General Computer Science II (320201) Spring 2008

Michael Kohlhase
Jacobs University Bremen

April 8, 2013

Contents

Assignment 1: Resolution, Graphs and Trees

(Given Feb. 6., Due Feb. 13.)

15pt

Problem 1.1 (Resolution Calculus - This formula actually is not valid)
Prove the following formula in the Resolution Calculus. Show all intermediate steps (do not use derived rules of inference).

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

Solution: Clause Normal Form transformation

$$\frac{\frac{\frac{((R \Rightarrow Q) \vee P) \Rightarrow (P \Rightarrow R) \Rightarrow Q^{\mathsf{F}}}{(R \Rightarrow Q) \vee P^{\mathsf{T}}; (P \Rightarrow R) \Rightarrow Q^{\mathsf{F}}}}{(R \Rightarrow Q)^{\mathsf{T}} \vee P^{\mathsf{T}}; P \Rightarrow R^{\mathsf{T}}; Q^{\mathsf{F}}}}{(R^{\mathsf{F}} \vee Q^{\mathsf{T}}) \vee P^{\mathsf{T}}; P^{\mathsf{F}} \vee R^{\mathsf{T}}; Q^{\mathsf{F}}}$$

Resolution Proof The rest of the proof is obvious, finally we get an empty disjunction

---

Problem 1.2 (Undirected tree properties) 45pt

We've defined the notion of *path* for the directed graphs.

- Define the notion of *path* and *cycle* for the undirected graphs.

We call an undirected graph connected, iff for any two nodes $n_1 \neq n_2$ there is a path starting at n_1 and ending at n_2.

An **undirected tree** is an undirected acyclic connected graph.

Let $G = \langle V, E\rangle$. Prove or refute that the following statements are equivalent:

1. G is an undirected tree
2. For any two nodes $n_1 \neq n_2$ there is a *single* path starting at n_1 and ending at n_2
3. G is a connected graph, but it becomes disconnected after deleting any edge
4. G is connected and $\#(E) = \#(V) - 1$
5. G is acyclic and $\#(E) = \#(V) - 1$
6. G is acyclic, but adding one edge to E introduces a cycle

Solution:

- (1)⇒(2) If there were two paths, then there would be a cycle. Contradiction.
- (2)⇒(3) If G is connected and after removing an edge $\langle u, v\rangle$ from E it remains connected, it means that there were two paths between u and v. Conradiction with condition 2.
- (3)⇒(4) The connected graph has no less edges than $\#(V) - 1$. If we remove one edge G becomes unconnected that means that $\#(E) < \#(V) - 1$, but before was that $\#(E) \geq \#(V) - 1$. That means that $\#(E) = \#(V) - 1$
- (4)⇒(5) If G is connected and has a cycle then after removing edge from the cycle G should remain connected, but in this case $\#(E) = \#(V) - 2$ and it means that G is not connected.
- (5)⇒(6) Assume that addition of edge $\langle u, v\rangle$ did not make a cycle in G, that means that u and v were in the different components of connectivity. But since $\#(E) = \#(V) - 1$ then one component of connectivity (let say $\langle V_1, E_1\rangle$) has $\#(V_1) = \#(E_1)$ and it means that it has a cycle. Contradiction.
- (6)⇒ (1) If G was disconnected then we could add an edge and not to add a cycle. But it contradicts with a condition 6.

Problem 1.3 (Bipartite graphs) 30pt

Let $G = \langle V, E \rangle$ is an undirected graph. Assume that we can split V to V_1 and V_2 where $V_1 \cap V_2 = \emptyset$ and every edge from G connects a node from V_1 with a node from V_2. We call such graphs *bipartite*. If every node from V_1 is connected with every node from V_2 than such graph is called *complete bipartite graph*. Let $\#(V_1) = m$ and $\#(V_2) = n$.

- Provide an example of complete bipartite graph. Write a mathematical representation.
- Prove or refute that in complete bipartite graph $\#(V) = m + n$ and $\#(E) = mn$.

Solution:

- The one example is a star graph. The middle node is a V_1, all the others are in V_2. The mathematical representation is obvious.
- It's an obvious statement. The challenge of this exercise is to understand the notion of a bipartite graph.

Problem 1.4 (Chromatic index, bonus task) 20pt

Graph G is a k-colorable graph if we can color all edges with k colors and the adjacent edges have different colors. If graph G is k-colorable, but not $k-1$-colorable than k is a chromatic index of graph G and is marked as $X_e(G) = k$.

- Prove or refute that $X_e(G) = \max n, m$ where G is a complete bipartite graph and n is a number of the nodes in V_1, m is a number of nodes in V_2.

Solution: We should just provide a painting of the graph and it will become obvious.

Assignment 2: Combinatorial Circuits
(Given Feb. 13., Due Feb. 20.)

25pt

Problem 2.5 (Length of the inner path in balanced trees)
Prove by induction or refute that in a balanced binary tree the length of the inner path is not more than $(n+1)\lfloor\log_2(n)\rfloor - 2\cdot 2^{\lfloor\log_2(n)\rfloor} + 2$. Here n is the number of nodes in the graph.

Note: Length of the inner path is the sum of all lengths of paths from the root to the nodes.

Solution: This solution is courtesy of Andrea Georgescu.
To show this, it is enough to show that if the tree is fully balanced

$$l(n) = (n+1)\lfloor\log_2(n)\rfloor - 2\cdot 2^{\lfloor\log_2(n)\rfloor} + 2$$

Here $l(n)$ denotes the the length of the inner path.

- $\#(V) = 2^{d+1} - 1$ ($d \to$ length, tree is fully balanced) $n = 2^{d+1} - 1$
- $l(d) = 2^{d+1}\lfloor\log_2(2^{d+1}-1)\rfloor - 2\cdot 2^{\lfloor\log_2(2^{d+1})-1\rfloor} + 2 = 2^{d+1}\cdot d - 2\cdot 2^d + 2 = 2^{d+1}(d-1) + 2$
- Now, we do induction on d
  - $d = 0 : l(d) = 0$
  - $d = 1 : l(d) = 2$
- Assume the statements holds for $d \leq k$
- $d = k+1$, when we add one more row of leaves, the path from the cost to each of these is $k+1$. So $l(k+1) = l(k) + (k+1)\cdot 2^{k+1} = 2^{k+1}(k-1) + 2 + 2^{k+1}(k+1) = 2^{k+2}\cdot k + 2$

Problem 2.6 (DNF Circuit with Quine McCluskey) 25pt

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

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

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

$$f_1(X) = X_1 \wedge \neg X_2$$

$$f_2(X) = (X_1 \vee \neg X_3) \wedge \neg X_1 \vee \neg X_2$$

$$f_3(X) = (\neg X_1 \vee \neg X_2) \wedge \neg X_3$$

Draw a circuit for these function is a trivial task (see similar tasks).

Problem 2.7 (Binary Comparator) 25pt

Design a combinational circuit from AND, OR, and NOT gates that takes three n-bit binary numbers and returns 1 if at least two of them are equal and zero otherwise. What is the depth of your circuit? Modify the first circuit to a circuit that returns 1 if all of three n-bit binary numbers are equal.

Note: n-bit binary number is a sequence of 1 or 0 with n elements ($n^{\{0,1\}}$).

Solution: This is quite simple. Write a Boolean expression which does it. Then draw a circuit.

Problem 2.8 (Is XOR universal?) 25pt

Imagine a logical gate XOR that computes the logical exclusive disjunction. Prove or refute whether the set $S = \{\mathrm{XOR}\}$ is *universal*, considering the following two cases:

1. combinational circuits without constants
2. combinational circuits with constants

If the set turns out to be *not* universal in either of the cases, add *one* appropriate non-universal gate $G \in \{\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}\}$ to S, and prove that the set $S' = \{\mathrm{XOR}, G\}$ is universal.

Note: A set of Boolean function is called *universal* (also called "functionally complete"), if *any* Boolean function can be expressed in terms of the functions from that set. {NAND} is an example from the lecture.

Solution: This solution is left to TAs. We had a similar task (impication-universality).

Assignment 3: Combinatorial Circuits, Arithmetics

(Given Feb. 20., Due Feb. 27.)

30pt

Problem 3.9 (Half Adder)

Design an explicit combinational circuit for the half-adder using only NAND gates. What is its cost and depth? Looking at the first, straightforward solution, can cost and depth be improved?

Solution: A half-adder is a combination of an AND and a XOR gate. The rather difficult part is how to express the XOR gate by NAND gates. Solution steps: First express the XOR gate by AND, OR and NOT gates then express each of these gates by NAND gates.

Problem 3.10 (Carry Chain and Conditional Sum Adder) 40pt

Draw an explicit combinational circuit of a 4-bit Carry Chain Adder and a 4-bit Conditional Sum Adder. Do not use abbreviations, but only NOT, AND, OR gates. Demonstrate the addition of the two binary numbers $\langle 1, 1, 0, 1\rangle$ and $\langle 0, 1, 0, 1\rangle$ on both adders; i.e. annotate the output of each logic gate of your adders with the result bit for the given two binary numbers as input of the whole adder.

Problem 3.11 (Combinational Circuit for Shift) 30pt

Design an explicit 4-bit shifter (combinational circuit) (using only NOT, AND and OR gates) that corresponds to $f_{\text{shift}} \colon \mathbb{B}^4 \times \mathbb{B} \times \mathbb{B} \to \mathbb{B}^4$ with

$$f_{\text{shift}}(\langle a_3, a_2, a_1, a_0 \rangle, s_1, s_2) \begin{cases} \langle a_3, a_2, a_1, a_0 \rangle & \text{if } s_1 = 0, s_2 = 0 \\ \langle a_2, a_1, a_0, 0 \rangle & \text{if } s_1 = 1, s_2 = 0 \\ \langle 0, a_3, a_2, a_1 \rangle & \text{if } s_1 = 0, s_2 = 1 \\ \langle a_0, a_3, a_2, a_1 \rangle & \text{if } s_1 = 1, s_2 = 1 \end{cases}$$

Solution:

Assignment 4: Sequential Logic Circuits, Memory, Register Machine
(Given Feb. 27., Due Mar. 5.)

15pt

Problem 4.12 (RS Flipflop from NAND gates)
Construct the RS flipflop *only* from NAND gates. Draw the sequential logic circuit as a graph, as well as a state table.

Solution: The solution looks exactly like the NOR implementation, with the NOR's replaced by NAND's, but the inputs need to be inverted using NOT's first. Then we should replace NOT by corresponding NAND's represantation.

See `http://www.play-hookey.com/digital/rs_nand_latch.html`

Problem 4.13 (3-bit Address Decoder) 25pt

Design a three-bit address decoder using only NOR gates. Draw your circuit as a graph.

Problem 4.14 (Exchanging elements) 30pt

Given are $n \geq 0$ (n is stored in $P(0)$) and $i \geq 2$ (i is stored in $P(1)$). $2n$ integers are stored in an array $P(i) \dots P(i + 2n - 1)$. Write an assembler program that exchanges k-th and $2n - k - 1$-th elements of array for all $k.0 \leq k < n$ (assuming that the beginning of an array starts at i).

Problem 4.15 (A Recursive Function) 30pt

Write an assembler program that computes the following recursive function:

$$\begin{aligned} f(0) &= 1 \\ f(i) &= (-4)^i \cdot f(i-1), i \geq 1 \end{aligned}$$

Assume that the input i is given in cell 0. Store the result in cell 3. If you need additional assembler instructions for which we know circuits from the lecture (e.g. left/right shifting n-bit numbers, or inverting them, i.e. computing the bit-wise NOT), you may give them descriptive names like *shiftr* or *inv*. But be sure to use only circuits that implement functions $f\colon \mathbb{B}^n \to \mathbb{B}^n$!

Assignment 5: Register Machine, Virtual Machine

(Given March 5., Due March 12.)

10pt

Problem 5.16 (Stack in SML)
Implement stack in SML with all methods covered on the lecture.

Problem 5.17 (Divider) 15pt

Assume the data stack initialized with `con` a and `con` b for some natural number a and b. Write a $\mathcal{L}$(`VM`) program that returns on top of the stack $a\,div\,b$. Furthermore draw the evolution of the stack during the execution of your program for $a = 15$ and $b = 4$.

Problem 5.18 (Convert Highlevel Code to $\mathcal{L}$(VM) Code) 25pt

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

```
for i := 0 to 9 do
 for j := 0 to 9−i do
  if a[j] > a[j+1] then
     swap(a[j], a[j+1]);
```

Suppose the array is loaded on stack (top value being A[10]). Convert the code into $\mathcal{L}$(VM) code, assuming that swap(a, b) is a function which swaps values of variables.

Problem 5.19 (The Catalan Numbers) 50pt

The n^{th} Catalan number C_n is the number of rooted ordered binary trees with $n+1$ leaves. We can express the Catalan number using the following recurrence relation:

$$C_0 = 1; \qquad C_n = \sum_{i=0}^{n-1} C_i \cdot C_{n-i-1}$$

for all $n \geq 1$.

Assume the data stack initialized with `con` n for some natural number n. Write a $\mathcal{L}$(`VM`) program that calculates C_n using recurrence relation above.

Assignment 6: Virtual Machine
(Given March 12., Due March 19.)

30pt

Problem 6.20 (New Statements for SW)

Extend the Simple While Language SW by a repeat−until and a for loop, two variants of the while loop. Consider both the abstract syntax data type sta and the compileS function.

The concrete syntax of a repeat−until loop looks as follows:

```
repeat
  (* statements *)
until (* conditional−expression *);
```

. . . where the statements in the body are repeated until the conditional expression evaluates to true (i.e. a number $\neq 0$).

The concrete syntax of a for loop looks as follows:

```
for (* counting−variable *) := (* start−expr *) to (* end−expr *) do
  (* statements *)
end;

(* Example:
var i;
var n := 10;
var sum := 0;
for i := 1 to n do
  sum := sum + i;
end; *)
```

Assume that a counting variable, e.g. named i, has been declared before. In the first run of the loop, i is initialized to the value of *start-expr*. After each run of the loop, i is increased by one. In the last run of the loop, i has the value of what *end-expr* evaluated to *before* the loop started; then the execution of the loop stops.

Problem 6.21 (Compiling If-construct) 30pt

Write an SML function simpleif that takes a string of form

IF $\RMdatastore{X}$ rel $\RMdatastore{Y}$ THEN $\RMdatastore{X}$+$\RMdatastore{Y}$ ELSE X+

where X, Y are non-negative integers, *rel* an element of $\{\leq, \geq, >, <, =, \neq\}$, and returns the appropriate $\mathcal{L}$(VM) language piece of code. For example, given

IF $\RMdatastore{2}$ $>$ $\RMdatastore{3}$ THEN $\RMdatastore{2}+\RMdatastore{3}$ ELSE 5

your function should return an array of $\mathcal{L}$(VM) instructions that reads the second and third cells of the stack and pushes their sum if the content of second cell is greater than the content of third cell, and $2 + 3 = 5$ otherwise. Note that your program should treat any (non-empty) concatenation of white-space as a single white-space.

Raise exception if the input string is not valid.

Problem 6.22 (Static Procedure for a Residue of Division) 40pt

Write a $\mathcal{L}(\mathtt{VM})$ program that implements the `mod` function for a residue of integer division as a static procedure and calls that procedure to compute $3\,mod\,2$. You may use any $\mathcal{L}(\mathtt{VM})$ instruction except `peek` and `poke`. Draw the evolution of the stack, including all intermediate steps.

Assignment 7: Turing Machines
(Given April 03, Due April 10)

25pt

Problem 7.23 (Bracket structure)
Design the following Turing Machine. Its input is a word $w \in \{(,)\}^*$, surrounded by hash marks as delimiters. You can assume that your head is initially positioned right after the first hash. Upon termination, your program should replace the second hash with Y or N indicating if the bracket structure is correct (Y - for correct, N otherwise). Thus the overall alphabet is $\{(,),\#,Y,N\}$.

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 7.24 (Eliminating zeros) 25pt

Design the following Turing Machine. Its input is a word $w \in \{0,1\}^*$, surrounded by hash marks as delimiters. You can assume that your head is initially positioned right after the first hash. Thus the overall alphabet is $\{0, 1, \#\}$. Upon termination, your program should compact the word by deleting all zeros from it, i.e. the word 11101010001 should become 111111. The beggining of the final word should be right after the first dash.

Problem 7.25 (SML Implementation of a Turing Machine Simulator) 40pt

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

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

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

```
    type Tape = Int −> Alphabet
    type TransTable = ((State ∗ Alphabet) ∗ (State ∗ Alphabet ∗ Direction)) list
```

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

Solution:

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

type Tape = int −> Alphabet
type TransTable = ((State ∗ Alphabet) ∗ (State ∗ Alphabet ∗ Direction)) list

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

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

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

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

(∗ Test 1: half bit adder ∗)

val MYTAPE_11 = fn 1 => one | 2 => one | n => zero;
val MYTAPE_10 = fn 1 => one | 2 => zero | n => zero;
val MYTAPE_01 = fn 1 => zero | 2 => one | n => zero;
val MYTAPE_00 = fn 1 => zero | 2 => zero | n => zero;
val MYTABLE = [(1, zero, 2, zero, R), (1, one, 3, one, R),
                 (2, zero, 4, zero, R), (2, one, 5, one, R),
                      (3, zero, 5, zero, R), (3, one, 6, one, R),
                                (4, zero, 7, zero, R), (4, one, 7, zero, R),
                                (7, zero, 9, zero, L), (7, one, 9, zero, L),
                                (5, zero, 8, zero, R), (5, one, 8, zero, R),
                                (8, zero, 9, one, L), (8, one, 9, one, L),
                                (6, zero, 7, one, R), (6, one, 7, one, R)];

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

(* Test 2: one's duplicator *)

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

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

val result2 = TM MYTABLE2 MYTAPE2;

(* another *)

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

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

(* another2 :) *)

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

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

val RESULT2 = TM TABLE2 TAPE2;
```

Assignment 8: Problems and Searching
(Given April 09, Due April 16)

Conjecture 1 *Let $G = \langle V, E\rangle$ be a undirected graph. Then $\sum_{i=1}^{\#(V)} deg(v_i)$ is even.*

Proof: by induction on $\#(V)$

P.1.1 $\#(V) = 1$ **(base case)**: The sum is 0 because there are no edges => even □

P.1.2 $\#(V) = n \rightarrow \#(V) = n+1$ **(induction step)**: We know that for $\#(V) = n$, the sum over all nodes of their degrees is even by I.H. To have $n+1$ nodes, we need to add a node $v_n + 1$ to the graph. This node will be connected to some other node v_j, so the edge uniting them adds 2 to the overall sum of degrees. Thus we have that

$$\sum_{i=1}^{n+1} \deg(v_i) = 2 + \sum_{i=1}^{n} \deg(v_i)$$

and since $\sum_{i=1}^{n} \deg(v_i)$ is even by I.H., the new sum is even as well. □

□

15pt

Problem 8.26 (Wrong induction proof)
Many of you wrote proof above in the midterm. Can you find what is wrong with it?

Note: In doing induction proofs, you need to make sure that your proof will cover all possible cases. For this, you need a good induction step that will take you from one case to the next more complicated case without leaving anything in between. Also, the set of base cases are important: starting from this set, you have to be able arrive at any other case by just applying the induction step. Another thing about the base cases is that they should not be unnecessarily many.

Solution: The problem with the proof is that it doesn't cover all cases, meaning you cannot reach all graphs by applying the induction step, because the induction step always puts exactly one edge between the new node and the rest of the graph. To get from 1 node to 2 nodes, you don't necessarily need to place an edge, the graph $(\{1, 2\}, \{\})$ will never be reached.

Problem 8.27 (Set Operations) 20pt

Given the three sets $A := \{a, b, c, d\}$, $B := \{1, 2, 3, 4\}$ and $C := \{v, w, x, y, z\}$. Write out the following *nonempty* relations in math notation. If such relations do not exist, briefly state why.

1. a total function $f \subseteq A \times C$ that is injective but not bijective
2. a total function $f \subseteq A \times B$ that is injective but not bijective
3. a relation $R \subseteq A \times A$ that is symmetric
4. a relation $R \subseteq B \times B$ that is reflexive and transitive
5. a linear order $R \subseteq C \times C$
6. a total function $f \subseteq C \times B$ whose converse relation f^{-1} is a partial function

Out of the relations you provided in 1...6, identify all the ones that are equivalence relations.

Solution:

1. $f: A \to C$; $f = \{\langle a, v\rangle, \langle b, w\rangle, \langle c, x\rangle, \langle d, y\rangle\}$
2. DNE. Since $\#(A) = \#(B)$ all injective total functions are also bijective[1]
3. $R \subseteq A \times A$ where $R = \{\langle a, b\rangle, \langle b, a\rangle, \langle c, c\rangle, \langle d, a\rangle, \langle a, d\rangle\}$
4. $R \subseteq B \times B$ where $R = \{\langle 1, 1\rangle, \langle 1, 2\rangle, \langle 2, 2\rangle, \langle 3, 3\rangle, \langle 4, 4\rangle\}$
5. $R \subseteq C \times C$ where $R = \{\langle v, w\rangle, \langle w, x\rangle, \langle x, y\rangle, \langle y, z\rangle\}$
6. DNE. A total function $f \subseteq C \times B$ is not injective, so the converse relation f^{-1} is not a function

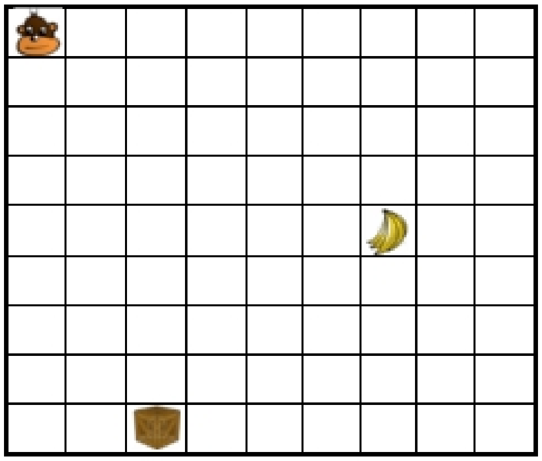
[1]EdNote: need to argue that or point to an assertion

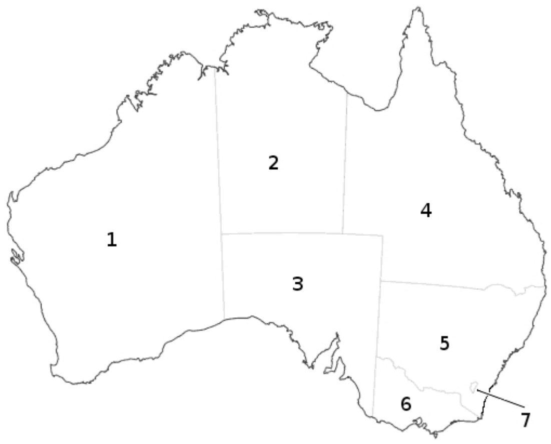
Problem 8.28 (Problem formulation and solution) 35pt

a) Write a problem formulation and path cost for each of the following problems:

1. A monkey is in a room with a crate, with bananas suspended just out of reach on the ceiling. The monkey would like to get the bananas.
2. You have to color a complex planar map using only four colors, with no two adjacent regions to have the same color.

b) Given the following concrete examples of the two problems from (a), provide a solution for each of the examples that conforms the problem formulation you gave in (a) and specify the cost of this solution according to the path cost you defined.





Solution:

a) 1.

$$
\begin{aligned}
P &= \langle S, O, I, G\rangle \\
S &= \{\langle a_1, a_2, a_3, a_4, a_5, a_6, a_7\rangle \,|\, a_i \in \{nocolor, color1, color2, color3, color4\}\} \\
O &= \{placecolor.ionmap\ section\ j \,|\, i \in \{0,1,2,3,4\}, j \in \{1,2,3,4,5,6,7\}\} \\
I &= \langle nocolor, nocolor, nocolor, nocolor, nocolor, nocolor, nocolor\rangle \\
G &= \{s \in S \,|\, every\ 2\ neighboring\ regions\ have\ different\ color\}
\end{aligned}
$$

path cost 1 for every coloring step

2. 1. See above. One option is to make a grid out of the room and give every cell a number, then the states are numbers (where the crate is), operators are left, right, up, down, initial state is initial number of the cell with the crate, goal state is number of the cell with bananas.

b) 1.

$$right, right, right, right, up, up, up, up$$

cost 8

2.

$$red, black, green, red, orange, black, red$$

cost 7 (correspondingly to enumeration of the areas)

Problem 8.29 (Connected graphs) 30pt

Consider the following SML datatype and exception definitions:

```
type node = int;
type edge = (node * node);
type Graph = (node list * edge list);
exception InvalidGraph;
```

Write an SML function Connected : Graph −> bool that returns true if the given graph is connected and returns false if it is not connected. The provided graphs should be considered to be indirected. Thus the following two edges are equivalent : $(1,2)$, $(2,1)$. If the input graph is invalid you should raise InvalidGraph. Here is a sample input and, please, stick to it.

```
Connected ([1,2,3], [(1,2),(2,3)]);
val it = true : bool
```

Solution: A solution would look like this:

```
(* Datatype definition *)
type node = int;
type edge = (node*node);
type Graph = (node list * edge list);
exception InvalidGraph;

(* Returns true if x exists in the specified list. *)
fun exists x nil = false
  | exists x (a::b) = if x=a then true else exists x b;

(* Returns true if all the elements from the first list exist in the second. *)
fun contains_all nil l = true
  | contains_all (a::b) l = if exists a l then contains_all b l else false;

(* Returns a list that has a on top if it was not in the original list. *)
fun add_if_new a l = if exists a l then l else (a::l);

(* Returns a list of all nodes that are accessible from nodes using the edges in a single step. *)
fun add_one nodes nil = nodes
  | add_one nodes ((a,b)::edges) = if exists a nodes then add_one (add_if_new b nodes) edges
                                   else (if exists b nodes then add_one (add_if_new a nodes) edges else add_one nodes edge

(* Returns a list of all nodes that are accessible from nodes using the edges. *)
fun add_all nodes edges =
      let
        val a = add_one nodes edges;
        val b = add_one a edges;
      in
        if (a <> b) then add_all b edges else b
      end;

(* Verifies that the provided edges refer only to existing nodes. *)
fun check_correct_edges nodes nil = true
  | check_correct_edges nodes ((a,b)::edges) =
        let
            val ok = (exists a nodes) andalso (exists b nodes);
        in
            if ok then true else raise InvalidGraph
        end;

(* Returns true if a graph is connected, and false otherwise. *)
fun Connected ((nil, nil):Graph) = true
   | Connected ((nil, l):Graph) = raise InvalidGraph
   | Connected ((l, nil) : Graph) = false
   | Connected ((nodes, ((a,b)::edges)) : Graph) =
        let
            val ok = check_correct_edges nodes edges;
            val connected_nodes = add_all (a::b::nil) edges;
        in
          contains_all nodes connected_nodes
        end;
```

The test cases are:

```
(* Test Cases *)
val test1 = Connected ([], []) = true;
val test2 = Connected ([1,2,3], []) = false;
val test3 = Connected ([1,2,3], [(1,2),(2,3)]) = true;
val test4 = Connected ([1,2,3,4,5], [(1,2),(2,3)]) = false;
val test5 = Connected ([1,2,3], [(1,2),(2,3)]) = true;
val test6 = Connected ([1,2,3], [(2,1),(2,3)]) = true;
val test7 = Connected ([1,2,3], [(2,1),(3,2)]) = true;
val test8 = Connected ([1,2,3], [(3,2),(2,1)]) = true;
val test9 = Connected ([3,1,2], [(3,2),(2,1)]) = true;
val test10 = Connected ([1,2,3,4], [(2,4),(1,2),(2,3)]) = true ;
val test11 = Connected ([1,2,3,4,5,6,7,8], [(8,7),
(2,4),(1,2),(2,3),(4,5),(5,6),(6,7)]) = true;
val test12 = Connected ([1,2,3,4], [(2,4),(2,3)]) = false;
val test13 = Connected ([1,3,2,3,3], [(1,2),(2,3)]) = true;
val test14 = Connected ([1,2,3], [(1,2),(2,3),(1,3)]) = true;
val test15 = Connected ([1,2,3], [(1,2),(2,3),(1,2),(2,3)]) = true;
val test16 = let val t = Connected ([], [(1,2),(2,3)]); in false end
handle InvalidGraph => true;
val test17 = let val t = Connected ([1,2], [(1,2),(2,3)]); in false end
handle InvalidGraph => true;
val test18 = let val t = Connected ([1,2,3], [(1,1),(2,3)]); in false
end handle InvalidGraph => true;
```

Assignment 9: Problems and Searching
(Given April 16, Due April 23)

15pt

Problem 9.30 (Bijective mapping)
Write a bijective function $f\colon A \to B$ for the following three combinations of A and B:

1. $A = \{2k \,|\, k \in \mathbb{N}\}$, $B = \{2k + 1 \,|\, k \in \mathbb{N}\}$
2. $A = \{k \,|\, k \in \mathbb{N}\}$, $B = \{k \,|\, k \in \mathbb{Z}\}$
3. $A = \{k \,|\, k \in \mathbb{R}, k \in [0, 1)\}$, $B = \{k \,|\, k \in \mathbb{R}, k \in [0, \infty)\}$

Note: $\mathbb{N}$ is the set of natural numbers, $\mathbb{Z}$ is the set of integers and $\mathbb{R}$ is the set of real numbers.

Solution:

1. $f(x) = x + 1$ (3 points)
2. $f(x) = -\frac{k}{2}$, if k even, $\frac{k+1}{2}$ otherwise (5 points)
3. $f(x) = \frac{1}{1-x} - 1$ (7 points)

Problem 9.31 (Complexity) 15pt

Find the time and space complexities of breadth-first search and depth-first search (and prove your results). For breadth-first search, consider b to be the branching factor (i.e. consider a hypothetical state space where every state can be expanded to yield b new states) and suppose that the solution for this problem has a path length of d. For depth-first, b is the braching factor and m is the maximal depth reached. Consider the number of nodes expanded before finding a solution for time and the number of nodes that must be mantained in memory for space.

Solution: BFS time: $1 + b + b^2 + b^3 + \cdots + b^d$ so $O(b^d)$

BFS space: all leaf nodes must be mantained in memory thus $O(b^d)$.

DFS time: $O(b^m)$.

DFS space: requires storage of only bm nodes, in contrast to the b^d that would be required by BFS.

Problem 9.32 (Search comparison) 10pt

Give a concrete example of a state space in which iterative deepening search performs much worse than depth-first search (for example $O(n^2)$ vs. $O(n)$). State exactly why this is the case (i.e briefly show that the time complexity is better for depth-first search in this example). In general, can you think of a certain type of problems where iterative deeping search will almost always perform worse than Depth first search.

Solution: There are a lot of such problems, but one of them could be:

There are n boxes labled $1 \cdots n$ in the north side of a room. Create an agent that will decide on an order for them to be put on the south side of the room. The first box that is moved has to be box 1.

For this, the states are pairs $\langle\{\text{boxes in north}\}, \{\text{boxes in south}\}\rangle$, the second set is 1 after the first step, then more and more boxes are added to this and removed from the north set. Since any permutation would do, the agent that is implemented with depth first search will find the solution in $n-1$ steps. On the other hand, the agent implemented with iterative deepening search will analyze alomost the whole tree before arriving at the solution.

Problem 9.33 (Implementing Search) 35pt

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 9.34: Write the next function, goal predicate and initial_state variable for the 8-puzzle presented on the slides (please check the slides for the description). Then use these to test your breadth-first and depth-first search algorithms from the previous problem. 25pt

Use the following :

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

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

Sample testcase:

```
test call : next(initial_state);

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

Solution:

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

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

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


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


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


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


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


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

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


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



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


(*Testcases*)

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

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

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

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

Assignment 10: Math, Searching, Turing Machine
(Given April 23, Due April 30)

Problem 10.35 (Two's complement) 10pt

a) Convert the decimal number 10 into two's complement using the minimal number of bits possible. How many bits do you need?

b) Convert the decimal number −10 into two's complement using the result from a). What is the algorithm that you use here?

c) Convert the decimal number 10 into a 9-bit two's complement number using the result from a). What is the operation that you need to do here.

d) Convert the decimal number −10 into a 9-bit two's complement number using the result from c). (so apply the same algorithm as in b))

e) Convert the decimal number −10 into a 9-bit two's complement number using the result from b) and the same operation from c).

Solution:

a) 01010 $\rightarrow$ 5 bits

b) flip all bits and add 1: 10110

c) sign bit extension: 000001010

d) flip all bits and add 1: 111110110

e) sign but extension of b: 111110110

Problem 10.36 (Greedy with particular heuristics) 10pt

Suppose that we run a greedy search algorithm with $h(n) = 1/g(n)$ and that the path cost is always positive. (Refer to the slides for the definition of $g(n)$) What kind of search strategy will this result in? Explain.

Solution: It will always expand the deepest node because the path is always greater than 1, so the node chosen is the one with the highest path cost. This implies it will go as deep as it can first $\Rightarrow$ DFS.

Problem 10.37 (Heuristics) 20pt

Come up with a heuristic function for the problem of algebraic equation solving, i.e. $f(x) = 0$. What are the states in this problem? Is your h admissible? Can the state space contain local minima (so states in which an algorithm like greedy could get stuck)?

Solution:

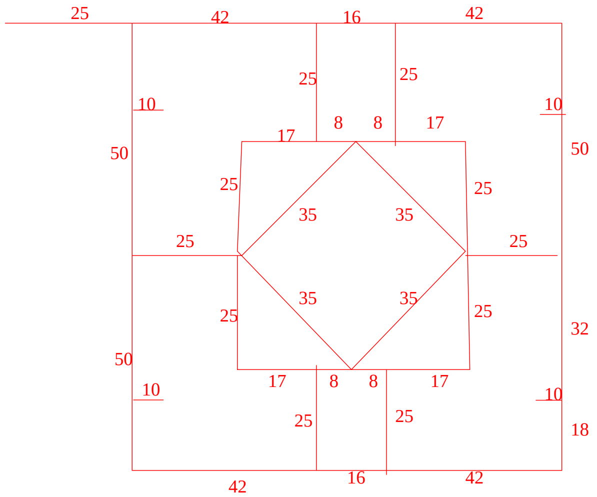
- x in R are the states
- $-h(x) = |f(x)|$ (the closer you are to 0, the smaller the cost to getting there is)
- no, it is not admissible, as it is not always true that $h(x) < |x - x_0|$, where x_0 is the closest solution (or even any solution)
- yes, it can contain local minima: imagine $f(x)$ has a positive local minimum at x_1, then x_1 would be a local minimum state - greedy can get stuck there

---

Problem 10.38 (*A^** **search on Jacobs campus**) 45pt

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

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



Solution:

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

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

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

fun heuristic(n,m) = let
 val (x1,y1) = coor(n);
 val (x2,y2) = coor(m);
```

```
in Real.round(Math.sqrt(Real.fromInt((x1−x2)∗(x1−x2)+(y1−y2)∗(y1−y2))))
end;

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

exception NoSolution;

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

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

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

Problem 10.39 (Halting problem) 15pt

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.

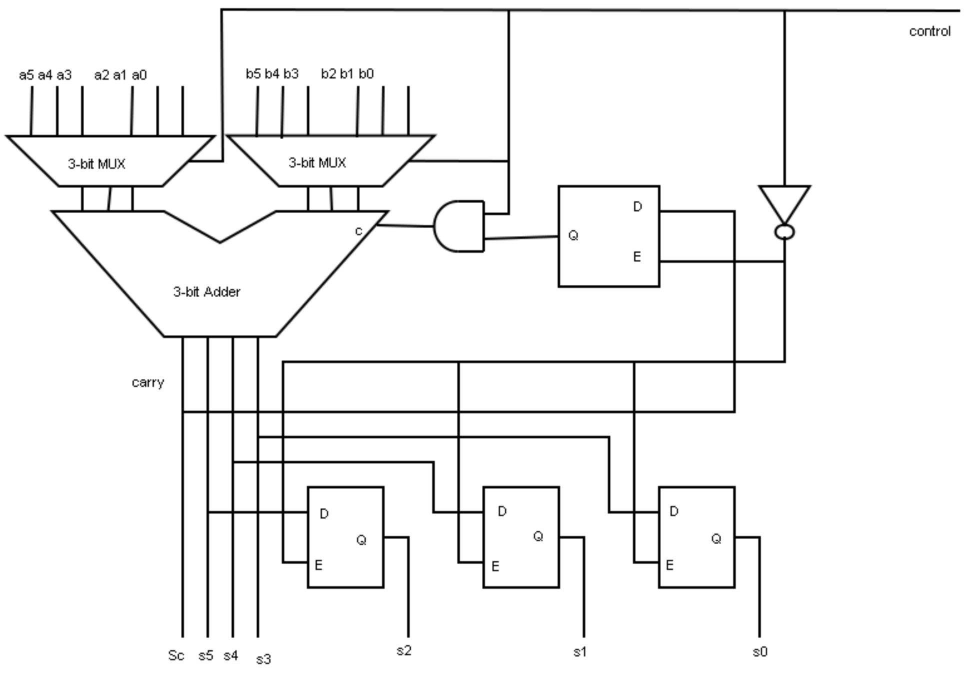
Solution: From any turing machine A and an input for it I we can construct a turing machine B, that first prints I on the input tape and then simulates A on I. We can then use M to decide if B will halt or not and thus solve the halting problem.

Assignment 11: ProLog
(Given April 30, Due May 7)

25pt

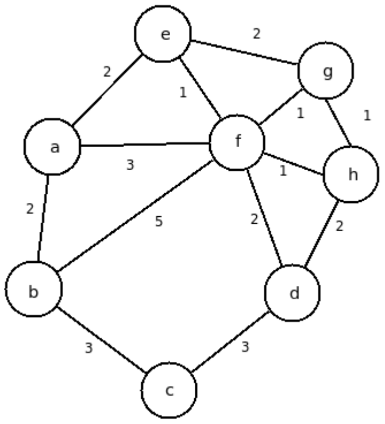
Problem 11.40 (2-Stage Adder)
Design a circuit that computes the sum of two 6-bit numbers. In your solution you can use only a single 3-bit Adder, you are not allowed to implement an additional adder using elementary gates. You have to perform the computation in two steps. Therefore an additional control input is available. At first it will be 0. Then it will be set to 1 (you do not have to implement this yourself). After that the output of your circuit should represent the sum of the two numbers including a carry bit. You may use all circuit gates and block from the lecture notes.

Solution:



Problem 11.41 (Path Cost) 15pt

Represent the graph below as facts in a `ProLog` knowledge base and write a predicate has_path(I,G,C) that is true if there exists a path from node I to node G that is of cost less or equal to C. Assume that every node has a path to itself with a cost of 0.



Here is sample run:

```
?−has_path(a,g,5).
Yes
```

Solution:

```
edge(a,b,2).
edge(a,e,2).
edge(a,f,3).
edge(b,a,2).
edge(b,c,3).
edge(b,f,5).
edge(c,b,3).
edge(c,d,3).
edge(d,c,3).
edge(d,f,2).
edge(d,h,2).
edge(e,a,2).
edge(e,f,1).
edge(e,g,2).
edge(f,a,3).
edge(f,b,5).
edge(f,d,2).
edge(f,e,1).
edge(f,g,1).
edge(f,h,1).
edge(g,e,2).
edge(g,f,1).
edge(g,h,1).
edge(h,f,1).
edge(h,d,2).
edge(h,g,1).

has_path(G,G,C) :- C >= 0.
has_path(I,G,C) :- C >= 0, edge(I,X,Y), Z is C-Y, has_path(X,G,Z).
```

Test Cases:

```
has_path(a,c,2). %-> No
has_path(g,b,5). %-> No
has_path(c,e,-3). %-> No
has_path(a,d,5). %-> Yes
has_path(d,a,5). %-> Yes
has_path(c,c,5). %-> Yes
has_path(h,e,2). %-> Yes
```

Problem 11.42 (Intersection of linear functions) 15pt

Recall from mathematics that a linear function $f(x)$ is defined by two parameters a and b such that $f(x) = ax + b$. Write a predicate intersect that receives defining parameters of 3 linear functions and evaluates to *Yes*, if these functions intersect in one point and *No* otherwise. More specifically, the program intersect takes 6 variables such that intersect(a,b,c,d,e,f) returns *Yes*, if the functions $\lambda x.ax + b$, $\lambda x.cx + d$ and $\lambda x.ex + f$ intersect in one point. Make sure your program works on any combination of functions.

Here is a sample run:

```
?−intersect(0,0,3,3,−3,−3).
Yes
```

Solution:

```
intersection_point(A,B,A,B,_).
intersection_point(A,B,C,D,P):− (A − C) =\= 0, P is (D − B) / (A − C).

intersect(A,B,C,D,E,F):− intersection_point(A,B,E,F,P), intersection_point(C,D,E,F,P).
```

Test cases:

```
intersect(0,0,3,3,−3,−3). %−> Yes
intersect(2,3,4,5,6,7). %−> Yes
intersect(4,0,2,0,1,0). %−> Yes
intersect(2,3,2,3,1,1). %−> Yes
intersect(2,3,2,3,2,5). %−> No
intersect(2,4,2,5,2,1). %−> No
intersect(0,0,4,5,12,1). %−> No
intersect(2,4,2,5,7,1). %−> No
```

Problem 11.43 **(Tracing ProLog Evaluation)** 15pt

Given the following ProLog knowledge base, trace the query evaluation. Keep in mind that ProLog evaluates queries using depth-first search with backtracking. Please refer to the slides for more detailed information.

```
sibling(john,jane).
sibling(anne,john).
sibling(john,susan).
father_of(john,paul).
father_of(alyssa,paul).
child_of(sarah,jeremy).
child_of(laura,alyssa).

mother_of(C,M):− child_of(M,C).
mother_of(C,M):− sibling(C,S), child_of(M,S).
mother_of(C,M):− sibling(S,C), child_of(M,S).
mother_of(C,M):− father_of(C,F), father_of(S,F), child_of(M,S).

?−mother_of(john,M).
```

Solution:

```
?−mother_of(john,M).

?−child_of(M,john). %backtrack point 1
FAIL %fails, backtrack to 1

?−sibling(john,S), child_of(M,S). %backtrack point 1
S = jane %backtrack point 2
?−child_of(M,jane).
FAIL %fails, backtrack to 2
S = susan %backtrack point 2
?−child_of(M,susan).
FAIL %fails, backtrack to 1

?−sibling(S,john), child_of(M,S). %backtrack point 1
S = anne %backtrack point 2
?−child_of(M,anne).
FAIL %fails, backtrack to 1

?−father_of(john,F), father_of(S,F), child_of(M,S). %backtrack point 1
F = paul %backtrack point 2
?−father_of(S,paul), child_of(M,S).
S = john %backtrack point 3
?−child_of(M,john).
FAIL %fails, backtrack to 3
S = alyssa %backtrack point 3
?−child_of(M,alyssa).
M = laura
Yes
```

Problem 11.44 (Merging Dictionaries) 30pt

Consider a dictionary data structure represented as a list of key = value pairs. Implement a predicate dict_merge(dict1,dict2,merger) that merges one dictionary with another one, if a merger is possible without a contradiction, i. e. if there is no pair of two different values for one key. If, for example, the first dictionary is [topic=gencs, lecturer=kohlhase], it can be merged with a second dictionary [university=jacobs], yielding the dictionary [topic=gencs, lecturer=kohlhase, university=jacobs]. Merging with [lecturer=kohlhase, semester=2] should also be possible, as the value for the "lecturer" key is the same, but merging with [lecturer=kramer, university=jacobs] should fail, as there are two contradicting values for the "lecturer" key.

Note: You will need the built-in predicate select(Elem, List, Rest), which selects an element Elem from a List (as member(Elem, List) does) leaving a Rest. Use \+ to negate an expression, e. g. when checking whether an entry is not contained in a dictionary.

You can easily access the two components of a pair by pattern matching. The following predicate would, for example, match the second argument with the key of the given pair: key(K=V, K).

Solution:

```
% An empty dictionary can be merged with any other dictionary.
dict_merge([], Dict, Dict).

% As our dictionaries are lists and therefore unordered,
% we have to retrieve the Key=Value pair from the second dictionary
% using the built−in predicate select.

dict_merge([Key=Value|Rest1], Dict, [Key=Value|Rest3]) :−
    % if Key=Value exists in both dictionaries, ...
    select(Key=Value, Dict, Rest2),
    % ... merge the remainders of both.
    dict_merge(Rest1, Rest2, Rest3).

% If the key of the Key=Value pair does not exist in the second
% dictionary, we can add it to the result dictionary.

dict_merge([Key=Value|Rest1], Dict, [Key=Value|Rest2]) :−
    % if Key=something does not exist in the second dictionary, ...
    \+ member(Key=_, Dict),
    % retain this entry and merge the rest of the first
    % dictionary into the second one.
    dict_merge(Rest1, Dict, Rest2).
```

Test cases:

```
dict_merge([lect=kohl,top=gencs],[uni=jac],R).
% R = [lect=kohl, top=gencs, uni=jac]
% Yes
dict_merge([lect=kohl,top=gencs],[lect=kohl,top=gencs],R).
% R = [lect=kohl, top=gencs]
% Yes
dict_merge([lect=kohl,top=gencs],[uni=jac,lect=kram],R).
```

```
% No
```

(Source of this problem: Wilhelm Weisweber, `ProLog` – logisches Programmieren in der Praxis. International Thomson Publishing 1997. ISBN 3-8266-0174-2.)

Assignment 12: Practice tasks

(Given May 8, Due May 21)

25pt

Problem 12.45 (FOL and Turing Machines)

There exists no algorithm that on input $< \Phi >, < \varphi >$ (where $< \Phi >, < \varphi >$ are suitably coded versions of a finite set of FOL formulae and a single formula, respectively) can decide whether Φ entails φ.

How would you go about proving this?

Solution: Reduction to Halting Problem

Problem 12.46 (High Order Functions) 25pt

Given the definition type matrix = int list list;, write a function that multiplies two matrices:

mat_mult : matrix −> matrix −> matrix

A matrix is represented as a list of rows. Each row is a list of integers.

For this task:

- Do not use explicit/indirect recursion. Use the foldl and foldr functions instead
- Do not use function cases/clauses. I.e. each function should be defined for a single case only
- Do not use the if ... then ... else ... language construct

You will need to use some of the standard SML list functions.

Here is a reference: `http://www.smlnj.org/doc/basis/pages/list.html`

Some of these functions you can use directly, for others you will need to write 'List.' in front of the name. E.g. List.nth([5,7,8,9], 2) = 8.

For this task you can assume that the input arguments are correct and you do not need to check for errors.

Solution:

```
(* A solution that does not follow the rules outlined in the statement should
receive a minimal amount of points as this is quite a trivial exercise without these restrictions. *)

type matrix = int list list;

(* Used to generate lists [0 .. n−1] *)
fun generate n = List.tabulate( n, ( fn (a) => a ) );

(* Picks all the cth elements from the rows of matrix m and forms a new row.*)
fun take_elem m (c,p) = (foldr ( fn (a,b) => (List.nth(a,c))::b ) [] m)::p;

(* Transposes the provided matrix m *)
fun transpose m = foldr (take_elem m) [] ( generate (length (hd m)));

(* Returns p + col[c]*row[c] *)
fun mult_add col row (c,p) = p + List.nth(col,c)*List.nth(row,c);

(* Multiplies a row with a column. *)
fun multiply_row_col row (col,p) = ( foldr (mult_add col row) 0 (generate (length col) ) ) :: p;

(* Multiplies a matrix row with a matrix transpose T and return the resulting row. *)
fun multiply_row_mat T (row,p) = ( foldr (multiply_row_col row) [] T) :: p;

(* Multiplies two matrices A and B. *)
fun mat_mult (A:matrix) (B:matrix):matrix = foldr ( multiply_row_mat ( transpose(B) ) ) [] A;

(* Test cases *)
val m_3x3_1 = [[1,2,1],[5,3,~3],[2,~6,1]];
val m_3x3_2 = [[~1,9,~5],[2,4,1],[~4,8,2]];
val m_res_1 = [[~1,25,~1],[13,33,~28],[~18,2,~14]];
val m_3x2_1 = [[5,2],[4,~1],[1,~3]];
val m_res_2 = [[14,~3],[34,16],[~13,7]];
val m_2x3_1 = [[3,~7,~3],[2,9,8]];
val m_res_3 = [[~38,3,21],[63,~17,~17]];

val test1 = mat_mult m_3x3_1 m_3x3_2 = m_res_1;
val test2 = mat_mult m_3x3_1 m_3x2_1 = m_res_2;
val test3 = mat_mult m_2x3_1 m_3x3_1 = m_res_3;
```

Problem 12.47 (Turing Machine) 15pt

Provide the transition table of a TM that goes to the *yes* state if the input provided is of the form $a^n b^n$, $n >= 0$, and *no* if not (a^n stands for the character a repeated n times). You should surround the input by a # on each side, so examples of an inputs which should lead to *yes* are ##, #*ab*#, #*aabb*#, #*aaabbb*#... The head will start at the first #.

Note: The input will contain only a's, b's and the 2 hashes.

Solution: Eliminate repeatedly an a, then a b by replacing them with a #. Go right to search for an a replace it with a hash and go until the end to find a b and replace that with a #. Then go to be beginning and start again. If there is no matching b or there are to many bs then *no* else *yes*.

Problem 12.48 (Hill Climbing) 40pt

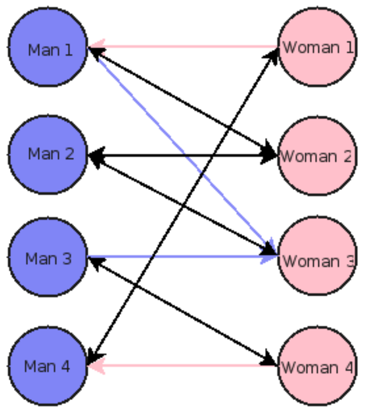
Consider a world with equal number of women and men. Every man is interested in a nonnegative number of women and vice versa. You are given a matrix that specifies a directed graph of *interest* between the people. Write an SML function that uses local search to find a pairing {<man,woman>,<man,woman>,...} such that no man is paired up with > 1 women and vice versa. A pairing is admissible if in every pair <man i, woman j> the two people are interested in each other. An optimal pairing is the pairing with the highest cardinality of all the possible pairings in a problem.

To accomplish this task follow the steps outlined below:

- Define what is a state in this problem
- Given any state, describe what the neighbours of this state are (i.e. describe how neighbours are related). Hint: think about neighbours in the Traveling Salesman Problem
- Find and describe a heuristic. What is the optimal value of your heuristic?
- Write an SML function pairup that takes an interest graph (represented as a matrix) and an initial paring (not necessarily admissible) and uses hill climbing to return an admissible pairing. A sample hill-climbing algorithm is provided in the slides. You may assume that the format of the input matrix is correct

Input: The following matrix encodes the graph below:

	Woman			
Man	$< 0,1 >$	$< 1,1 >$	$< 1,0 >$	$< 0,0 >$
	$< 0,0 >$	$< 1,1 >$	$< 1,1 >$	$< 0,0 >$
	$< 0,0 >$	$< 0,0 >$	$< 1,0 >$	$< 1,0 >$
	$< 1,1 >$	$< 0,0 >$	$< 0,0 >$	$< 0,1 >$



The first value indicates if the man is interested in the woman, while the second value indicates if the woman is interested in the man.

It would be encoded as follows:

```
val matrix = [[(false,true),(true,true),(true,false),(false,false)],[(false,false),(true,true),(true,true),(false,false)],
[(false,false),(false,false),(true,false),(true,true)],[(true,true),(false,false),(false,false),(false,true)]]
```

Use the following datatypes:

```
datatype man = man of int
datatype woman = woman of int
type pairing = (man * woman) list
type matrix = (bool * bool) list list
```

Function signature:

```
val pairup = fn : pairing -> matrix -> pairing
```

Sample run:

```
val matrix = [[(false,true),(true,true),(true,false),(false,false)],[(false,false),(true,true),(true,true),(false,false)],
[(false,false),(false,false),(true,false),(true,true)],[(true,true),(false,false),(false,false),(false,true)]];
val init = [(man 1,woman 2),(man 2,woman 3),(man 3,woman 1),(man 4,woman 4)];
pairup init matrix;
(*Ideally*)
val it = [(man 1,woman 2),(man 2,woman 3),(man 3,woman 4),(man 4,woman 1)] : (man * woman) list
```

Solution:

- A state is a bijective pairing Men->Women
- A neighbour of a state is any state such that 2 men have switched partners, e.g. states {<man 1,woman 1>,<man 2,woman 2>,<man 3,woman 3>} and {<man 1,woman 2>,<man 2,woman 1>,<
  are neighbours
- A heuristic simply counts how many *good pairs* there are, i.e. how many pairs are really interested in each other. An optimal value is n (the number of men/women)
- **exception** LookupException

```
datatype man = man of int;
datatype woman = woman of int;
type pairing = (man * woman) list;
type matrix = (bool * bool) list list;

local
        (* reduce boolean values (a,b) to true if both a and b are true, false otherwise
        fun strip [] = []
                | strip ((bool1,bool2)::tail) = (bool1 andalso bool2)::(strip tail)

        (* reduce the interest matrix to bool list list form: (a,b) -> true iff (a and b) *)
        fun stripall [] = []
                | stripall (list::tail) = (strip list)::(stripall tail)

        (* find member j of the given list *)
        fun lookup_one j [] = raise LookupException
                | lookup_one 1 (h::t) = h
                | lookup_one j (h::t) = lookup_one (j-1) t
```

```
(* find element at position i,j in the matrix *)
fun lookup i j [] = raise LookupException
        | lookup 1 j (h::t) = lookup_one j h
        | lookup i j (h::t) = lookup (i−1) j t

(* evaluate the state using the interest matrix *)
fun heuristic interest [] = 0
        | heuristic interest ((man i, woman j)::tail) = if (lookup i j interest)



(* find which woman is in the i−th couple *)
fun find i [] = raise LookupException
        | find 1 ((man(_), woman j)::t) = j
        | find i (h::t) = find (i−1) t

(* replace i−th woman with woman w *)
fun replace_woman i w [] = raise LookupException
        | replace_woman 1 w ((man m, woman(_))::t) = ((man m, woman w)::t)
        | replace_woman i w (h::t) = h::(replace_woman (i−1) w t)

(* switch the women in the i−th and j−th couples *)
fun switch i j state = let val wi = find i state;
                                val wj = find j state;
                                                                in
                        replace_woman i wj (replace_woman j wi state)
                                                                end

(* finds the highest−value neighbour with the couples i and j swapped s.t. i < j <= n *)
fun high_neigh_small i j n state interest =
     if (j > n) then (state,0)
    else
        let val (best_state,best_heu) = high_neigh_small i (j+1) n state interest;
                        val this_state = switch i j state;
                        val this_heu = heuristic interest this_state;
        in
                 if (this_heu > best_heu) then (this_state,this_heu) else (best_state,best_heu)
        end

(* the highest_neighbour function returns the highests of all of the state's neighbours
(i.e. all states that have exactly one couple switched). All possible pair switchings are
 treated using the high_neigh_small function that computes the best of all neighbours with
 couple i and all j−s s.t. i < j <= n swapped. The high_neigh_small function is run on all 1 <= i < n

fun highest_neighbour i n state interest =
        if (i >= n) then (state,0)
          else
        let val (small_state,small_heu) = high_neigh_small i (i+1) n state interest;
                    val (best_state,best_heu) = highest_neighbour (i+1) n state interest;
        in
         if (small_heu > best_heu) then (small_state,small_heu) else (best_state,best_heu)
```

```
            end

        (* performs the search using highest_neighbour function, stops searching when no neighbour
        has better heuristic than the current state*)

        fun search (state,heu) n interest =
            let val (neigh,neigh_heu) = highest_neighbour 1 n state interest
            in
                if (heu >= neigh_heu) then state else search (neigh,neigh_heu) n interest
            end

        (* only leave the pairs that are interested in each other *)
        fun extract_correct interest [] = []
            | extract_correct interest ((man i,woman j)::t) = if (lookup i j interest)



in
        fun pairup (init:pairing) (interest:matrix) : pairing =
                    let val inter = stripall interest;
                            val n = length inter;
                    in
                            extract_correct inter (search (init,(heuristic inter init)) n inter)
                    end
end
```

Test cases: The test cases below can only be applied to solutions with that define states and neighbours as in the sample solution. All other solutions should be treated on a case by case basis. Sorry :)

```
local
        fun member a [] = false
         | member a (h::t) = if (a = h) then true else member a t

        fun extract a [] = []
         | extract a (h::t) = if (a = h) then t else h::(extract a t)
in
        fun compare_res [] [] = true
         | compare_res (h::t) [] = false
         | compare_res [] (h::t) = false
         | compare_res (h1::t1) l2 = if (member h1 l2) then compare_res t1 (extract h1 l2) else false
end;

(* CHANGE HERE IF THEIR SIGNATURE DOESN'T MATCH!! *)
fun f (init : pairing) (interest : matrix) = pairup init interest;

val matrix1 = [[(true,true),(false,false)],[(false,false),(true,true)]];
val init1 = [(man 1,woman 2),(man 2,woman 1)];
val matrix2 = [[(false,true),(true,true),(true,false),(false,false)],[(false,false),(true,true),(true,true),(false,false)],[
val init2 = [(man 1,woman 2),(man 2,woman 3),(man 3,woman 1),(man 4,woman 4)];
val init3 = [(man 1,woman 1),(man 2,woman 2),(man 3,woman 3),(man 4,woman 4)];
val init4 = [(man 2,woman 1),(man 3,woman 2),(man 3,woman 3),(man 1,woman 4)];
val test1 = compare_res (f init1 matrix1) [(man 1,woman 1),(man 2,woman 2)];
val test2 = compare_res (f init2 matrix2) [(man 2,woman 3),(man 1,woman 2),(man 3,woman 4),(man 4,woma
val test3 = compare_res (f init3 matrix2) [(man 2,woman 3),(man 1,woman 2),(man 3,woman 4),(man 4,woma
val test4 = compare_res (f init4 matrix2) [(man 2,woman 3),(man 3,woman 4),(man 1,woman 2)];
val test5 = compare_res (f [] []) [];
val test6 = compare_res (f [(man 1,woman 1)] [[(true,true)]]) [(man 1,woman 1)];
val test7 = compare_res (f [(man 1,woman 1)] [[(true,false)]]) [];
```

Problem 12.49 (Cartesian and Cascading Functions) 25pt

You have recently discussed in class cascading and cartesian procedures. Convince yourself (i.e. prove) that the two concepts of multi-ary functions are equivalent:

1. Prove or refute that for given sets A, B, and C the Cartesian function space $\mathcal{F}_2 := (A \times B) \to C$ and cascading function space $\mathcal{F}_2' := A \to (B \to C)$ are isomorphic (i.e. there is a bijective mapping $\iota\colon \mathcal{F}_2 \to \mathcal{F}_2'$).
2. Prove or refute that the function application is compatible with this, i.e. for all $a \in A$, $b \in B$, $f \in \mathcal{F}_2$, and $f' \in \mathcal{F}_2'$ we have $f(\langle a, b\rangle) = \iota(f(a))b$ and $f'(a)(b) = \iota^{-1}(f')(\langle a, b\rangle)$.
3. Generalize the above to the n-ary case.

Solution:

1. Let $f \in \mathcal{F}_2$, then we choose $\iota(f) := \{\langle a, \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in f\}\rangle \,|\, a \in A\}$.

   We first have to convince ourselves that $\iota(f) \in \mathcal{F}_2'$: clearly, $\iota(f) \in (A)B \to C$; to see that $\iota(f)$ is a function, we assume that there is an $a \in A$ and functions $g, g' \in B \to C$, such that $\langle a, g\rangle \in \iota(f)$ and $\langle a, g'\rangle \in \iota(f)$, but $g \neq g'$. From the last condition we know that there is an argument $b \in B$, such that $g(b) \neq g'(b) =: c'$. Let $c := g(b)$ in the following. By construction of $\iota(f)$ we have $\langle\langle a, b\rangle, c\rangle \in f$ and $\langle\langle a, b\rangle, c'\rangle \in f$, which contradicts the fact that f is a function.

   ι is clearly total, since there are no restrictions on the construction.

   The next step is to see that ι is injective: for any two functions $f, g \in \mathcal{F}_2$ with $\iota(f) = \iota(g)$ we show that $f = g$. If $f' = g' \in \mathcal{F}_2'$ for $f' := \iota(f)$ and $g' := \iota(g)$, then for all $a \in A$, we must have $f_a' = g_a' \in B \to C$ for $f_a' := f'(a)$ and $g_a' := g'(a)$, and thus for all $b \in B$ we must have $f_a'(b) = g_a'(b) \in C$. Now, by construction we have $f_a' = \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in f\}$ and $g_a' = \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in g\}$, so in particular, for all pairs $\langle a, b\rangle \in A \times B$ we have $f(\langle a, b\rangle) = f_a'(b) = g_a'(b) = g(\langle a, b\rangle)$ and thus $f = g$.

   The final step is to see that ι is surjective: we give ourselves a $f' \in \mathcal{F}_2'$ and show that there is a $f \in \mathcal{F}_2$, such that $\iota(f) = f'$. We choose $f = \iota^{-1}(f') := \{\langle\langle a, b\rangle, c\rangle \,|\, f'(a)(b)\} = c$. So

   $$\begin{array}{rcl} \iota(f) & = & \{\langle a, \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in f\}\rangle \,|\, a \in A\} \\ & = & \{\langle a, \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in \{\langle\langle a, b\rangle, c\rangle \,|\, f'(a)(b) = c\}\}\rangle \,|\, a \in A\} \\ & = & \{\langle a, \{\langle b, c\rangle \,|\, f'(a)(b) = c\}\rangle \,|\, a \in A\} \\ & = & f' \end{array}$$

2. Let $a \in A$, $b \in B$, $f \in \mathcal{F}_2$, and $f' \in \mathcal{F}_2'$, then we have

   $$\begin{array}{rcl} \iota(f)(a)(b) & = & \{\langle a, \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in f\}\rangle \,|\, a \in A\}(a)(b) \\ & = & \{\langle b, c\rangle \,|\, \langle\langle a, b\rangle, c\rangle \in f\}(b) \\ & = & f(\langle a, b\rangle) \end{array}$$

   and

   $$\iota^{-1}(f')(\langle a, b\rangle) = \{\langle\langle a, b\rangle, c\rangle \,|\, f'(a)(b) = c\}(\langle a, b\rangle) = f'(a)(b)$$

3. Proven by induction.
