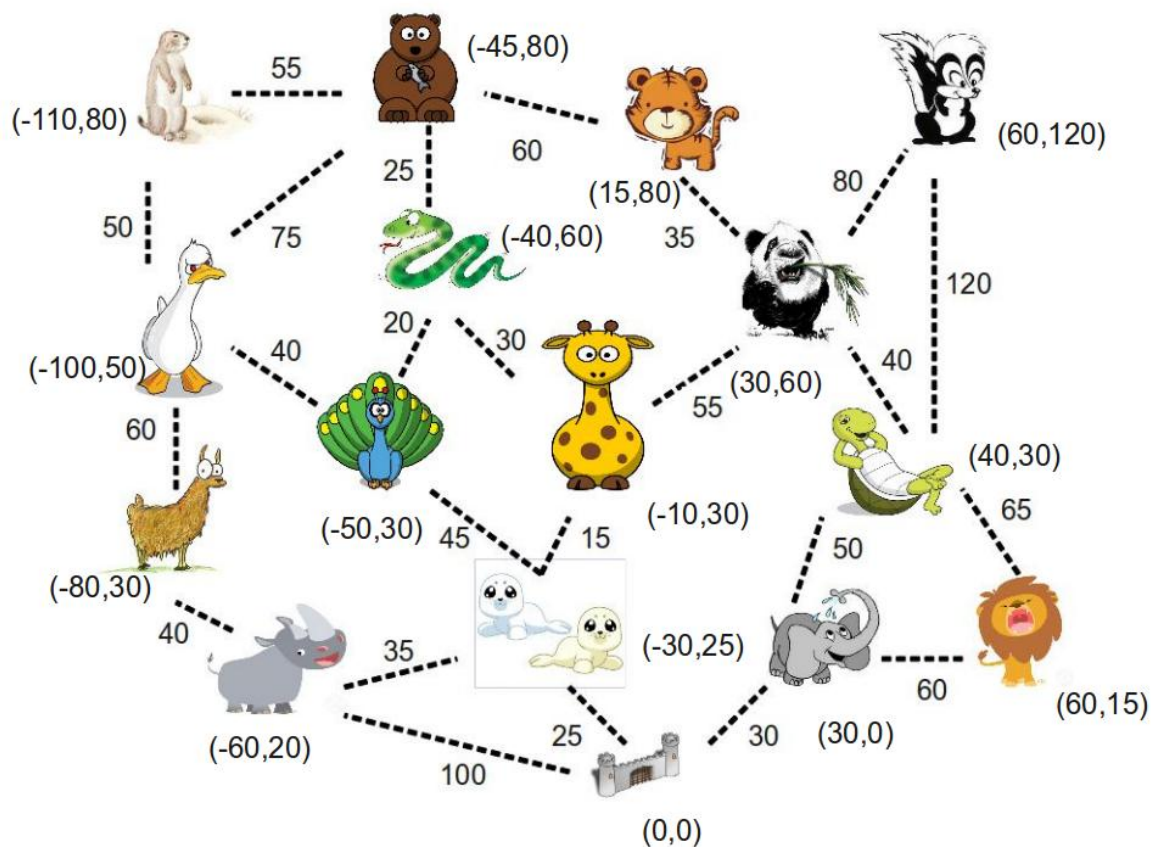
Solution:

Problem 9.38 (*A^* search on Hamburg Zoo*) 40pt

Implement the A^* search algorithm in SML and test it on the following problem. You and your friends are visiting Hamburg Zoo. Not to get lost, you set up a few meeting points. You all start from the main gate, and the first area you get together is near the skunk. Next you decide to meet near the duck, while the last checkpoint is of course the main gate before leaving.

You may consider linear distance as heuristic. The lengths 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 rally point and the associated cost. Since we have three different rally points, you may write the output for each of them in comments or document it in a similar manner.



Solution:

The solution belongs to student Daniel Cazacu and represents a nicely-commented straightforward manner of approaching the problem, with clean results.

```
(*First of all, we define the custom datatype that we will use for the vertices :*)

datatype vertex=
  meerkat
| bear
| duck
| cat
| snake
| panda
| giraffe
| llama
| skunk
| peacock
| turtle
| lion
| rhino
| seals
| elephant
| gate;

(*-----------------------------------------------------------------------------------------------

(*In the following we set the coordinates of each vertex :*)

fun coordinates(meerkat)=(~110.0,80.0)
| coordinates(bear)=(~45.0,80.0)
| coordinates(duck)=(~100.0,50.0)
| coordinates(snake)=(~40.0,60.0)
| coordinates(cat)=(15.0,80.0)
| coordinates(peacock)=(~50.0,30.0)
| coordinates(llama)=(~80.0,30.0)
| coordinates(giraffe)=(~10.0,30.0)
| coordinates(panda)=(30.0,60.0)
| coordinates(skunk)=(60.0,120.0)
| coordinates(turtle)=(40.0,30.0)
| coordinates(rhino)=(~60.0,20.0)
| coordinates(seals)=(~30.0,25.0)
| coordinates(elephant)=(30.0,0.0)
| coordinates(lion)=(60.0,15.0)
| coordinates(gate)=(0.0,0.0);

(*-----------------------------------------------------------------------------------------------

(*Now we define the set of adjacent vertices for each of the vertices in the graph :*)

fun adjacent_vertices(meerkat)=[(bear,65.0),(duck,31.6)]
| adjacent_vertices(bear)=[(meerkat,65.0),(duck,62.6),(snake,20.6),(cat,60.0)]
| adjacent_vertices(duck)=[(llama,28.3),(peacock,53.8),(meerkat,31.6),(bear,62.6)]
| adjacent_vertices(snake)=[(peacock,31.6),(giraffe,42.3),(bear,20.6)]
| adjacent_vertices(cat)=[(bear,60.0),(panda,25.0)]
| adjacent_vertices(peacock)=[(duck,53.8),(snake,31.6),(seals,20.6)]
| adjacent_vertices(llama)=[(duck,28.3),(rhino,22.4)]
| adjacent_vertices(giraffe)=[(snake,42.3),(seals,20.6),(panda,50.0)]
| adjacent_vertices(panda)=[(cat,25.0),(giraffe,50.0),(skunk,67.1),(turtle,31.6)]
| adjacent_vertices(skunk)=[(panda,67.1),(turtle,91.2)]
| adjacent_vertices(turtle)=[(panda,31.6),(skunk,91.2),(lion,25.0),(elephant,31.6)]
| adjacent_vertices(rhino)=[(llama,22.4),(seals,30.4),(gate,63.2)]
| adjacent_vertices(seals)=[(rhino,30.4),(peacock,20.6),(giraffe,20.6),(gate,39.0)]
| adjacent_vertices(elephant)=[(lion,33.5),(gate,30.0),(turtle,31.6)]
| adjacent_vertices(lion)=[(elephant,33.5),(turtle,25.0)]
| adjacent_vertices(gate)=[(elephant,30.0),(seals,39.0),(rhino,63.2)];

(*-----------------------------------------------------------------------------------------------

(*For the following,we will need the math library :*)

open Math;

(*Now that we have that, we create the heuristic function, which in our case is the straight-line distance*)

fun SLD(a:vertex,b:vertex)= let val (x1,y1)=coordinates(a)
                            in let val (x2,y2)=coordinates(b)
                               in sqrt(pow((x2-x1),2.0)+pow((y2-y1),2.0)) end end;

(*-----------------------------------------------------------------------------------------------

(*Now we create a function that computes the value of f(n), by which we expand in the A* algorithm*)

fun function_computer(initial,goal,nil)=nil
| function_computer(initial,goal,(vertex,cost)::tail)= (vertex,(SLD(goal,vertex)+cost))::function_computer(initial,goal,tail);

(*-----------------------------------------------------------------------------------------------

(*This function returns the adjacent vertices of a node, with respect to the goal, together with their f-value*)

fun adjacent_vertices_plus_f(initial,goal)= function_computer(initial,goal,adjacent_vertices(initial));

(*-----------------------------------------------------------------------------------------------

(*This function selects the adjacent node to the initial node that has the lowest f-value(most desirable) *)

fun select_lowest_cost_node((vertex,cost:real)::nil)= (vertex,cost)
| select_lowest_cost_node((vertex_1,cost_1)::tail)= let val (vertex_2,cost_2)=select_lowest_cost_node(tail)
```

```
                                                in
                                                    if (cost_1<cost_2) then
                                                           (vertex_1,cost_1)
                                                      else
                                                        (vertex_2,cost_2) end;

(*-----------------------------------------------------------------------------------------------

(*This function selects that respective node (see above comment) and outputs it*)

fun next_vertex(initial,goal)= let val (vertex,cst)=select_lowest_cost_node(adjacent_vertices_plus_f(initial,goal))
                              in vertex

                              end;

(*-----------------------------------------------------------------------------------------------

(*A preliminary function, that returns only the path*)

fun path_of_search(initial,goal)=if (next_vertex(initial,goal)=goal)
                                   then
                                     initial::goal::nil
                                     else
                                     initial::path_of_search(next_vertex(initial,goal),goal);

(*-----------------------------------------------------------------------------------------------

(*Now a function that computes the cost of a path*)

fun path_cost([t])=0.0
| path_cost(h::t::rest)=SLD(h,t)+path_cost(t::rest);

(*-----------------------------------------------------------------------------------------------

(*And now the final function,that includes both the path and its cost*)

fun final_astar_search(a,b)=(path_of_search(a,b),path_cost(path_of_search(a,b)));

(*-----------------------------------------------------------------------------------------------

(*Here we apply the algorithm to get the required paths :*)

final_astar_search(gate,skunk);
final_astar_search(skunk,duck);
final_astar_search(duck,gate);
```

Problem 9.39 (Bonus search on Hamburg Zoo) 40pt

As diligent CS students, even when visiting a zoo, one must consider an optimal path for checking out each objective. Consider the case when you are in a hurry to reach a gaming tournament back in Bremen for example, but you still wish to see as much as possible at the zoo.

Implement a search algorithm in SML which you think will produce the best result on this problem and explain your choice. The goal is, starting from the main gate, to visit all the animals and then return to the main gate again for the trip home.

You have no signature restrictions, however the output must clearly present the chosen path and the cost of the solution. The solution must be outputed directly from your code in a file named 'Zoo.txt'. Also, you should spend some time on discussing why you chose your particular algorithm over the others.

Solution:

The solution is provided thanks to Jan Wilken Doerrie.

```
(* Declarations of Datatypes and types*)
datatype Animal = Gate | Elephant | Lion | Turtle | Skunk | Panda | Tiger |
Beaver | Snake | Giraffe | Peacock | Meerkat | Duck | Lama | Rhino | Seals;

type Coords = real*real;
type Cost = real;

(* Coordinates function, takes animal, returns Coordinates *)
fun get_coords(Gate) = (0.0,0.0):Coords
  | get_coords(Elephant) = (30.0,0.0)
  | get_coords(Lion) = (60.0,15.0)
  | get_coords(Turtle) = (40.0,30.0)
  | get_coords(Skunk) = (60.0,120.0)
  | get_coords(Panda) = (30.0,60.0)
  | get_coords(Tiger) = (15.0,80.0)
  | get_coords(Beaver) = (~45.0,80.0)
  | get_coords(Snake) = (~40.0,60.0)
  | get_coords(Giraffe) = (~10.0,30.0)
  | get_coords(Peacock) = (~50.0,30.0)
  | get_coords(Meerkat) = (~110.0,80.0)
  | get_coords(Duck) = (~100.0,50.0)
  | get_coords(Lama) = (~80.0,30.0)
  | get_coords(Rhino) = (~60.0,20.0)
  | get_coords(Seals) = (~30.0,25.0);

(* path function, takes animal and returns list of animals which have a direct path to the initial animal *)
fun path(Gate) = [Rhino,Seals,Elephant]
  | path(Elephant) = [Gate,Turtle,Lion]
  | path(Lion) = [Elephant,Turtle]
  | path(Turtle) = [Elephant,Lion,Panda,Skunk]
  | path(Skunk) = [Panda,Turtle]
  | path(Panda) = [Tiger,Giraffe,Turtle,Skunk]
  | path(Tiger) = [Panda,Beaver]
  | path(Beaver) = [Tiger,Snake,Duck,Meerkat]
  | path(Snake) = [Beaver,Giraffe,Peacock]
  | path(Giraffe) = [Snake,Panda,Seals]
  | path(Peacock) = [Snake,Panda,Duck]
  | path(Meerkat) = [Duck,Beaver]
  | path(Duck) = [Meerkat,Beaver,Peacock,Lama]
  | path(Lama) = [Duck,Rhino]
  | path(Rhino) = [Lama,Seals,Gate]
  | path(Seals) = [Peacock,Giraffe,Rhino,Gate];

(* list of all animals *)
val animals = [Gate, Elephant, Lion, Turtle, Skunk, Panda, Tiger,
Beaver, Snake, Giraffe, Peacock, Meerkat, Duck, Lama, Rhino, Seals];

(* straight line distance function *)
fun SLD (((a,b):Coords),((c,d):Coords)) = Math.sqrt((a-c)*(a-c)+(b-d)*(b-d)):Cost;

(* more sophisticated path function, not only takes animal, but also cost and the current route,
   returns list with next animals with updated cost and route *)
fun path2((Gate,C,rt)) = [(Rhino,C+SLD(get_coords(Gate),get_coords(Rhino)),Gate::rt),(Seals,C+SLD(get_coords(Gate),get_coords(Seals)),Gate::rt),(Elephant,C+SLD
  | path2((Elephant,C,rt)) = [(Gate,C+SLD(get_coords(Gate),get_coords(Elephant)),Elephant::rt),(Turtle,C+SLD(get_coords(Turtle),get_coords(Elephant)),Elephant::r
  | path2((Lion,C,rt)) = [(Elephant,C+SLD(get_coords(Elephant),get_coords(Lion)),Lion::rt),(Turtle,C+SLD(get_coords(Turtle),get_coords(Lion)),Lion::rt)]
  | path2((Turtle,C,rt)) = [(Elephant,C+SLD(get_coords(Elephant),get_coords(Turtle)),Turtle::rt),(Lion,C+SLD(get_coords(Lion),get_coords(Turtle)),Turtle::rt),(Panda
```

```
  | path2((Skunk,C,rt)) = [(Panda,C+SLD(get_coords(Panda),get_coords(Skunk)),Skunk::rt),(Turtle,C+SLD(get_coords(Turtle),get_coords(Skunk)),Skunk::rt)]
  | path2((Panda,C,rt)) = [(Tiger,C+SLD(get_coords(Tiger),get_coords(Panda)),Panda::rt),(Giraffe,C+SLD(get_coords(Giraffe),get_coords(Panda)),Panda::rt),(Turtle,C
  | path2((Tiger,C,rt)) = [(Panda,C+SLD(get_coords(Panda),get_coords(Tiger)),Tiger::rt),(Beaver,C+SLD(get_coords(Beaver),get_coords(Tiger)),Tiger::rt)]
  | path2((Beaver,C,rt)) = [(Tiger,C+SLD(get_coords(Tiger),get_coords(Beaver)),Beaver::rt),(Snake,C+SLD(get_coords(Snake),get_coords(Beaver)),Beaver::rt),(Duck,C
  | path2((Snake,C,rt)) = [(Beaver,C+SLD(get_coords(Beaver),get_coords(Snake)),Snake::rt),(Giraffe,C+SLD(get_coords(Giraffe),get_coords(Snake)),Snake::rt),(Peacock
  | path2((Giraffe,C,rt)) = [(Snake,C+SLD(get_coords(Snake),get_coords(Giraffe)),Giraffe::rt),(Panda,C+SLD(get_coords(Panda),get_coords(Giraffe)),Giraffe::rt),(Seals
  | path2((Peacock,C,rt)) = [(Snake,C+SLD(get_coords(Snake),get_coords(Peacock)),Peacock::rt),(Panda,C+SLD(get_coords(Panda),get_coords(Peacock)),Peacock::rt),
  | path2((Meerkat,C,rt)) = [(Duck,C+SLD(get_coords(Duck),get_coords(Meerkat)),Meerkat::rt),(Beaver,C+SLD(get_coords(Beaver),get_coords(Meerkat)),Meerkat::rt)]
  | path2((Duck,C,rt)) = [(Meerkat,C+SLD(get_coords(Meerkat),get_coords(Duck)),Duck::rt),(Beaver,C+SLD(get_coords(Beaver),get_coords(Duck)),Duck::rt),(Peacock,
  | path2((Lama,C,rt)) = [(Duck,C+SLD(get_coords(Duck),get_coords(Lama)),Lama::rt),(Rhino,C+SLD(get_coords(Rhino),get_coords(Lama)),Lama::rt)]
  | path2((Rhino,C,rt)) = [(Gate,C+SLD(get_coords(Gate),get_coords(Rhino)),Rhino::rt),(Seals,C+SLD(get_coords(Seals),get_coords(Rhino)),Rhino::rt),(Lama,C+SLD
  | path2((Seals,C,rt)) = [(Giraffe,C+SLD(get_coords(Giraffe),get_coords(Seals)),Seals::rt),(Rhino,C+SLD(get_coords(Rhino),get_coords(Seals)),Seals::rt),(Peacock,C+S

(* minimal function, calculates f(n) = g(n) + h(n), returns animal with lowest f, parameter D is destination, needed for h(n) *)
fun min((a,b,C)::nil,D) = (a,b,C)
  | min((a,b,C)::((c,d,E)::ls),D) = if (SLD(get_coords(a),get_coords(D))) + b < (SLD(get_coords(c),get_coords(D))) + d
                                    then min((a,b,C)::ls,D) else min((c,d,E)::ls,D);

(* drops a specfic animal *)
fun drop ((l,k,m)::ls,(a,b,c)) = if l = a then ls else (l,k,m)::drop(ls,(a,b,c));

(* sort function, sorts by cost (algorithm is certainly not optimal, but easy to understand) *)
fun sort(nil,D) = nil
  | sort(L,D) = min(L,D)::sort(drop(L,min(L,D)),D);

(* A* help function, expands animal with lowest f(n), sorts by cost again. stops if destination is reached, returns cost and path *)
fun astar_help(L,D) = let val (curr,cost,path)::ls = L
                      in if curr = D then (cost,rev(D::path))
                                     else astar_help(sort(path2(hd(L))@ls,D),D)
                      end;

(* main nA* function, takes to animals and returns shortest path + cost *)
fun astar S D = astar_help([(S,0.0,nil)],D);
(* some astar outputs:
astar Gate Skunk;
val it = (153.818221175,[Gate,Elephant,Turtle,Skunk]) : Cost * Animal list
astar Skunk Duck;
val it = (212.164743653,[Skunk,Panda,Giraffe,Seals,Peacock,Duck])
  : Cost * Animal list
astar Duck Gate;
val it = (113.890504226,[Duck,Lama,Rhino,Gate]) : Cost * Animal list
*)

(***************************
* *
* 9.5 functions start here *
* *
***************************)

(* flatten function *)
fun flatten(nil) = nil
  | flatten(l::ls) = l@flatten(ls);

(* returns first element from a pair *)
fun get_first(a,b) = a;
fun get_second(a,b) = b;

(* eval_help, gets list of animals, calculates path cost in order of the list *)
fun eval_help(l::nil,i) = i
  | eval_help(l::k::ls,i) = eval_help(k::ls,i+get_first((astar l k)));

(* big eval function, takes Animal list list, returns best path + associated cost *)
fun eval(nil,best,cost) = (best,cost)
  | eval(L,nil,cost) = eval(tl(L),hd(L),eval_help(hd(L),0.0))
  | eval(L,best,cost) = if eval_help(hd(L),0.0) < cost then eval(tl(L),hd(L),eval_help(hd(L),0.0))
                                                       else eval(tl(L),best,cost);

(* simple append function *)
fun append (_,nil) = nil
  | append (L,l::ls) = (L@[l])::append(L,ls);

(* simple member function *)
fun member(nil,i) = false
  | member(l::ls,i) = if l=i then true else member(ls,i);

(* checks a list if it has duplicate entries *)
fun ck_dupl(nil) = false
  | ck_dupl(L) = if member(tl(L),hd(L)) then true else ck_dupl(tl(L));

(* removes lists with duplicate entries from a list list *)
fun rem_dupl(nil) = nil
  | rem_dupl(l::ls) = if ck_dupl(l) then rem_dupl(ls) else l::rem_dupl(ls);

(* build functions, build list of animals *)
(* the following assumptions were made:
   - a desired next animal is a not already visited one
       (this doesn't mean this doesn't happen at all, it's still possible to pass an animal which was already visited, but we won't stop and take pictures!
   - a desired next animal has a direct path to the current animal. this should still return an optimal path, altough im not completely positive on this one
     however if we don't do this assumption we have a total number of 15! = 1.3*10^12 paths to check, and we don't really want to do that ...
     with these assumptions build3([[Gate]],15) only returns 3 possible paths we have to check *)

fun build(L) = append(L,path(hd(rev(L))));
```

```
fun build2(nil) = nil
  | build2(l::ls) = build(l)::build2(ls);

fun build3(L,0) = L
  | build3(L,i) = build3(rem_dupl(flatten(build2(L))),i−1);

(* simple appends the Gate to the end, since we want to return to bremen *)
fun AppendGate(nil) = nil
  | AppendGate(l::ls) = (l@[Gate])::AppendGate(ls);

(* stores result in variables *)
val (bestpath,cost) = eval(AppendGate(build3([[Gate]],15)),nil,0.0);

(* expand the whole bath, e.g. bestpath can have Giraffe −> Gate, altohugh it actually is Giraffe −> Seals −> Gate *)
fun expand_path(Gate::nil) = [Gate]
  | expand_path(S::D::ls) = (get_second(astar S D))@expand_path(D::ls);

(* removes duplicates in list generated by the above function *)
fun rem_dupl2(l::nil)=l::nil
  | rem_dupl2(l::k::ls)=if l=k then rem_dupl2(k::ls) else l::rem_dupl2(k::ls);

(* gets list of animals, returns path *)
fun animals_to_string(Gate::nil)= "Gate"
  | animals_to_string(l::ls)= case l of
                                  Gate => "Gate -> "^animals_to_string(ls)
                                | Elephant => "Elephant -> "^animals_to_string(ls)
                                | Lion => "Lion -> "^animals_to_string(ls)
                                | Turtle => "Turtle -> "^animals_to_string(ls)
                                | Skunk => "Skunk -> "^animals_to_string(ls)
                                | Panda => "Panda -> "^animals_to_string(ls)
                                | Tiger => "Tiger -> "^animals_to_string(ls)
                                | Beaver => "Beaver -> "^animals_to_string(ls)
                                | Snake => "Snake -> "^animals_to_string(ls)
                                | Giraffe => "Giraffe -> "^animals_to_string(ls)
                                | Peacock => "Peacock -> "^animals_to_string(ls)
                                | Meerkat => "Meerkat -> "^animals_to_string(ls)
                                | Duck => "Duck -> "^animals_to_string(ls)
                                | Lama => "Lama -> "^animals_to_string(ls)
                                | Rhino => "Rhino -> "^animals_to_string(ls)
                                | Seals => "Seals -> "^animals_to_string(ls);

(* write to file function *)
fun to_file (outfile:string,list) =
        let val outs = TextIO.openOut outfile
        fun looplist(nil) = TextIO.closeOut outs |
                looplist(l::ls) = (TextIO.output(outs,l);looplist(ls))
    in
      looplist(list)
    end;

(* output list of strings *)
val output = ["Best Path:\r",animals_to_string(rem_dupl2(expand_path(bestpath))),"\r\rTotal Cost:\r",Real.toString(cost)];

(* call write function with right arguments *)
to_file("Zoo.txt",output);
```

Assignment 10: Informed Search Algorithms
(Given Apr. 14., Due Apr. 27.)

20pt

Problem 10.40 (Call of Duty)

An important war has just started around our area. You, the general of our international forces, have n elite soldiers under your command. By coincidence, there also happen to be n different battle fronts that our grand alliance is fighting on. You must show your commitment and send one specialized soldier to each front. This is not a trivial mission though. You also need to maximize the satisfaction of each of your soldiers. In a war, there are many factors that may contribute to one's morale, from distance to the homeland, to availability to apply one's specialty in combat etc. In the end the overall fitness of an assignment of soldiers to their front, which is the sum of the satisfaction of each soldier, should be as close to optimal as possible.

Think about applying Genetic Algorithms for this problem: your task is to come up with an encoding that allows only admissible states and with crossover and mutation operators that preserve admissibility.

Solution: Encoding: An n-permutation (e.g. for 8, 12376548) Crossover: There are many ways to do this, one of them is to compose the 2 permutations - permutations are functions and they can be composed (apply the first one and then the second one), yielding a permutation. E.g. composing 132 with 213 yields 312 (or 231 depending on the order, this is up to convention). Mutation: Compose with any 2-cycle permutation (i.e. one that switches 2 of the entries). E.g 12376548 can become 21376548.

Problem 10.41 (Family relations) 20pt

Using the following `ProLog` predicates:

- woman(X)
- man(X)
- mother_of(X,Y) X is the mother of Y
- father_of(X,Y) X is the father of Y

write a sample knowledge base that contains facts about the family relations of several people. Afterwards define rules for the following predicates and test them on your knowledge base:

- sister_of(X,Y) X is the sister of Y
- brother_of(X,Y) X is the brother of Y
- sibling_of(X,Y) X is the sibling of Y
- grandma_of(X,Y) X is the grandmother of Y
- grandpa_of(X,Y) X is the grandfather of Y
- uncle_of(X,Y) X is the uncle of Y
- aunt_of(X,Y) X is the aunt of Y

Solution:

```
woman(alice).
woman(sally).
woman(kate).
woman(jessica).
woman(marry).
man(john).
man(vincent).
man(patrick).
man(kevin).

mother_of(sally, kate).
mother_of(sally, vincent).
mother_of(susan,jessica).
mother_of(marry,patrick).
mother_of(alice,marry).
mother_of(sally,kevin).
father_of(john, kate).
father_of(john, vincent).
```

```
father_of(vincent,jessica).
father_of(vincent,patrick).
father_of(john,kevin).

% ------------------------------------------------
% Here are the relation predicates

parent_of(X,Y) :- mother_of(X,Y).
parent_of(X,Y) :- father_of(X,Y).
sister_of(X,Y) :- woman(X), parent_of(Z,X), parent_of(Z,Y).
brother_of(X,Y) :- man(X), parent_of(Z,X), parent_of(Z,Y).
sibling_of(X,Y) :- sister_of(X,Y).
sibling_of(X,Y) :- brother_of(X,Y).
grandma_of(X,Y) :- mother_of(X,Z), parent_of(Z,Y).
grandpa_of(X,Y) :- father_of(X,Z), parent_of(Z,Y).
uncle_of(X,Y) :- brother_of(X,Z), parent_of(Z,Y).
aunt_of(X,Y) :- sister_of(X,Z), parent_of(Z,Y).

% ------------------------------------------------
% Test queries

% The following should result in a YES
?- grandma_of(sally, jessica).
?- grandpa_of(john, jessica).
?- sibling_of(jessica, patrick).
?- sibling_of(patrick, jessica).
?- brother_of(patrick, jessica).
?- sister_of(jessica, patrick).
?- aunt_of(kate, jessica).
?- uncle_of(kevin,jessica).
?- brother_of(vincent,kate).
?- brother_of(vincent,kevin).
?- brother_of(kevin,vincent).
?- sister_of(kate,kevin).
?- sister_of(kate,vincent).
?- sibling_of(kate,vincent).
?- sibling_of(kate,kevin).
?- sibling_of(vincent,kevin).
?- sibling_of(vincent,kate).
?- sibling_of(kevin,kate).
?- sibling_of(kevin,vincent).
?- grandma_of(alice,patrick).
?- aunt_of(kate,patrick).
?- uncle_of(kevin,patrick).
?- grandma_of(sally,patrick).
?- grandpa_of(john,patrick).

% The following should FAIL
?- grandma_of(alice,jessica).
?- sibling_of(susan,marry).
?- sibling_of(susan,vincent).
```

```
?- sibling_of(kevin,john).
?- brother_of(jessica,patrick).
```

Problem 10.42 (Greatest Common Divisor in ProLog) 20pt

Write a ProLog predicate GCD(A,B,C) that is true only if C is the greatest divisor of A and B. Please use A, B and C as unary natural numbers as defined in the slides. You can assume that $A > B$.

Example:

```
?- gcd(s(s(s(s(zero)))), s(s(zero)), C).
C = s(s(zero)) .
```

read(Δ,t)oΓ

Problem 10.43 (Genetic systems) 40pt

You want to find the best solution to a system of N equations with N variables given as a product of coefficient matrix and variable vector:

$$Ax = p$$

Here, A is an N by N matrix, x is your variable vector and p is the solution vector.

For solving this problem you want to start with a set of random vectors x and use a genetic algorithm to find a solution that minimizes the error, which is defined as:

$$\varepsilon(x) = (Ax - p) \cdot (Ax - p)$$

You will have to implement a cross-over and a random mutation function. A good fitness function is the error ε itself. You are free to pick how big the population used for reproduction is.

Provide an SML function genetic which, given A (represented as a list of N lists, each of which having N floating point numbers), and p (represented as a list of N numbers), will return the vector x (represented as a list of N numbers). More exactly, the function will have the signature:

```
val genetic = fn : real list list * real list -> int
```

read(Δ,t)oΓ

Assignment 11: ProLog
(Given Apr. 27., Due May 4.)

15pt

Problem 11.44 (Girls are witches)
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 11.45 (Elementary Math) 30pt

Construct a predicate $compute(X)$ which, given a list of natural numbers as members of X, you must output all possible arithmetic operations that may take place between the respective elements, in the given order. For example, $compute([1,2,3,6])$ should yield the following solutions: $1 = 2 \cdot (3/6)$, $1 = 2 \cdot 3/6$, $1/2 = 3/6$, $1 + (2 + 3) = 6$, $1 \cdot (2 \cdot 3) = 6$, $1 + 2 + 3 = 6$, $1 \cdot 2 \cdot 3 = 6$. The allowed operators are $+, -, \cdot, /$ and brackets.

Solution:

```
% equation(L,LT,RT) :- L is the list of numbers which are the leaves
% in the arithmetic terms LT and RT - from left to right. The
% arithmetic evaluation yields the same result for LT and RT.

equation(L,LT,RT) :-
   split(L,LL,RL), % decompose the list L
   term(LL,LT), % construct the left term
   term(RL,RT), % construct the right term
   LT =:= RT. % evaluate and compare the terms

% term(L,T) :- L is the list of numbers which are the leaves in
% the arithmetic term T - from left to right.

term([X],X). % a number is a term in itself
% term([X],-X). % unary minus
term(L,T) :- % general case: binary term
   split(L,LL,RL), % decompose the list L
   term(LL,LT), % construct the left term
   term(RL,RT), % construct the right term
   binterm(LT,RT,T). % construct combined binary term

% binterm(LT,RT,T) :- T is a combined binary term constructed from
% left-hand term LT and right-hand term RT

binterm(LT,RT,LT+RT).
binterm(LT,RT,LT-RT).
binterm(LT,RT,LT*RT).
binterm(LT,RT,LT/RT) :- RT =\= 0. % avoid division by zero

% split(L,L1,L2) :- split the list L into non-empty parts L1 and L2
% such that their concatenation is L

split(L,L1,L2) :- append(L1,L2,L), L1 = [_|_], L2 = [_|_].

% do(L) :- find all solutions to the problem as given by the list of
% numbers L, and print them out, one solution per line.

do(L) :-
   equation(L,LT,RT),
      writef('%w = %w\n',[LT,RT]),
   fail.
do(_).
```

Problem 11.46 (Pythagorean Triples) 20pt

Construct a predicate $divide(X, Y, Z)$ that, given a set of elements in X and a set of cardinalities in Y, you must split the members in X into groups of the sizes specified in Y. The result should be of course a list of lists, Z.

As an example, try $divide([Andrew, Michael, Shaun, Albert, Jimmy], [1, 2, 2], Z)$. The result should be of the form $Z = [[Andrew], [Michael, Shaun], [Albert, Jimmy]]$ or $Z = [[Andrew], [Michael, Albert], [Shaun, Jimmy]]$, etc. You can assume that the sum of the cardinalities in Y is the same as the cardinality of X. Also, the output order of your solution is not important, but you must 'catch' all possibilities.

Solution:

```
fermat(2,[1,1]) :− !.
fermat(N,L) :− N > 2, littlefermat(N,L,1).

is_square(N) :− integer(N), X is sqrt(N), floor(X) =:= X.

littlefermat(N,[P,Aux],P) :− P > 0, Q is (N − P*P), Q > 0, is_square(Q), Aux is floor(sqrt(Q)), !.
littlefermat(N,L,P) :− P < N, P1 is (P+1), littlefermat(N,L,P1).

pythagorean(2,L,2) :− fermat(2,L).
pythagorean(N,L,Y) :− N>1, between(2,N,Y), littlefermat(Y,L,1).
```

Problem 11.47 (Hail to the Thief) 35pt

A valuable painting was stolen from the Liar's Club, but the police are having a hard time identifying the culprit because every statement made by a member of the Liar's Club is false. Only four members visited the club on the day that the painting was stolen. This is what they told the police:

- Ann: None of us took the painting. The painting was here when I left.
- Bob: I arrived second. The painting was already gone.
- Chuck: I was the third to arrive. The painting was here when I arrived.
- Tom: Whoever stole the painting arrived before me. The painting was already gone.

Who of these four liars stole the painting?

To solve this problem, first figure out what those liars are actually saying. Then, write a ProLog knowledge base that includes everything we know from their statements. Finally, write a ProLog predicate solution(A,B,C,D,X) which returns a possible solution of our mystery (where (A,B,C,D) are the names of the liars, in order in which they arrive at the club, and X is a number from 1 to 4, which denotes which one of them is the thief); write down the solutions you've found!

A query of the predicate could possibly look something like this (don't mind the solution presented here; it is just an example):

```
?- solution(A,B,C,D,X).
A = tom,
B = bob,
C = chuck,
D = ann,
X = 2 ;
false.
```

read(Δ,t)oΓ

Assignment 12: Internet and Revision
(Given May 4., Due May 11.)

30pt

Problem 12.48 (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.

read(Δ,t)oΓ

Problem 12.49 (POP Mail Reader) 30pt

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.

read(Δ,t)oΓ

Problem 12.50 (Email relay server) 30pt

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.

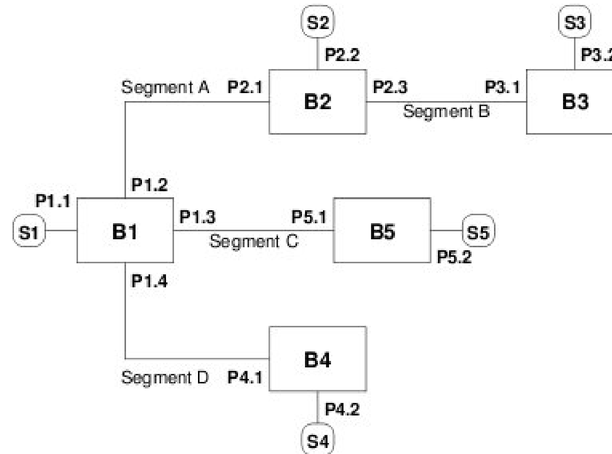
read(Δ,t)oΓ

Assignment 13: Revision
(Given May 11., Due May 18.)

read(,t)o.tex .tex

Problem 13.51 (Backward Learning) 15pt

Consider a Local Area Network (LAN) with the following topology:



where BX boxes represent bridges, SX boxes represent stations and $PX.Y$ represent port numbers. Each bridge has a so called forwarding database (made up of MAC addresses of stations and their associated port numbers) and every time they recieve a frame, they lookup an entry with a matching destination address in the forwarding database and forward the frame on the associated port. If no matching entry exists, forward the frame to all outgoing ports except the port from which the frame was received (flooding). Whenever a frame is received by a bridge, an entry is added to its forwarding database (if it does not yet exist) using the frames source address and the incoming port number.

Assume that all systems have just been initialized and that the forwarding databases of the bridges are empty. The following frames are now transmitted through the bridged LAN:

1. Station S4 sends a frame to station S5.
2. Station S5 sends a frame to station S2.
3. Station S2 sends a frame to station S5.
4. Station S2 sends a frame to station S4.

a) For each bridge, determine the contents of the forwarding database (the stations of which the MAC addresses are known, and their associated port number) after the exchange of the four frames.

b) For each frame, write down over which segments the frame is transmitted on.

Solution:

a) Contents of the bridge forwarding tables after the transmission of the four frames:

Bridge B1		Bridge B2		Bridge B3		Bridge B4		Bridge B5	
MAC	Port	MAC	Port	MAC	Port	MAC	Port	MAC	Port
S4	P1.4	S4	P2.1	S4	P3.1	S4	P4.2	S4	P5.1
S5	P1.3	S5	P2.1	S5	P3.1	S5	P4.1	S5	P5.2
S2	P1.2	S2	P2.2			S2	P4.1	S2	P5.1

b) Segments over which frames are transmitted:

Frame	Segments
$S4 \longrightarrow S5$	$\{A, B, C, D\}$
$S5 \longrightarrow S2$	$\{A, B, C, D\}$
$S2 \longrightarrow S5$	$\{A, C\}$
$S2 \longrightarrow S4$	$\{A, D\}$

Problem 13.52 (Bracket structure) 20pt

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 $\{(,),\#,Y,N\}$.

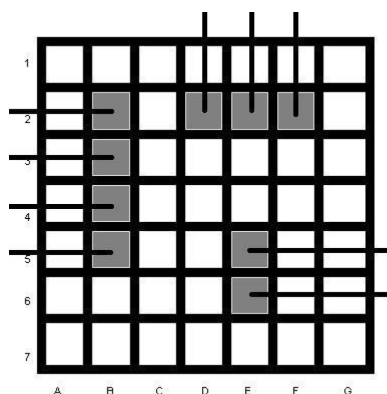
Note: Correct bracket structure is a sequence of the same amount of opening and closing brackets, where the amount of opening brackets is not less than the amount of closing brackets in any prefix of such a sequence.

Problem 13.53 (Battleship) 20pt

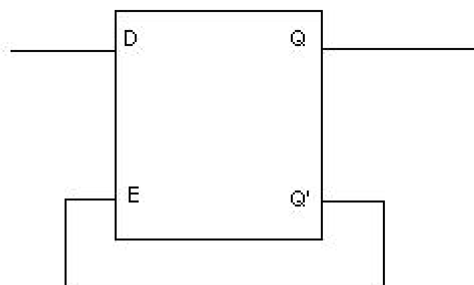
You have to design a circuit system that signals when a player of Battleship lost. The game board designed as below (where the shaded regions represent ships) provides a wire from each cell that signals 1 if the opponent hit the cell and 0 otherwise. The only difference is that in a turn, a player can shoot in 7 cells at a time, and in order to sink a ship, you have to hit all its cells in one turn (doesn't matter what else you hit). For example, to sink the 2-cell ship you have to hit at least both $E5$ and $E6$ in the same turn. Once a ship is destroyed it remains that way until the end of the game.

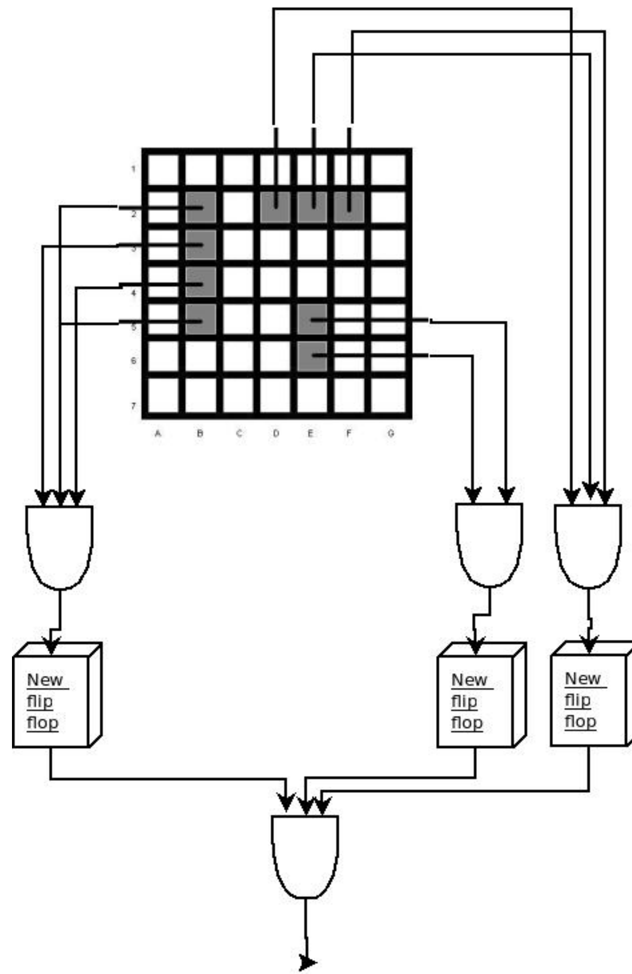
Your task:

- Use a D-flip-flop with initial state 0, to create a black-box circuit that once is set to 1, it remains that way (**Hint:** until the end of the game).
- Use this black-box in the given board circuit to create a system with 1 output wire that signals 0 while the player is still in the game and 1 once he lost.



Solution:




New
flip
flop
New
flip
flop
New
flip
flop