General Computer Science
320201 GenCS I & II Problems & Solutions

Michael Kohlhase

School of Engineering & Science
Jacobs University, Bremen Germany
`m.kohlhase@jacobs-university.de`

November 24, 2012

Preface

This document contains selected homework and self-study problems for the course General Computer Science I/II held at Jacobs University Bremen[1] in the academic years 2003-2012. It is meant as a supplement to the course notes [Koh11a, Koh11b]. We try to keep the numbering consistent between the documents.

This document contains the solutions to the problems, it should only be used for checking one's own solutions or to learn proper formulations. There is also a version without solutions [Koh11c, Koh11d], which is intended for self-study and practicing the concepts introduced in class.

This document is made available for the students of this course only. It is still a draft, and will develop over the course of the course. It will be developed further in coming academic years.

Acknowledgments: Immanuel Normann, Christoph Lange, Christine Müller, and Vyacheslav Zholudev have acted as lead teaching assistants for the course, have contributed many of the initial problems and organized them consistently. Throughout the time I have tought the course, the teaching assistants (most of them Jacobs University undergraduates; see below) have contributed new problems and sample solutions, have commented on existing problems and refined them.

GenCS Teaching Assistants: The following Jacobs University students have contributed problems while serving as teaching assiatants over the years: Darko Pesikan, Nikolaus Rath, Florian Rabe, Andrei Aiordachioaie, Dimitar Asenov, Alen Stojanov, Felix Schlesinger, Ştefan Anca, Anca Dragan, Vladislav Perelman, Josip Djolonga, Lucia Ambrošová, Flavia Grosan, Christoph Lange, Ankur Modi, Gordan Ristovski, Darko Makreshanski, Teodora Chitiboj, Cristina Stancu-Mara, Alin Iacob, Vladislav Perelman, Victor Savu, Mihai Cotizo Sima, Radu Cimpeanu, Mihai Cîlănaru, Maria Alexandra Alecu, Miroslava Georgieva Slavcheva, Corneliu-Claudiu Prodescu, Flavia Adelina Grosan, Felix Gabriel Mance, Anton Antonov, Alexandra Zayets, Ivaylo Enchev.

[1]International University Bremen until Fall 2006

Contents

0.1 Getting Started with “General Computer Science”

0.1.1 Overview over the Course

This should pose no problems

0.1.2 Administrativa

Neither should the administrativa

0.1.3 Motivation and Introduction

Problem 0.1 (Algorithms)
One of the most essential concepts in computer science is the Algorithm.

- What is the intuition behind the term “algorithm”.
- What determines the quality of an algorithm?
- Give an everyday example of an algorithm.

Solution:

- An algorithm is a series of instructions to control a (computation) process.
- Termination, correctness, performance
- e. g. a recipe

Problem 0.2 (Keywords of General Computer Science)
Our course started with a motivation of ”General Computer Science” where some fundamental notions where introduced. Name three of these fundamental notions and give for each of them a short explanation.

Solution:

- Algorithms are abstract representations of computation instructions
- Data are representations of the objects the computations act on
- Machines are representations of the devices the computations run on

---

Problem 0.3 (Representations)
An essential concept in computer science is the Representation.

- What is the intuition behind the term “representation”?
- Why do we need representations?
- Give an everyday example of a representation.

Solution:

- A representation is the realization of real or abstract persons, objects, circumstances, Events, or emotions in concrete symbols or models. This can be by diverse methods, e.g. visual, aural, or written; as three-dimensional model, or even by dance.
- we should always be aware, whether we are talking about the real thing or a representation of it. Allows us to abstract away from unnecessary details. Easy for computer to operate with
- e.g. graph is a representation of a maze from the lecture notes

---

0.2 Motivation and Introduction

Problem 0.4 (Algorithms)
One of the most essential concepts in computer science is the Algorithm.

- What is the intuition behind the term "algorithm".
- What determines the quality of an algorithm?
- Give an everyday example of an algorithm.

Solution:

- An algorithm is a series of instructions to control a (computation) process.
- Termination, correctness, performance
- e. g. a recipe

Problem 0.5 (Keywords of General Computer Science)
Our course started with a motivation of "General Computer Science" where some fundamental notions where introduced. Name three of these fundamental notions and give for each of them a short explanation.

Solution:

- Algorithms are abstract representations of computation instructions
- Data are representations of the objects the computations act on
- Machines are representations of the devices the computations run on

---

Problem 0.6 (Representations)
An essential concept in computer science is the Representation.

- What is the intuition behind the term "representation"?
- Why do we need representations?
- Give an everyday example of a representation.

Solution:

- A representation is the realization of real or abstract persons, objects, circumstances, Events, or emotions in concrete symbols or models. This can be by diverse methods, e.g. visual, aural, or written; as three-dimensional model, or even by dance.
- we should always be aware, whether we are talking about the real thing or a representation of it. Allows us to abstract away from unnecessary details. Easy for computer to operate with
- e.g. graph is a representation of a maze from the lecture notes

---

1 Representation and Computation

1.1 Elementary Discrete Math

1.1.1 Mathematical Foundations: Natural Numbers

25pt

Problem 1.1 (A wrong induction proof)
What is wrong with the following "proof by induction"?

Theorem: All students of Jacobs University have the same hair color.

Proof: We prove the assertion by induction over the number n of students at Jacobs University.

base case: $n = 1$. If there is only one student at Jacobs University, then the assertion is obviously true.

step case: $n > 1$. We assume that the assertion is true for all sets of n students and show that it holds for sets of $n+1$ students. So let us take a set S of $n+1$ students. As $n > 1$, we can choose students $s \in S$ and $t \in S$ with $s \neq t$ and consider sets $S_s = S \backslash \{s\}$ and $S_t := S \backslash \{t\}$. Clearly, $\#(S_s) = \#(S_t) = n$, so all students in S_s and have the same hair-color by inductive hypothesis, and the same holds for S_t. But $S = S_s \cup S_t$, so any $u \in S$ has the same hair color as the students in $S_s \cap S_t$, which have the same hair color as s and t, and thus all students in S have the same hair color $\square$

Solution:

The problem with the proof is that the inductive step should also cover the case when $n = 1$, which it doesn't. The argument relies on the fact that there intersection of S_s and S_t is non-empty, giving a mediating element that has the same hair color as s and t. But for $n = 1$, $S = \{s, t\}$, and $S_s = \{t\}$, and $S_t = \{s\}$, so $S_s \cap S_t = \emptyset$.

Problem 1.2 (Natural numbers)
Prove or refute that $s(s(o))$ and $s(s(s(o)))$ are unary natural numbers and that their successors are different.

Solution:

Proof: We will prove the statement using the Peano axioms:

P.1 o is a unary natural number (axiom P1)

P.2 $s(o)$ is a unary natural number (axiom P2 and 1.)

P.3 $s(s(o))$ is a unary natural number (axiom P2 and 2.)

P.4 $s(s(s(o)))$ is a unary natural number (axiom P2 and 3.)

P.5 Since $s(s(s(o)))$ is the successor of $s(s(o))$ they are different unary natural numbers (axiom P2)

P.6 Since $s(s(s(o)))$ and $s(s(o))$ are different unary natural numbers their successors are also different (axiom P4 and 5.)

□

Problem 1.3 (Peano's induction axiom)

State Peano's induction axiom and discuss what it can be used for.

Solution: Peano's induction axiom: Every unary natural number possesses property P , if

- the zero has property P and
- the successor of every unary natural number that has property P also possesses property P

Peano's induction axiom is useful to prove that all natural numbers possess some property. In practice we often use the axiom to prove useful equalities that hold for all natural numbers (e.g. binomial theorem, geometric progression).

1.1.2 Naive Set Theory

25pt

Problem 1.4: Let A be a set with n elements (i.e $\#(A) = n$). What is the cardinality of the power set of A, (i.e. what is $\#(\mathcal{P}(A))$)?

Solution: Let $\#(A) = n$, the power set $\mathcal{P}(A) = \{S \mid S \subseteq A\}$ is the set of all the possible subsets of A. The number of possible subsets having $r \leq n$ elements can be given by

$$\binom{n}{r} = \frac{(n)!}{(r)! \cdot ((n-r))!}$$

r takes values from 0 to n, so the total number of subsets of A is

$$\#(\mathcal{P}(A)) = \sum_{r=0}^{n} \binom{n}{r}$$

and we have

$$\#(\mathcal{P}(A)) = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n}$$

Consider,

$$a + b^n = \binom{n}{0} a^n \cdot b^0 + \binom{n}{1} a^{n-1} \cdot b^1 + \binom{n}{2} a^{n-2} \cdot b^2 + \ldots + \binom{n}{n} a^0 \cdot b^n$$

If we choose $a = 1$ and $b = 1$ then $2^n = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{n}$. Combing this with the equation above, we get $\#(\mathcal{P}(A)) = 2^n$.

Thus the cardinality of the power set of A it is 2^n. This is also the number of subsets of a set with n elements.

Solution: We can obtain this result in a simpler way if we consider representing a subset S of a given finite set A with cardinality $n := \#(A)$ under the form of a binary number. First, associate to each element of A an index between 1 and n. Then write an n-bit binary number N_S putting a 1 in the i-th position if the element with index i is included in the set S and a 0 otherwise. In is evident that between the n-bit binary numbers and the elements of the power set $\mathcal{P}(A)$ exists a one-to-one relation (a bijection) and therefore we conclude that the number of elements in $\mathcal{P}(A)$ is equal to that of n-bit representable numbers, that is 2^n.

Solution: The simplest way to obtain this result is by induction on the number n. If $n = 0$, then A is a singleton, wlog. $A = \{a\}$. So $\mathcal{P}(A) = \{\emptyset, A\}$ and $\#(\mathcal{P}(A)) = 2 = 2^1$. For the step case let us assume that $\#(\mathcal{P}(A)) = 2^n$ for all sets A with $\#(A) = n$. We can write any set B with $\#(B) = n + 1$ as $B = A \cup \{c\}$ for some set A with $\#(A) = n$ and $B \backslash A = \{c\}$. Now, each subset C of B can either contain c (then it is of the form $C \cup \{c\}$ for some $D \in \mathcal{P}(A)$) or not (then $C \in \mathcal{P}(A)$). Thus we have $\mathcal{P}(B) = \mathcal{P}(A) \cup \{D \cup \{c\} \mid D \in \mathcal{P}(A)\}$, and hence

$$\#(\mathcal{P}(B)) = \#(\mathcal{P}(A)) + \#(\mathcal{P}(A)) = 2\#(\mathcal{P}(A)) = 2 \cdot 2^n = 2^{n+1}$$

by inductive hypothesis.

Problem 1.5: Let $A := \{5, 23, 7, 17, 6\}$ and $B := \{3, 4, 8, 23\}$. Which of the relations are reflexive, antireflexive, symmetric, antisymmetric, and transitive? 15pt

Note: Please justify the answers.

$$\begin{aligned}
R_1 \subseteq A \times A, R_1 &= \{\langle 23, 7\rangle, \langle 7, 23\rangle, \langle 5, 5\rangle, \langle 17, 6\rangle, \langle 6, 17\rangle\}\\
R_2 \subseteq B \times B, R_2 &= \{\langle 3, 3\rangle, \langle 3, 23\rangle, \langle 4, 4\rangle, \langle 8, 23\rangle, \langle 8, 8\rangle, \langle 3, 4\rangle, \langle 23, 23\rangle, \langle 4, 23\rangle\}\\
R_3 \subseteq B \times B, R_3 &= \{\langle 3, 3\rangle, \langle 3, 23\rangle, \langle 8, 3\rangle, \langle 4, 23\rangle, \langle 8, 4\rangle, \langle 23, 23\rangle\}
\end{aligned}$$

Solution: R_1 is not reflexive since there are not all elements of A are in R_1 as pairs like $\langle a, a\rangle$ where $a \in A$. R_1 is not antireflexive either, because there is one of those pairs present. R_1 is symmetric, because all pairs in R_1 are "turnable", specifically, $\langle 23, 7\rangle$ exists and $\langle 7, 23\rangle$ exists. This holds for all pairs in R_1. Since R_1 is symmetric, it is therefore not antisymmetric. R_1 is also not transitive since there are no pair "triangles".

R_2 is reflexive, it holds all elements of B in pairs like $\langle b, b\rangle$ where $b \in B$. Therefore, it is not antireflexive. R_2 is not symmetric, because for a given pair $\langle a, b\rangle$ where $a, b \in B$ there does not exist a pair $\langle b, a\rangle$. R_2 is, however, antisymmetric since for any "turnable" pair (like $\langle 3, 3\rangle$) the two elements in the pair are equal. Also, R_2 is transitive since such a triangle (the only one in the set) exists. Namely, that is $\langle 3, 23\rangle, \langle 3, 4\rangle and \langle 4, 23\rangle$.

R_3 is neither reflexive nor antireflexive. Also, it is not symmetric or transitive. It is, however, antisymmetric.

Problem 1.6: Given two relations $R \subseteq C \times B$ and $Q \subseteq C \times A$, we define a relation $P \subseteq C \times (B \cap A)$ such that for every $x \in C$ and every $y \in (B \cap A)$, $\langle x, y\rangle \in P \Leftrightarrow \langle x, y\rangle \in R \vee \langle x, y\rangle \in Q$. Prove or refute (by giving a counterexample) the following statement: If Q and P are total functions, then P is a partial function. 20pt

Solution: The statement is false. A counterexample is $C = \{c\}, A = B = \{a, b\}, R = \{\langle c, a\rangle\}, Q = \{\langle c, b\rangle\}$. Then $P = \{\langle c, a\rangle, \langle c, b\rangle\}$ is not a partial function.

1.1.3 Naive Set Theory

Problem 1.7: Fill in the blanks in the table of Greek letters. Note that capitalized names denote capital Greek letters. 3pt 3min

Symbol					γ	Σ	π	Φ
Name	alpha	eta	lambda	iota				

Solution:

Symbol	α	η	λ	ι	γ	Σ	π	Φ
Name	alpha	eta	lambda	iota	gamma	Sigma	pi	Phi

1.1.4 Relations and Functions

Problem 1.8 (Associativity of Relation Composition)

Let R, S, and T be relations on a set M. Prove or refute that the composition operation for relations is associative, i. e. that

$$(T \circ S) \circ R = T \circ (S \circ R)$$

Solution:

Proof:

P.1 Let $\langle x, y\rangle \in ((T \circ S) \circ R)$.

P.2 $\exists z_1 \in M.\langle x, z_1\rangle \in R \wedge \langle z_1, y\rangle \in (T \circ S)$

P.3 $\exists z_1, z_2 \in M.\langle x, z_1\rangle \in R \wedge (\langle z_1, z_2\rangle \in S \wedge \langle z_2, y\rangle \in T)$

P.4 $\exists z_2 \in M.\langle x, z_2\rangle \in (S \circ R) \wedge \langle z_2, y\rangle \in T$

P.5 $\langle x, y\rangle \in (T \circ (S \circ R))$. $\square$

1.2 Computing with Functions over Inductively Defined Sets

1.2.1 Standard ML: Functions as First-Class Objects

Problem 1.9: Define the `member` relation which checks whether an integer is member of a list of integers. The solution should be a function of type `int * int list -> bool`, which evaluates to `true` on arguments `n` and `l`, iff `n` is an element of the list `l`.

Solution: The simplest solution is the following

```
fun member(n,nil) = false
  | member(n,h::r) = if n=h then true else member(n,r);
```

The intuition here is that a is a member of a list l, iff it is the first element, or it is a member of the rest list.

Note that we cannot just use `member(n,n::r)` to eliminate the conditional, since SML does not allow duplicate variables in matching. But we can simplify the conditional after all: we can make use of SML's `orelse` function which acts as a logical "or" and get the slightly more elegant program

```
fun member(n,nil) = false
  | member(n,h::r) = (n=h) orelse member(n,r);
```

Problem 1.10: Define the `subset` relation. Set T is a subset of S iff all elements of T are also elements of S. The empty set is subset of any set.

Hint: Use the member function from ??

Solution: Here we make use of SML's `andalso` operator, which acts as a logical "and"

```
fun subset(nil,_) = true
  | subset(x::xs,m) = member(x,m) andalso subset(xs,m);
```

The intuition here is that $S \subseteq T$, iff for some $s \in S$ we have $s \in T$ and $S \backslash \{s\} \subseteq T$.

Problem 1.11: Define functions to zip and unzip lists. zip will take two lists as input and create pairs of elements, one from each list, as follows: `zip [1,2,3] [0,2,4]` $\leadsto$ `[[1,0],[2,2],[3,4]]`. `unzip` is the inverse function, taking one list of tuples as argument and outputing two separate lists. `unzip [[1,4],[2,5],[3,6]]` $\leadsto$ `[1,2,3] [4,5,6]`. 20pt

Solution: Zipping is relatively simple, we will just define a recursive function by considering 4 cases:

```
fun zip nil nil = nil
  | zip nil l = l
  | zip l nil = l
  | zip (h::t) (k::l) = [h,k]::(zip t l)
```

Unzipping is slightly more difficult. We need map functions that select the first and second elements of a two-element list over the zipped list. Since the problem is somewhat under-specified by the example, we will put the rest of the longer list into the first list. To avoid problems with the empty tails for the shorter list, we use the `mapcan` function that appends the tail lists.

```
fun mapcan(f,nil) = nil | mapcan(f,h::t) = (f h)@(mapcan(f,t))
fun unzip (l) = if (l = nil) then nil
              else [(map head l),(mapcan tail l)]
```

Problem 1.12 (Compressing binary lists)
Define a data type of binary digits. Write a function that takes a list of binary digits and returns an int list that is a compressed version of it and the first binary digit of the list (needed for reconversion). For example,

```
     ZIPit([zero,zero,zero, one,one,one,one,
      zero,zero,zero, one, zero,zero]) -> (0,[3,4,3,1,2]),
```

because the binary list begins with 3 zeros, followed by 4 ones etc.

Solution:

```
datatype bin = zero | one;
local fun ZIP(nil,_,cnt) = [cnt] |
   ZIP(hd::tl, last, cnt) =
      if hd=last then ZIP(tl, hd, cnt+1)
      else cnt::ZIP(tl, hd, 1);
in
   fun ZIPit(hd::tl) = (hd, ZIP(tl, hd, 1))
end;
```

Problem 1.13 (Decompressing binary lists)
Write an inverse function `UNZIPit` of the one written in ??.

Solution:

```
local fun pump(a,0) = nil |
  pump(a,n) = a::pump(a,n-1);
fun not zero = one |
    not one = zero;
in
fun UNZIPit(a,nil) = nil |
    UNZIPit(a, hd::tl) = pump(a,hd)@UNZIPit(not(a),tl);
end;
```

Problem 1.14: Program the function f with $f(x) = x^2$ on unary natural numbers without using the multiplication function. 15pt

Solution: We will use the abstract data type `mynat`

```
datatype mynat = zero | s of mynat
fun add(n,zero) = n | add(n,s(m))=s(add(n,m))
fun sq(zero)=zero|sq(s(n))=s(add(add(sq(n),n),n))
```

Problem 1.15 **(Translating between Integers and Strings)** 20pt

SML has pre-defined types `int` and `string`, write two conversion functions:

- `int2string` converts an integer to a string, i.e. `int2string(~317)` $\leadsto$ `"~317":string`
- `string2int` converts a suitable string to an integer, i.e. `string2int("444")` $\leadsto$ `444:int`. For the moment, we do not care what happens, if the input string is unsuitable, i.e does not correspond to an integer.

do not use any built-in functions except elementary arithmetic (which include `mod` and `div` BTW), `explode`, and `implode`.

Solution:

```
    (* Note: this implementation does not consider negative numbers *)

    (*integer to string*)

    fun dig2chr 0 = #"0" | dig2chr 1 = #"1" |
        dig2chr 2 = #"2" | dig2chr 3 = #"3" |
        dig2chr 4 = #"4" | dig2chr 5 = #"5" |
        dig2chr 6 = #"6" | dig2chr 7 = #"7" |
        dig2chr 8 = #"8" | dig2chr 9 = #"9";

    fun int2lst 0 = [] |
int2lst num = int2lst(num div 10) @ [dig2chr(num mod 10)];

    fun int2string 0 = "0" |
        int2string num = implode(int2lst num);

    (*string to integer*)

    fun chr2dig #"0" = 0 | chr2dig #"1" = 1 |
        chr2dig #"2" = 2 | chr2dig #"3" = 3 |
        chr2dig #"4" = 4 | chr2dig #"5" = 5 |
        chr2dig #"6" = 6 | chr2dig #"7" = 7 |
        chr2dig #"8" = 8 | chr2dig #"9" = 9;

    fun lst2int [] = 0 |
    lst2int (h::t) = (lst2int t + chr2dig h )*10;

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

    fun string2int(s) = lst2int(rev (explode s)) div 10;
```

Problem 1.16: Write a function that takes an odd positive integer and returns a `char list list` which represents a triangle of stars with n stars in the last row. For example,

```
triangle 5;
val it =
[#" ", #" ", #"*", #" ", #" "],
[#" ", #"*", #"*", #"*", #" "],
[#"*", #"*", #"*", #"*", #"*"]]
```

Solution:

```
fun stars(0) = nil |
    stars(n) = #"*" :: stars(n-1)

fun wall(nil) = nil |
    wall(hd::tl) = ((#" "::hd)@[#" "])::wall(tl)

fun triangle(1) = [[#"*"]] |
    triangle(n) = wall(triangle(n-2))@[stars(n)];
```

Problem 1.17: Write a non-recursive variant of the `member` function from ?? using the `foldl` function.

Solution:

```
fun member (x,xs) = foldl (fn (y,b) => b orelse x=y) false
```

Problem 1.18 (Decimal representations as lists) 15pt 10min

The decimal representation of a natural number is the list of its digits (i. e. integers between 0 and 9). Write an SML function `decToInt` of type `int list -> int` that converts the decimal representation of a natural number to the corresponding number:

```
- decToInt [7,8,5,6];
val it = 7856 : int
```

Hint: Use a suitable built-in higher-order list function of type `fn : (int * int -> int) -> int -> int list -> int` that solves a great part of the problem.

Solution:

```
val decToInt = foldl (fn (x,y) => 10*y + x) 0;
```

Problem 1.19 (List functions via `foldl`/`foldr`) 30pt

Write the following procedures using `foldl` or `foldr`

1. `length` which computes the length of a list
2. `concat`, which gets a list of lists and concatenates them to a list.
3. `map`, which maps a function over a list
4. `myfilter`, `myexists`, and `myforall` from ??

Solution:

```
fun length xs = foldl (fn (x,n) => n+1) 0 xs
fun concat xs = foldr op@ nil xs
fun map f = foldr (fn (x,yr) => (f x)::yr) nil
fun myfilter f =
   foldr (fn (x,ys) => if f x then x::ys else ys) nil
fun myexists f = foldl (fn (x,b) => b orelse f x) false
fun myall f = foldl (fn (x,b) => b andalso f x) true
```

Problem 1.20 (Mapping and Appending) 10pt

Can the functions `mapcan` and `mapcan2` be written using `foldl`/`foldr`?

Solution:

```
fun mapcan_with(f,l) = foldl(fn (v,s) => s@f(v)) nil l;
```

1.2.2 Inductively Defined Sets and Computation

Problem 1.21: Figure out the functions on natural numbers for the following defining equations

$$\tau(o) = o$$

$$\tau(s(n)) = s(s(s(\tau(n))))$$

Solution: The function τ triples its argument.

Problem 1.22 (A function on natural numbers) 15pt 5min

Figure out the function on natural numbers defined by the following equations:

$$\eta(o) = o$$

$$\eta(s(o)) = o$$

$$\eta(s(s(n))) = s(\eta(n))$$

Solution:

The function η halves its argument.

Problem 1.23: In class, we have been playing with defining equations for functions on the natural numbers. Give the defining equations for the function σ with $\sigma(x) = x^2$ without using the multiplication function (you may use the addition function though). Prove from the Peano axioms or refute by a counterexample that your equations define a function. Indicate in each step which of the axioms you have used. 15pt

Solution:

Lemma 1 *The relation defined by the equations $\sigma(o) = o$ and $\sigma(s(n)) = +(\langle +(\langle \sigma(n), n\rangle), n\rangle)$ is a function.*

Proof:

P.1 The proof of functionality is is carried out by induction. We show that for every $n \in \mathbb{N}$ sq is one-valued.

P.1.1 $n = o$: Then the value is fixed to o there so we have the assertion.

P.1.2 $n > 0$**: let** σ **is one-valued for** n**.**:

P.1.2.1 By the defining equation we know that $\sigma(s(n)) = +(\langle +(\langle \sigma(n), n\rangle), n\rangle)$

P.1.2.2 By inductive hypothesis, $\sigma(n)$ is one-valued, so $\sigma(s(n))$ is as $+$ is a function.

P.1.2.3 This completes the step case and proves the assertion. □

□

1.2.3 Inductively Defined Sets in SML

Problem 1.24: Declare an SML datatype `pair` representing pairs of integers and define SML functions `fst` and `snd` where `fst` returns the first- and `snd` the second component of q the pair. Moreover write down the type of the constructor of `pair` as well as of the two procedures `fst` and `snd`. 8pt 8min

Use SML syntax for the whole problem.

Solution:

```
datatype pair = pair of int * int; (* val pair = fn : int * int -> pair *)

fun fst(pair(x,_)) = x; (* val fst = fn : pair -> int *)
fun snd(pair(_,y)) = y; (* val snd = fn : pair -> int *)
```

Problem 1.25: Declare a data type `myNat` for unary natural numbers and `NatList` for lists of natural numbers in SML syntax, and define a function that computes the length of a list (as a unary natural number in `mynat`). Furthermore, define a function `nms` that takes two unary natural numbers `n` and `m` and generates a list of length `n` which contains only `m`s, i.e. `nms(s(s(zero)),s(zero))` evaluates to `construct(s(zero),construct(s(zero),elist))`. 4pt 8min

Solution:

```
datatype mynat = zero | s of mynat;
datatype natlist = elist | construct of mynat * natlist;
fun length (nil) = zero | length (construct (n,l)) = s(length(l));
fun nms(zero,m) = elist | nms(s(n),m) = construct(m,nms(n));
```

Problem 1.26: Given the following SML data type for an arithmetic expressions 20pt

```
datatype arithexp = aec of int (* 0,1,2,... *)
                  | aeadd of arithexp * arithexp (* addition *)
                  | aemul of arithexp * arithexp (* multiplication *)
                  | aesub of arithexp * arithexp (* subtraction *)
                  | aediv of arithexp * arithexp (* division *)
                  | aemod of arithexp * arithexp (* modulo *)
                  | aev of int (* variable *)
```

give the representation of the expression $(4x + 5) - 3x$.

Write a (cascading) function `eval : (int -> int) -> arithexp -> int` that takes a variable assignment φ and an arithmetic expresson e and returns its evaluation as a value.

Note: A variable assignment is a function that maps variables to (integer) values, here it is represented as function φ of type `int -> int` that assigns $\varphi(n)$ to the variable `aev(`n`)`.

Solution:

```
datatype arithexp = aec of int (* 0,1,2,... *)
| aeadd of arithexp * arithexp (* addition *)
| aemul of arithexp * arithexp (* multiplication *)
| aesub of arithexp * arithexp (* subtraction *)
| aediv of arithexp * arithexp (* division *)
| aemod of arithexp * arithexp (* modulo *)
| aev of int (* variable *)

(* aesub(aeadd(aemul(aec(4),aev(1)),aec(5)),aemul(aec(3),aev(1))) *)

fun eval phi =
let
    fun calc (aev(x)) = phi(x) |
        calc (aec(x)) = x |
        calc (aeadd(e1,e2)) = calc(e1) + calc(e2) |
        calc (aesub(e1,e2)) = calc(e1) - calc(e2) |
        calc (aemul(e1,e2)) = calc(e1) * calc(e2) |
        calc (aediv(e1,e2)) = calc(e1) div calc(e2) |
        calc (aemod(e1,e2)) = calc(e1) mod calc(e2);
in fn x => calc(x)
end;

(* Test:
- eval (fn 1=>6) (aesub(aeadd(aemul(aec(4),aev(1)),aec(5)),aemul(aec(3),aev(1))));
stdIn:14.7-14.14 Warning: match nonexhaustive
1 => ...

val it = 11 : int
- *)
```

Problem 1.27 (Your own lists)
Define a data type `mylist` of lists of integers with constructors `mycons` and `mynil`. Write translators `tosml` and `tomy` to and from SML lists, respectively.

Solution: The data type declaration is very simple

```
datatype mylist = mynil | mycons of int * mylist;
```

it declares three symbols: the base type `mylist`, the individual constructor `mynil`, and the constructor function `mycons`.

The translator function `tosml` takes a term of type `mylist` and gives back the corresponding SML list; the translator function `tomy` does the opposite.

```
fun tosml mynil = nil
  | tosml mycons(n,l) = n::tosml(l)
fun tomy nil = mynil
  | tomy (h::t) = mycons(h,tomy(t))
```

Problem 1.28 (Unary natural numbers)
Define a `datatype nat` of unary natural numbers and implement the functions

- `add = fn : nat * nat -> nat` (adds two numbers)
- `mul = fn : nat * nat -> nat` (multiplies two numbers)

Solution:

```
datatype nat = zero | s of nat;
fun add(zero:nat,n2:nat) = n2
  | add(n1,zero) = n1
  | add(s(n1),s(n2)) = s(add(n1,s(n2)));
fun mult(zero:nat,_) = zero
  | mult(_,zero) = zero
  | mult(n1,s(zero)) = n1
  | mult(s(zero),n2) = n2
  | mult(n1,s(n2)) = add(n1,mult(n1,n2));
```

Problem 1.29 (*N*ary Multiplication)
By defining a new datatype for n-tuples of unary natural numbers, implement an n-ary multiplications using the function `mul` from ??. For $n = 1$, an n-tuple should be constructed by using a constructor named `first`; for $n > 1$, further elements should be prepended to the first by using a constructor named `next`. The multiplication function `nmul` should return the product of all elements of a given tuple.

For example,

```
nmul(next(s(s(zero)),
       next(s(s(zero)),
         first(s(s(s(zero)))))))
```

should output `s(s(s(s(s(s(s(s(s(s(s(s(zero))))))))))))` since 223 = 12.

Solution:

```
datatype tuple = first of nat | next of nat*tuple;
fun nmult(first(num)) = num |
    nmult(next(num, rest)) = mult(num, nmult(rest));
```

1.2.4 A Theory of SML: Abstract Data Types and Term Languages

5pt
Abstract Data Types and Ground Constructor Terms
5min

Problem 1.30: Translate the abstract data types given in mathematical notation into SML datatypes

1. $\langle\{\mathbb{S}\},\{[c_1\colon\mathbb{S}],[c_2\colon\mathbb{S}\to\mathbb{S}],[c_3\colon\mathbb{S}\times\mathbb{S}\to\mathbb{S}],[c_4\colon\mathbb{S}\to\mathbb{S}\to\mathbb{S}]\}\rangle$
2. $\langle\{\mathbb{T}\},\{[c_1\colon\mathbb{T}],[c_2\colon\mathbb{T}\times(\mathbb{T}\to\mathbb{T})\to\mathbb{T}]\}\rangle$

Solution:

1. `datatype S = c1 | c2 of S | c3 of S * S | c4 of S -> S`
2. `datatype S = c1 | c2 of T * (T -> T)`

Problem 1.31: Translate the given SML datatype

5pt
5min

```
datatype T = 0 | c1 of T * T | c2 of T -> (T * T)
```

into abstract data type in mathmatical notation.

Solution:

$$\langle\{\mathbb{T}\}, \{[c_1 : \mathbb{T}], [c_2 : \mathbb{T} \times \mathbb{T}]\mathbb{T}, [c_2 : \mathbb{T}]\mathbb{T} \times \mathbb{T} \to \mathbb{T}\}\rangle$$

Problem 1.32 (Nested lists) 20pt

In class, we have defined an abstract data type for lists of natural numbers. Using this intuition, construct an abstract data type for lists that contain natural numbers or lists (nested up to arbitrary depth). Give the constructor term (the trace of the construction rules) for the list $[3,4,[7,[8,2],9],122,[2,2]]$.

Solution: We choose the abstract data type

$$\langle\{\mathbb{N},\mathbb{L}\},\{[c_l\colon \mathbb{L}\times\mathbb{L}\to\mathbb{L}],[c_n\colon \mathbb{N}\times\mathbb{L}\to\mathbb{L}],[nil\colon \mathbb{L}],[o\colon \mathbb{N}],[s\colon \mathbb{N}\to\mathbb{N}]\}\rangle$$

The constructors c_l and c_l construct lists by adding a list or a number at the front of the list. With this, the list above has the constructor term.

$$c_n(\underline{3},c_n(\underline{4},c_l(c_n(\underline{7},c_l(c_n(\underline{8},c_n(\underline{2},nil)),c_n(\underline{9},nil)),c_n(\underline{122}),c_l(c_n(\underline{2},c_n(\underline{2},nil)))nil))))$$

where $\underline{n}$ is the s,o-constructor term of the number n.

A First Abstract Interpreter Problem 1.33: Give the defining equations for the maximum function for two numbers. This function takes two arguments and returns the larger one. 30pt

Hint: You may define auxiliary functions with defining equations of their own. You can use ι from above.

Solution: We first define the equality predicate on natural numbers by the rules

$$eq(o,o) \rightsquigarrow T \qquad eq(s(n_{\mathbb{N}}),o) \rightsquigarrow F \qquad eq(s(n_{\mathbb{N}}),s(m_{\mathbb{N}})) \rightsquigarrow eq(n_{\mathbb{N}},m_{\mathbb{N}})$$

Using this we define a relation of "greater than" by the rules

$$g(o,n_{\mathbb{N}}) \rightsquigarrow F \qquad g(s(n_{\mathbb{N}}),m_{\mathbb{N}}) \rightsquigarrow \vee(eq(s(m_{\mathbb{N}}),n_{\mathbb{N}}),g(n_{\mathbb{N}},m_{\mathbb{N}}))$$

This allows us to finally define the function *max* by the rule

$$max(n_{\mathbb{N}},m_{\mathbb{N}}) \rightsquigarrow \iota(\vee(g(n_{\mathbb{N}},m_{\mathbb{N}}),eq(svarn\mathbb{N},m_{\mathbb{N}})),n_{\mathbb{N}},m_{\mathbb{N}})$$

Problem 1.34: Using the abstract data type of truth functions from ??, give the defining equations for a function ι that takes three arguments, such that $\iota(\varphi_{\mathbb{B}}, a_{\mathbb{N}}, b_{\mathbb{N}})$ behaves like "if φ then a, else b", where a and b are natural numbers. 15pt

Solution: The defining equations are $\iota(T, a_{\mathbb{N}}, b_{\mathbb{N}}) \rightsquigarrow a_{\mathbb{N}}$ and $\iota(F, a\mathbb{N}, b_{\mathbb{N}}) \rightsquigarrow b_{\mathbb{N}}$.

Problem 1.35: Consider the following abstract data type: 6pt

$$\mathcal{A} := \langle\{\mathbb{A},\mathbb{B},\mathbb{C}\},\{[f\colon \mathbb{C}\to\mathbb{B}],[g\colon \mathbb{A}\times\mathbb{B}\to\mathbb{C}],[h\colon \mathbb{C}\to\mathbb{A}],[a\colon \mathbb{A}],[b\colon \mathbb{B}],[c\colon \mathbb{C}]\}\rangle$$

Which of the following expressions are constructor terms (with variables), which ones are ground. Give the sorts for the terms.

Answer with **Y**es or **N**o or /. and give the sort (if term)			
expression	term?	ground?	Sort
$f(g(a))$			
$f(g(\langle a,b\rangle))$			
$h(g(\langle h(x_{\mathbb{C}}),f(c)\rangle))$			
$h(g(\langle h(x_{\mathbb{B}}),f(y_{\mathbb{C}})\rangle))$			

Solution:

expression	term?	ground?	Sort
$f(g(a))$	N	/	/
$f(g(\langle a,b\rangle))$	Y	Y	$\mathbb{B}$
$h(g(\langle h(x_{\mathbb{C}}),f(c)\rangle))$	Y	N	$\mathbb{A}$
$h(g(\langle h(x_{\mathbb{B}}),f(y_{\mathbb{C}})\rangle))$	N	/	/

Problem 1.36 (Substitution)

Apply the substitutions $\sigma := [b/x], [(g(a))/y], [a/w]$ and $\tau := [(h(c))/x], [c/z]$ to the terms $s := f(g(x, g(a, x, b), y))$ and $t := g(x, x, h(y))$ (give the 4 result terms $\sigma(s)$, $\sigma(t)$, $\tau(s)$, and $\tau(t)$).

Substitutions
5pt
5min

Solution:

$$\begin{array}{rclrcl} \sigma(s) & = & f(g(a, f(b), g(a, a, b))) & \sigma(t) & = & g(a, f(b), h(a)) \\ \tau(s) & = & f(g(f(b), y, g(a, f(b), b))) & \tau(t) & = & g(f(b), y, h(c)) \end{array}$$

Definition 2 We call a substitution σ **idempotent**, iff $\sigma(\sigma(\mathbf{A})) = \sigma(\mathbf{A})$ for all terms $\mathbf{A}$.

Definition 3 For a substitution $\sigma = [\mathbf{A}_1/x_1], \cdots, [\mathbf{A}_n/x_n]$, we call the set $\mathbf{intro}(\sigma) := \bigcup_{1 \leq i \leq n} \mathbf{free}(\mathbf{A}_i)$ the set of variables **introduced** by σ, and the set $\mathbf{supp}(\sigma) := \{x_i \mid 1 \leq i \leq n\}$

30pt

Problem 1.37: Prove or refute that σ is idempotent, if $\mathbf{intro}(\sigma) \cap \mathbf{supp}(\sigma) = \emptyset$.

Problem 1.38 (Substitution Application) 30pt

Consider the following SML data type of terms:

```
datatype term = const of string
              | var of string
              | pair of term * term
              | appl of string * term
```

Constants and variables are represented by a constructor taking their name string, whereas applications of the form $f(t)$ are constructed from the name string and the argument. Remember that we use $f(a,b)$ as an abbreviation for $f(\langle a,b\rangle)$. Thus a term $f(a,g(x))$ is represented as `appl("f",pair(const("a"), appl("g", var("x"))))`.

With this, we can represent substitutions as lists of elementary substitutions, which are pairs of type `term * string`. Thus we can set

```
type subst = term * string list
```

and represent a substitution $\sigma = [(f(a))/x], [b/y]$ as `[(appl("f", const("a")), "x"), (const("b"), "y")]`. Of course we may not allow ambiguous substitutions which contain duplicate strings.

Write an SML function `substApply` for the substitution application operation, i.e. `substApply` takes a substitution σ and a term $\mathbf{A}$ as arguments and returns the term $\sigma(\mathbf{A})$ if σ is unambiguous and raises an exception otherwise.

Make sure that your function applies substitutions in a parallel way, i.e. that $[y/x], [x/z](f(z)) = f(x)$.

Solution:

```
exception ambiguous_substitution

local
   fun sa(s,const(str)) = const(str)
     | sa(s,pair(t1,t2)) = pair(sa(s,t1),sa(s,t2))
     | sa(s,appl(fs,t1)) = appl(fs,sa(s,t1))
     | sa(nil,var(str)) = var(str)
     | sa((t,x)::L,var(str)) = if str = x then t else sa(L,var(str))
   fun ambiguous = ...
in
  fun substApply (s,t) = if ambiguous(s)
                         then raise ambiguous_substitution
                         else sa(s,t)
end
```

or

```
(* (C) by Anna Michalska *)

datatype term = const of string
| var of string
| pair of term * term
| appl of string * term;
type subst = (term * string) list;

exception ania;

fun comparing1 ((x1,x2), []) = true | comparing1 ((x1,x2), hd::tl) = if
hd=x2 then false else comparing1 ((x1,x2),tl);

fun comparing2([],_)=true | comparing2 ((x3,x4)::t,tl) = if (comparing1
((x3,x4),tl)) then comparing2 (t,x4::tl) else raise ania;

fun tab (a,[]) = var(a)
| tab (a, (a1,a2)::tl) = if (a=a2) then a1 else tab(a,tl);

fun substApply_r (appl(a,b),subst_in) = appl(a,substApply_r(b,subst_in))
   |substApply_r (pair(a,b),subst_in) =
   pair(substApply_r(a,subst_in),substApply_r(b,subst_in))
   |substApply_r (var(a),subst_in) = tab(a,subst_in)
   |substApply_r (const(x),subst_in) = const(x);
```

```
fun substApply (subst_in,term_in) =
    if (comparing2(subst_in,[])) then substApply_r(term_in,subst_in)
    else raise ania;
```

A Second Abstract Interpreter Problem 1.39: Consider the following abstract procedure on the abstract data type of natural numbers: 20pt

$$\mathcal{P} := \langle f{::}\mathbb{N} \to \mathbb{N}\,;\, \{f(o) \rightsquigarrow o, f(s(o)) \rightsquigarrow o, f(s(s(n_{\mathbb{N}}))) \rightsquigarrow s(f(n_{\mathbb{N}}))\}\rangle$$

1. Show the computation process for $\mathcal{P}$ on the arguments $s(s(s(o)))$ and $s(s(s(s(s(s(o))))))$.
2. Give the recursion relation of $\mathcal{P}$.
3. Does $\mathcal{P}$ terminate on all inputs?
4. What function is computed by $\mathcal{P}$?

Solution:

1. $f(s(s(s(o)))) \rightsquigarrow s(f(s(o))) \rightsquigarrow s(o)$, and $f(s(s(s(s(s(s(o))))))) \rightsquigarrow s(f(s(s(s(s(o)))))) \rightsquigarrow s(s(f(s(s(o))))) \rightsquigarrow s(s(s(f(o)))) \rightsquigarrow s($
2. The recursion relation is $\{\langle s(s(n)), n\rangle \in (\mathbb{N} \times \mathbb{N}) \mid n \in \mathbb{N}\}$ (or $\langle n+2, n\rangle$)
3. the abstract procedure terminates on all inputs.
4. the abstract procedure computes the function $f\colon \mathbb{N} \to \mathbb{N}$ with $2n \mapsto n$ and $2n - 1 \mapsto n$.

Problem 1.40: Explain the concept of a "call-by-value" programming language in terms of evaluation order. Give an example program where this effects evaluation and termination, explain it.

1pt **Evaluation Order and Termination** 10min

Note: One point each for the definition, the program and the explanation.

Solution: A "call-by-value" programming language is one, where the arguments are all evaluated before the defining equations for the function are applied. As a consequence, an argument that contains a non-terminating call will be evaluated, even if the function ultimately disregards it. For instance, evaluation of the last line does not terminate.

```
fun myif (true,A,_) = A | myif (false,_,B) = B
fun bomb (n) = bomb(n+1)
myif(true,1,bomb(1))
```

Problem 1.41: Give an example of an abstract procedure that diverges on all arguments, and another one that terminates on some and diverges on others, each example with a short explanation. 2pt 3min

Solution: The abstract procedure $\langle f{::}\mathbb{N} \to \mathbb{N}\,;\, \{f(n_{\mathbb{N}}) \rightsquigarrow s(f(n_{\mathbb{N}}))\}\rangle$ diverges everywhere. The abstract procedure $\langle f{::}\mathbb{N} \to \mathbb{N}\,;\, \{f(s(s(n_{\mathbb{N}}))) \rightsquigarrow n_{\mathbb{N}}, f(s(o)) \rightsquigarrow f(s(o))\}\rangle$ terminates on all odd numbers and diverges on all even numbers.

Problem 1.42: Give the recursion relation of the abstract procedures in ??, ??, ??, and ?? and discuss termination. 15pt

Solution:

??: $\{\langle s(n), n\rangle \mid n \in \mathbb{N}\}$

??: all recursion relations are empty

??: the recursion relation is empty

??: the recursion relation for g is $\{\langle s(n), n\rangle \mid n \in \mathbb{N}\}$, the one for max is empty

1.2.5 More SML: Recursion in the Real World

No problems supplied yet.

1.2.6 Even more SML: Exceptions and State in SML

Problem 1.43 (Integer Intervals) 5pt 10min

Declare an SML data type for natural numbers and one for lists of natural numbers in SML. Write an SML function that given two natural number n and m (as a constructor term) creates the list `[n,n+1,\ldots,m-1,m]` if $n \leq m$ and raises an exception otherwise.

Solution:

```
datatype nat = z | s of nat;
datatype lnat = nil | c of nat*lnat;

exception Bad;

(* cmp(a,b) returns 1 if a>b, 0 if a=b, and ~1 if a<b *)
fun cmp(z,z) = 0 |
     cmp(s(_),z) = 1 |
     cmp(z,s(_)) = ~1 |
     cmp(s(n),s(m)) = cmp(n,m);

fun makelist(n, m) =
    case cmp(n, m) of
          ~1 => c(n, makelist(s(n),m)) |
           0 => c(m, nil) |
           1 => raise Bad
```

Problem 1.44 (Operations with Exceptions)
Add to the functions from ?? functions for subtraction and division that raise exceptions where necessary.

- `function sub: nat*nat -> nat` (subtracts two numbers)
- `function div: nat*nat -> nat` (divides two numbers)

Solution:

```
exception Underflow;
datatype nat = zero | s of nat;
fun sub(n1:nat,zero) = n1
  | sub(zero,s(n2)) = raise Underflow
  | sub(s(n1),s(n2)) = sub(n1,n2);
```

Problem 1.45 (List Functions with Exceptions) 6pt 20min

Write three SML functions `nth`, `take`, `drop` that take a list and an integer as arguments, such that

1. `nth(xs,n)` gives the `n`-th element of the list `xs`.
2. `take(xs,n)` returns the list of the first `n` elements of the list `xs`.
3. `drop(xs,n)` returns the list that is obtained from `xs` by deleting the first `n` elements.

In all cases, the functions should raise the exception `Subscript`, if $n < 0$ or the list `xs` has less than `n` elements. We assume that list elements are numbered beginning with 0.

Solution:

```
exception Subscript
fun nth (nil,_) = raise Subscript
  | nth (h::t,n) = if n < 0 then raise Subscript
                   else if n=0 then h else nth(t,n-1)
fun take (l,0) = nil
  | take (nil,_) = raise Subscript
  | take (h::t,n) = if n < 0 then raise Subscript
                    else h::take(t,n-1)
fun drop (l,0) = l
  | drop (nil,_) = raise Subscript
  | drop (h::t,n) = if n < 0 then raise Subscript
                    else drop(t,n-1)
```

Problem 1.46 (Transformations with Errors) 10pt

Extend the function from ?? by an error flag, i.e. the value of the function should be a pair consisting of a string, and the boolean value `true`, if the string was suitable, and `false` if it was not.

Solution: [1]

---

[1] EdNote: need one; please help

Problem 1.47 (Simple SML data conversion) 10pt

Write an SML function `char_to_int = fn : char -> int` that given a single character in the range $[0-9]$ returns the corresponding integer. Do not use the built-in function `Int.fromString` but do the character parsing yourself. If the supplied character does not represent a valid digit raise an `InvalidDigit` exception. The exception should have one parameter that contains the invalid character, i.e. it is defined as `exception InvalidDigit of char`

Solution:

```
exception InvalidDigit of char;

(* Converts a character representing a digit to an integer *)
fun char_to_int c =
  let
    val res = (ord c) - (ord #"0");
  in
    if res >= 0 andalso res <= 9 then res else raise InvalidDigit(c)
  end;

(* TEST CASES *)
val test1 = char_to_int #"0" = 0;
val test2 = char_to_int #"3" = 3;
val test3 = char_to_int #"9" = 9;
val test4 = char_to_int #"~" = 6 handle InvalidDigit c => true | other => false;
val test5 = char_to_int #"a" = 6 handle InvalidDigit c => true | other => false;
val test6 = char_to_int #"Z" = 6 handle InvalidDigit c => true | other => false;
```

Problem 1.48 (Strings and numbers) 10pt

Write two SML functions

1. `str_to_int = fn : string -> int`
2. `str_to_real = fn : string -> real`

that given a string convert it to an integer or a real respectively. Do not use the built-in functions `Int.fromString, Real.fromString` but do the string parsing yourself.

- Negative numbers begin with a `'~'` character (not '-').
- If the string does not represent a valid integer raise an exception as in the previous exercise. Use the same definition and indicate which character is invalid.
- If the input string is empty raise an exception.
- Examples of valid inputs for the second function are: `~1, ~1.5, 4.63, 0.0, 0, .123`

Solution:

```
(* Converts a list of characters to an integer. The list must be reversed and
  there should be only digit characetrs (no minus). *)
fun inv_pos_charl_to_int nil = 0
  | inv_pos_charl_to_int (a::l) = char_to_int a + 10*inv_pos_charl_to_int(l);

(* Converts a list of characters to a positive or a negative integer. *)
fun charl_to_int (#"~"::l) = ~( inv_pos_charl_to_int(rev l))
  | charl_to_int l = inv_pos_charl_to_int(rev(l));

(* Converts a string to a negative or a positive integer *)
fun str_to_int s = charl_to_int(explode(s));

(* TEST CASES *)
val test1 = str_to_int "0" = 0;
val test2 = str_to_int "1" = 1;
val test3 = str_to_int "234" = 234;
val test4 = str_to_int "~0" = 0;
val test5 = str_to_int "~4" = ~4;
val test6 = str_to_int "~5734" = ~5734;
val test7 = str_to_int "hello" = 6 handle InvalidDigit c => true| other => false;
val test8 = str_to_int "~13.2" = 6 handle InvalidDigit c => true| other => false;
```

Solution:

```
exception NegativeFraction;

(* Splits a character list into two lists delimited by a '.' character *)
fun cl_get_num_parts nil whole _ = (whole,nil)
  | cl_get_num_parts (#"."::l) whole fract = (whole, l)
  | cl_get_num_parts (c::l) whole fract = cl_get_num_parts l (whole @ [c] ) fract;

(* Given a real number makes it into a fraction by dividing by 10 until the
  input is less than 1 *)
fun make_fraction fr =
  if fr < 1.0 then fr else make_fraction (fr / 10.0);

(* Converts a string to a real number. Only decimal dot notation is allowed *)
fun str_to_real s =
  let
    val (w,f) = cl_get_num_parts (explode s) nil nil;
    val is_negative = (length w > 0) andalso (hd w = #"~");
    val whole_r = real( str_to_int (implode w) );
    val fract = real ( str_to_int (implode f) );
    val fract_r = if fract < 0.0
                  then raise NegativeFraction
                  else make_fraction fract;
  in
```

```
    if is_negative then whole_r - fract_r else whole_r + fract_r
  end;

(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;

val test1 = eq ( str_to_real "0") 0.0;
val test2 = eq ( str_to_real "0.156") 0.156;
val test3 = eq ( str_to_real "14.723") 14.723;
val test4 = eq ( str_to_real "~0.123") ~0.123;
val test5 = eq ( str_to_real "~12.789") ~12.789;
val test6 = eq ( str_to_real ".123") 0.123;
val test7 = eq ( str_to_real "hello") 4.2 handle InvalidDigit c => true| other => false;
val test8 = eq ( str_to_real "~13..2") 4.2 handle InvalidDigit c => true| other => false;
val test9 = eq ( str_to_real "~13.~2") 4.2 handle NegativeFraction => true| other => false;
```

Problem 1.49 (Recursive evaluation) 10pt

Write an SML function `evaluate = fn : expression -> real` that takes an expression of the following datatype and computes its value:

```
datatype expression = add of expression*expression (* add *)
                  | sub of expression*expression (* subtract *)
                  | dvd of expression*expression (* divide *)
                  | mul of expression*expression (* multiply *)
                  | num of real;
```

For example we have

```
evaluate(num(1.3)) -> 1.3
evaluate(div(num(2.2),num(1.0))) -> 2.2
evaluate(add(num(4.2),sub(mul(num(2.1),num(2.0)),num(1.4)))) -> 7.0
```

Solution:

```
datatype expression = add of expression*expression (* add *)
                  | sub of expression*expression (* subtract *)
                  | dvd of expression*expression (* divide *)
                  | mul of expression*expression (* multiply *)
                  | num of real;

(* Evaluates an arithmetic expression to a real value *)
fun evaluate (add(x,y)) = (evaluate x) + (evaluate y)
  | evaluate (sub(x,y)) = (evaluate x) - (evaluate y)
  | evaluate (dvd(x,y)) = (evaluate x) / (evaluate y)
  | evaluate (mul(x,y)) = (evaluate x) * (evaluate y)
  | evaluate (num(x)) = x;

(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;

val test1 = eq ( evaluate (num(0.0)) ) 0.0;
val test2 = eq ( evaluate (num(1.23)) ) 1.23;
val test3 = eq ( evaluate (num(~2.78)) ) ~2.78;
val test4 = eq ( evaluate (add(num(1.52),num(~1.78))) ) ~0.26;
val test5 = eq ( evaluate (sub(num(1.52),num(~1.78))) ) 3.3;
val test6 = eq ( evaluate (mul(num(1.5),num(~3.2))) ) ~4.8;
val test7 = eq ( evaluate (dvd(num(3.2),num(~0.5))) ) ~6.4;
val test8 = eq ( evaluate (add(add(add(num(1.0),num(1.0)),num(1.0)),num(1.0)))) 4.0;
val test9 = eq ( evaluate (add(mul(add(num(2.0),num(1.0)), sub(num(9.0),
               mul(num(2.0),add(num(1.0),num(2.0))))),dvd(mul(num(2.0),
               num(4.0)),dvd(add(num(1.0),num(1.0)),num(~4.0)))))) ~7.0;
```

Problem 1.50 (List evaluation) 10pt

Write a new function `evaluate_list = fn : expression list -> real list` that evaluates a list of expressions and returns a list with the corresponding results. Extend the `expression` datatype from the previous exercise by the additional constructor: `var of int`.

The variables here are the final results of previously evaluated expressions. I.e. the first expression from the list should not contain any variables. The second can contain the term `var(0)` which should evaluate to the result from the first expression and so on ... If an expression contains an invalid variable term raise: `exception InvalidVariable of int` that indicates what identifier was used for the variable.

For example we have

```
evaluate_list [num(3.0), num(2.5), mul(var(0),var(1))] -> [3.0,2.5,7.5]
```

Solution:

```
exception InvalidVariable of int;

datatype expression = add of expression*expression (* add *)
                    | sub of expression*expression (* subtract *)
                    | dvd of expression*expression (* divide *)
                    | mul of expression*expression (* multiply *)
                    | num of real
                    | var of int;

(* Evaluates an arithmetic expression to a real value *)
fun evaluate vars (add(x,y)) = (evaluate vars x) + (evaluate vars y)
  | evaluate vars (sub(x,y)) = (evaluate vars x) - (evaluate vars y)
  | evaluate vars (dvd(x,y)) = (evaluate vars x) / (evaluate vars y)
  | evaluate vars (mul(x,y)) = (evaluate vars x) * (evaluate vars y)
  | evaluate _ (num(x)) = x
  | evaluate vars (var(v)) = if v < 0 orelse v>= length vars
                             then raise InvalidVariable(v)
                             else List.nth(vars, v);

fun evaluate_list_helper nil vars = vars
  | evaluate_list_helper (a::l) vars =
      let
        val res = evaluate vars a;
      in
        evaluate_list_helper l (vars @ [res ])
      end;

fun evaluate_list l = evaluate_list_helper l nil;
```

Solution:

```
(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;
fun eql nil nil = true
  | eql l nil = false
  | eql nil l = false
  | eql (a::l) (b::m) = (eq a b) andalso (eql l m);

val test1 = eql ( evaluate_list [num(1.0)] ) [1.0];
val test2 = eql ( evaluate_list [num(1.0),num(~2.3)] ) [1.0,~2.3];
val test3 = eql ( evaluate_list [num(1.0),num(~2.3),add(var(0),var(1))] )
                            [1.0,~2.3,~1.3];
val test4 = eql ( evaluate_list [add(num(1.0),num(4.2)),
                             mul(num(~2.0),sub(num(2.0),num(~5.0))),
                             add(var(0),mul(var(1),num(~1.0)))] )
                             [5.2,~14.0,19.2];

val test5 = eql ( evaluate_list [var(~1)] ) [1.0]
                handle InvalidVariable v => true| other => false;
val test6 = eql ( evaluate_list [var(0)] ) [1.0]
                handle InvalidVariable v => true| other => false;
```

```
val test7 = eql ( evaluate_list [var(1)] ) [1.0]
                handle InvalidVariable v => true| other => false;
val test8 = eql ( evaluate_list [num(1.0),var(1)] ) [1.0]
                handle InvalidVariable v => true| other => false;
```

Problem 1.51 (String parsing) 10pt

Write an SML function `evaluate_str = fn : string list -> real list` that given a list of arithmetic expressions represented as strings returns their values. The strings follow the following conventions:

- strict bracketing: every expression consists of 2 operands joined by an operator and has to be enclosed in brackets, i.e. $1 + 2 + 3$ would be represented as `((1+2)+3)` (or `(1+(2+3))`)
- no spaces: the string contains no empty characters

The value of each of the expressions is stored in a variable named *vn* with *n* the position of the expression in the list. These variables can be used in subsequent expressions.

Raise an exception `InvalidSyntax` if any of the strings does not follow the conventions.

For example we have

```
evaluate_str ["((4*.5)-(1+2.5))"] -> [~1.5]
evaluate_str ["((4*.5)-(1+2.5))","(v0*~2)"] -> [~1.5,3.0]
evaluate_str ["(1.8/2)","(1-~3)","(v0+v1)"] -> [0.9,4.0,4.9]
```

Solution:

```
exception InvalidSyntax;

fun parserest [] n = raise InvalidSyntax
  | parserest [#")"] 0 = []
  | parserest (#"("::t) n = #"("::(parserest t (n+1))
  | parserest (#")"::t) n = #")"::(parserest t (n-1))
  | parserest (h::t) n = h::(parserest t n);

fun findop [] n left = raise InvalidSyntax
  | findop (#"+"::t) 0 left = (#"+",left,(parserest t 0))
  | findop (#"-"::t) 0 left = (#"-",left,(parserest t 0))
  | findop (#"*"::t) 0 left = (#"*",left,(parserest t 0))
  | findop (#"/"::t) 0 left = (#"/",left,(parserest t 0))
  | findop (#"("::t) n left = findop t (n+1) (left@[#"("])
  | findop (#")"::t) n left = findop t (n-1) (left@[#")"])
  | findop (h::t) n left = findop t n (left@[h]);

fun charl_to_exp [] = raise InvalidSyntax
  | charl_to_exp (#"("::t) =
      let val (c,x,y) = findop t 0 [];
      in
        if (c = #"+") then add(charl_to_exp x,charl_to_exp y)
        else if (c = #"-") then sub(charl_to_exp x,charl_to_exp y)
        else if (c = #"*") then mul(charl_to_exp x,charl_to_exp y)
        else dvd(charl_to_exp x,charl_to_exp y)
      end
  | charl_to_exp (#"v"::t) = var(str_to_int (implode t))
  | charl_to_exp (h::t) = num(str_to_real (implode(h::t)));

fun str_to_exp str = charl_to_exp (explode str);

fun evaluate_str l = evaluate_list ( map str_to_exp l);
```

Solution:

```
(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;
fun eql nil nil = true
  | eql l nil = false
  | eql nil l = false
  | eql (a::l) (b::m) = (eq a b) andalso (eql l m);

val test1 = eql (evaluate_str ["0"] ) [0.0];
val test2 = eql (evaluate_str ["1.5"] ) [1.5];
val test3 = eql (evaluate_str [".5"] ) [0.5];
val test4 = eql (evaluate_str ["~1.2"] ) [~1.2];
val test5 = eql (evaluate_str ["(1+3)"] ) [4.0];
val test6 = eql (evaluate_str ["(1.2+3.5)"] ) [4.7];
val test7 = eql (evaluate_str ["(1.2+~3.5)"] ) [~2.3];
val test8 = eql (evaluate_str ["(1.2-~3.5)"] ) [4.7];
val test9 = eql (evaluate_str ["(~1.5+3.2)"] ) [1.7];
val test10 = eql (evaluate_str ["(~1.5*~3.2)"] ) [4.8];
val test11 = eql (evaluate_str ["(5.5/~1.1)"] ) [~5.0];
val test12 = eql (evaluate_str ["(~1.5/3.0)"] ) [~0.5];
val test13 = eql (evaluate_str
      ["(((6.4/~1.6)-7)+((.50-~10)*(20/(2.5/0.5))))"] ) [31.0];
val test14 = eql (evaluate_str ["42.5","v0"] ) [42.5,42.5];
val test15 = eql (evaluate_str
      ["~2","(v0*v0)","(v1*v0)","(v2*v0)"] ) [~2.0,4.0,~8.0,16.0];
val test16 = eql (evaluate_str
      ["~2","(v0*v0)","(v1*(v0+(~2.5/v0)))"] ) [~2.0,4.0,~3.0];
val test17 = eql (evaluate_str ["(((1+2)*3)"] ) [42.5] handle all => true;
val test18 = eql (evaluate_str ["((1+2)3)"] ) [42.5] handle all => true;
val test19 = eql (evaluate_str ["(13"] ) [42.5] handle all => true;
val test20 = eql (evaluate_str ["(((1+2)*3)"] ) [42.5] handle all => true;
val test21 = eql (evaluate_str ["*3)"] ) [42.5] handle all => true;
val test22 = eql (evaluate_str ["(*3)"] ) [42.5] handle all => true;
val test23 = eql (evaluate_str ["(7/3*2)"] ) [42.5] handle all => true;
val test24 = eql (evaluate_str ["((7/3)*(2#6))"] ) [42.5] handle all => true;
val test25 = eql (evaluate_str ["(3-6))"] ) [42.5] handle all => true;
val test26 = eql (evaluate_str ["v0"] ) [42.5] handle all => true;
val test27 = eql (evaluate_str ["0","v1"] ) [42.5] handle all => true;
```

Problem 1.52 (SML File IO) 10pt

Write an SML function `evaluate_file = fn : string -> string -> unit` that performs file IO operations. The first argument is an input file name and the second is an output file name. The input file contains lines which are arithmetic expressions. `evaluate_file` reads all the expressions, evaluates them, and writes the corresponding results to the output file, one result per line.

For example we have

```
evaluate_list "input.txt" "output.txt";

Contents of input.txt:
4.9
0.7
(v0/v1)

Contents of output.txt (after evaluate_list is executed):
4.9
0.7
7.0
```

Solution:

```
fun get_lines istream =
  let
    val line = TextIO.inputLine (istream);
  in
    case line of
         NONE => nil
      | SOME(l) => let
                    val cl = explode l;
                    val cl = List.take(cl, length cl - 1);
                    val l = implode cl;
                  in
                    (l :: (get_lines istream) )
                  end
  end;

fun write_lines nil ostream = true
  | write_lines ((s:real)::l) ostream =
  let
    val _ = TextIO.output (ostream, Real.toString(s));
               val _ = TextIO.output (ostream, "\textbackslash{n}");
  in
    write_lines l ostream
  end;

fun evaluate_file in_filename out_filename =
  let
    val input = TextIO.openIn in_filename;
    val output = TextIO.openOut out_filename;
    val l = evaluate_str ( get_lines input );
               val _ = write_lines l output;
  in
    (TextIO.closeIn input; TextIO.closeOut output)
  end;
```

1.3 A Theory of SML: Abstract Data Types and Term Languages

1.3.1 Abstract Data Types and Ground Constructor Terms

Problem 1.53: Translate the abstract data types given in mathematical notation into SML datatypes 5pt 5min

1. $\langle\{\mathbb{S}\},\{[c_1\colon\mathbb{S}],[c_2\colon\mathbb{S}\to\mathbb{S}],[c_3\colon\mathbb{S}\times\mathbb{S}\to\mathbb{S}],[c_4\colon\mathbb{S}\to\mathbb{S}\to\mathbb{S}]\}\rangle$
2. $\langle\{\mathbb{T}\},\{[c_1\colon\mathbb{T}],[c_2\colon\mathbb{T}\times(\mathbb{T}\to\mathbb{T})\to\mathbb{T}]\}\rangle$

Solution:

1. `datatype S = c1 | c2 of S | c3 of S * S | c4 of S -> S`
2. `datatype S = c1 | c2 of T * (T -> T)`

Problem 1.54: Translate the given SML datatype

5pt
5min

```
datatype T = 0 | c1 of T * T | c2 of T -> (T * T)
```

into abstract data type in mathmatical notation.

Solution:

$$\langle\{\mathbb{T}\}, \{[c_1\colon\mathbb{T}], [c_2\colon\mathbb{T}\times\mathbb{T}]\mathbb{T}, [c_2\colon\mathbb{T}]\mathbb{T}\times\mathbb{T}\to\mathbb{T}\}\rangle$$

Problem 1.55 (Nested lists) 20pt

In class, we have defined an abstract data type for lists of natural numbers. Using this intuition, construct an abstract data type for lists that contain natural numbers or lists (nested up to arbitrary depth). Give the constructor term (the trace of the construction rules) for the list $[3, 4, [7, [8, 2], 9], 122, [2, 2]]$.

Solution: We choose the abstract data type

$$\langle\{\mathbb{N}, \mathbb{L}\}, \{[c_l \colon \mathbb{L} \times \mathbb{L} \to \mathbb{L}], [c_n \colon \mathbb{N} \times \mathbb{L} \to \mathbb{L}], [nil \colon \mathbb{L}], [o \colon \mathbb{N}], [s \colon \mathbb{N} \to \mathbb{N}]\}\rangle$$

The constructors c_l and c_l construct lists by adding a list or a number at the front of the list. With this, the list above has the constructor term.

$$c_n(\underline{3}, c_n(\underline{4}, c_l(c_n(\underline{7}, c_l(c_n(\underline{8}, c_n(\underline{2}, nil)), c_n(\underline{9}, nil)), c_n(\underline{122}), c_l(c_n(\underline{2}, c_n(\underline{2}, nil)))nil))))$$

where $\underline{n}$ is the s, o-constructor term of the number n.

1.3.2 A First Abstract Interpreter

30pt

Problem 1.56: Give the defining equations for the maximum function for two numbers. This function takes two arguments and returns the larger one.

Hint: You may define auxiliary functions with defining equations of their own. You can use ι from above.

Solution: We first define the equality predicate on natural numbers by the rules

$$eq(o,o) \rightsquigarrow T \qquad eq(s(n_{\mathbb{N}}),o) \rightsquigarrow F \qquad eq(s(n_{\mathbb{N}}),s(m_{\mathbb{N}})) \rightsquigarrow eq(n_{\mathbb{N}},m_{\mathbb{N}})$$

Using this we define a relation of "greater than" by the rules

$$g(o,n_{\mathbb{N}}) \rightsquigarrow F \qquad g(s(n_{\mathbb{N}}),m_{\mathbb{N}}) \rightsquigarrow \vee(eq(s(m_{\mathbb{N}}),n_{\mathbb{N}}),g(n_{\mathbb{N}},m_{\mathbb{N}}))$$

This allows us to finally define the function *max* by the rule

$$max(n_{\mathbb{N}},m_{\mathbb{N}}) \rightsquigarrow \iota(\vee(g(n_{\mathbb{N}},m_{\mathbb{N}}),eq(svarn\mathbb{N},m_{\mathbb{N}})),n_{\mathbb{N}},m_{\mathbb{N}})$$

Problem 1.57: Using the abstract data type of truth functions from ??, give the defining equations for a function ι that takes three arguments, such that $\iota(\varphi_{\mathbb{B}}, a_{\mathbb{N}}, b_{\mathbb{N}})$ behaves like "if φ then a, else b", where a and b are natural numbers. 15pt

Solution: The defining equations are $\iota(T, a_{\mathbb{N}}, b_{\mathbb{N}}) \rightsquigarrow a_{\mathbb{N}}$ and $\iota(F, a\mathbb{N}, b_{\mathbb{N}}) \rightsquigarrow b_{\mathbb{N}}$.

Problem 1.58: Consider the following abstract data type: 6pt

$$\mathcal{A} := \langle\{\mathbb{A},\mathbb{B},\mathbb{C}\},\{[f\colon \mathbb{C}\to\mathbb{B}],[g\colon \mathbb{A}\times\mathbb{B}\to\mathbb{C}],[h\colon \mathbb{C}\to\mathbb{A}],[a\colon \mathbb{A}],[b\colon \mathbb{B}],[c\colon \mathbb{C}]\}\rangle$$

Which of the following expressions are constructor terms (with variables), which ones are ground. Give the sorts for the terms.

Answer with **Y**es or **N**o or /. and give the sort (if term)			
expression	term?	ground?	Sort
$f(g(a))$			
$f(g(\langle a,b\rangle))$			
$h(g(\langle h(x_{\mathbb{C}}),f(c)\rangle))$			
$h(g(\langle h(x_{\mathbb{B}}),f(y_{\mathbb{C}})\rangle))$			

Solution:

expression	term?	ground?	Sort
$f(g(a))$	N	/	/
$f(g(\langle a,b\rangle))$	Y	Y	$\mathbb{B}$
$h(g(\langle h(x_{\mathbb{C}}),f(c)\rangle))$	Y	N	$\mathbb{A}$
$h(g(\langle h(x_{\mathbb{B}}),f(y_{\mathbb{C}})\rangle))$	N	/	/

1.3.3 Substitutions

Problem 1.59 (Substitution) 4pt 5min

Apply the substitutions $\sigma := [b/x], [(g(a))/y], [a/w]$ and $\tau := [(h(c))/x], [c/z]$ to the terms $s := f(g(x, g(a, x, b), y))$ and $t := g(x, x, h(y))$ (give the 4 result terms $\sigma(s)$, $\sigma(t)$, $\tau(s)$, and $\tau(t)$).

Solution:

$$\begin{array}{rclcrcl} \sigma(s) & = & f(g(a, f(b), g(a, a, b))) & & \sigma(t) & = & g(a, f(b), h(a)) \\ \tau(s) & = & f(g(f(b), y, g(a, f(b), b))) & & \tau(t) & = & g(f(b), y, h(c)) \end{array}$$

Definition 4 We call a substitution σ **idempotent**, iff $\sigma(\sigma(\mathbf{A})) = \sigma(\mathbf{A})$ for all terms $\mathbf{A}$.

Definition 5 For a substitution $\sigma = [\mathbf{A}_1/x_1], \cdots, [\mathbf{A}_n/x_n]$, we call the set $\mathbf{intro}(\sigma) := \bigcup_{1 \leq i \leq n} \mathbf{free}(\mathbf{A}_i)$ the set of variables **introduced** by σ, and the set $\mathbf{supp}(\sigma) := \{x_i \mid 1 \leq i \leq n\}$

30pt

Problem 1.60: Prove or refute that σ is idempotent, if $\mathbf{intro}(\sigma) \cap \mathbf{supp}(\sigma) = \emptyset$.

Problem 1.61 (Substitution Application) 30pt

Consider the following SML data type of terms:

```
datatype term = const of string
              | var of string
              | pair of term * term
              | appl of string * term
```

Constants and variables are represented by a constructor taking their name string, whereas applications of the form $f(t)$ are constructed from the name string and the argument. Remember that we use $f(a, b)$ as an abbreviation for $f(\langle a, b\rangle)$. Thus a term $f(a, g(x))$ is represented as `appl("f",pair(const("a"), appl("g", var("x"))))`.

With this, we can represent substitutions as lists of elementary substitutions, which are pairs of type `term * string`. Thus we can set

```
type subst = term * string list
```

and represent a substitution $\sigma = [(f(a))/x], [b/y]$ as `[(appl("f", const("a")), "x"), (const("b"), "y")]`. Of course we may not allow ambiguous substitutions which contain duplicate strings.

Write an SML function `substApply` for the substitution application operation, i.e. `substApply` takes a substitution σ and a term $\mathbf{A}$ as arguments and returns the term $\sigma(\mathbf{A})$ if σ is unambiguous and raises an exception otherwise.

Make sure that your function applies substitutions in a parallel way, i.e. that $[y/x], [x/z](f(z)) = f(x)$.

Solution:

```
exception ambiguous_substitution

local
   fun sa(s,const(str)) = const(str)
     | sa(s,pair(t1,t2)) = pair(sa(s,t1),sa(s,t2))
     | sa(s,appl(fs,t1)) = appl(fs,sa(s,t1))
     | sa(nil,var(str)) = var(str)
     | sa((t,x)::L,var(str)) = if str = x then t else sa(L,var(str))
   fun ambiguous = ...
in
  fun substApply (s,t) = if ambiguous(s)
                         then raise ambiguous_substitution
                         else sa(s,t)
end
```

or

```
(* (C) by Anna Michalska *)

datatype term = const of string
| var of string
| pair of term * term
| appl of string * term;
type subst = (term * string) list;

exception ania;

fun comparing1 ((x1,x2), []) = true | comparing1 ((x1,x2), hd::tl) = if
hd=x2 then false else comparing1 ((x1,x2),tl);

fun comparing2([],_)=true | comparing2 ((x3,x4)::t,tl) = if (comparing1
((x3,x4),tl)) then comparing2 (t,x4::tl) else raise ania;

fun tab (a,[]) = var(a)
| tab (a, (a1,a2)::tl) = if (a=a2) then a1 else tab(a,tl);

fun substApply_r (appl(a,b),subst_in) = appl(a,substApply_r(b,subst_in))
   |substApply_r (pair(a,b),subst_in) =
   pair(substApply_r(a,subst_in),substApply_r(b,subst_in))
   |substApply_r (var(a),subst_in) = tab(a,subst_in)
   |substApply_r (const(x),subst_in) = const(x);
```

```
fun substApply (subst_in,term_in) =
    if (comparing2(subst_in,[])) then substApply_r(term_in,subst_in)
    else raise ania;
```

1.3.4 A Second Abstract Interpreter

20pt

Problem 1.62: Consider the following abstract procedure on the abstract data type of natural numbers:

$$\mathcal{P} := \langle f{::}\mathbb{N} \to \mathbb{N}\,;\, \{f(o) \rightsquigarrow o, f(s(o)) \rightsquigarrow o, f(s(s(n_\mathbb{N}))) \rightsquigarrow s(f(n_\mathbb{N}))\}\rangle$$

1. Show the computation process for $\mathcal{P}$ on the arguments $s(s(s(o)))$ and $s(s(s(s(s(s(o))))))$.
2. Give the recursion relation of $\mathcal{P}$.
3. Does $\mathcal{P}$ terminate on all inputs?
4. What function is computed by $\mathcal{P}$?

Solution:

1. $f(s(s(s(o)))) \rightsquigarrow s(f(s(o))) \rightsquigarrow s(o)$, and $f(s(s(s(s(s(s(o))))))) \rightsquigarrow s(f(s(s(s(s(o)))))) \rightsquigarrow s(s(f(s(s(o))))) \rightsquigarrow s(s(s(f(o)))) \rightsquigarrow s($
2. The recursion relation is $\{\langle s(s(n)), n\rangle \in (\mathbb{N} \times \mathbb{N}) \mid n \in \mathbb{N}\}$ (or $\langle n+2, n\rangle$)
3. the abstract procedure terminates on all inputs.
4. the abstract procedure computes the function $f\colon \mathbb{N} \to \mathbb{N}$ with $2n \mapsto n$ and $2n-1 \mapsto n$.

1.3.5 Evaluation Order and Termination

4pt
10min

Problem 1.63: Explain the concept of a "call-by-value" programming language in terms of evaluation order. Give an example program where this effects evaluation and termination, explain it.

Note: One point each for the definition, the program and the explanation.

Solution: A "call-by-value" programming language is one, where the arguments are all evaluated before the defining equations for the function are applied. As a consequence, an argument that contains a non-terminating call will be evaluated, even if the function ultimately disregards it. For instance, evaluation of the last line does not terminate.

```
fun myif (true,A,_) = A | myif (false,_,B) = B
fun bomb (n) = bomb(n+1)
myif(true,1,bomb(1))
```

Problem 1.64: Give an example of an abstract procedure that diverges on all arguments, and another one that terminates on some and diverges on others, each example with a short explanation. 2pt 5min

Solution: The abstract procedure $\langle f{::}\mathbb{N}\to\mathbb{N}\,;\,\{f(n_{\mathbb{N}})\leadsto s(f(n_{\mathbb{N}}))\}\rangle$ diverges everywhere. The abstract procedure $\langle f{::}\mathbb{N}\to\mathbb{N}\,;\,\{f(s(s(n_{\mathbb{N}})))\leadsto n_{\mathbb{N}}, f(s(o))\leadsto f(s(o))\}\rangle$ terminates on all odd numbers and diverges on all even numbers.

Problem 1.65: Give the recursion relation of the abstract procedures in ??, ??, ??, and ?? and discuss termination. 15pt

Solution:

??: $\{\langle s(n), n\rangle \mid n \in \mathbb{N}\}$

??: all recursion relations are empty

??: the recursion relation is empty

??: the recursion relation for g is $\{\langle s(n), n\rangle \mid n \in \mathbb{N}\}$, the one for max is empty

1.4 More SML

1.4.1 More SML: Recursion in the Real World

No problems supplied yet.

1.4.2 Programming with Effects: Imperative Features in SML

Input and Output nothing here yet.

5pt
Even more SML: Exceptions and State in SML
10min

Problem 1.66 (Integer Intervals)
Declare an SML data type for natural numbers and one for lists of natural numbers in SML. Write an SML function that given two natural number n and m (as a constructor term) creates the list `[n,n+1,\ldots,m-1,m]` if $n \leq m$ and raises an exception otherwise.

Solution:

```
datatype nat = z | s of nat;
datatype lnat = nil | c of nat*lnat;

exception Bad;

(* cmp(a,b) returns 1 if a>b, 0 if a=b, and ~1 if a<b *)
fun cmp(z,z) = 0 |
     cmp(s(_),z) = 1 |
     cmp(z,s(_)) = ~1 |
     cmp(s(n),s(m)) = cmp(n,m);

fun makelist(n, m) =
    case cmp(n, m) of
          ~1 => c(n, makelist(s(n),m)) |
           0 => c(m, nil) |
           1 => raise Bad
```

Problem 1.67 (Operations with Exceptions)
Add to the functions from ?? functions for subtraction and division that raise exceptions where necessary.

- `function sub: nat*nat -> nat` (subtracts two numbers)
- `function div: nat*nat -> nat` (divides two numbers)

Solution:

```
exception Underflow;
datatype nat = zero | s of nat;
fun sub(n1:nat,zero) = n1
  | sub(zero,s(n2)) = raise Underflow
  | sub(s(n1),s(n2)) = sub(n1,n2);
```

Problem 1.68 (List Functions with Exceptions) 6pt 20min

Write three SML functions `nth`, `take`, `drop` that take a list and an integer as arguments, such that

1. `nth(xs,n)` gives the `n`-th element of the list `xs`.
2. `take(xs,n)` returns the list of the first `n` elements of the list `xs`.
3. `drop(xs,n)` returns the list that is obtained from `xs` by deleting the first `n` elements.

In all cases, the functions should raise the exception `Subscript`, if $n < 0$ or the list `xs` has less than `n` elements. We assume that list elements are numbered beginning with 0.

Solution:

```
exception Subscript
fun nth (nil,_) = raise Subscript
  | nth (h::t,n) = if n < 0 then raise Subscript
                   else if n=0 then h else nth(t,n-1)
fun take (l,0) = nil
  | take (nil,_) = raise Subscript
  | take (h::t,n) = if n < 0 then raise Subscript
                   else h::take(t,n-1)
fun drop (l,0) = l
  | drop (nil,_) = raise Subscript
  | drop (h::t,n) = if n < 0 then raise Subscript
                   else drop(t,n-1)
```

Problem 1.69 (Transformations with Errors) 10pt

Extend the function from ?? by an error flag, i.e. the value of the function should be a pair consisting of a string, and the boolean value `true`, if the string was suitable, and `false` if it was not.

Solution: [2]

---

[2] EdNote: need one; please help

Problem 1.70 (Simple SML data conversion) 10pt

Write an SML function `char_to_int = fn : char -> int` that given a single character in the range $[0-9]$ returns the corresponding integer. Do not use the built-in function `Int.fromString` but do the character parsing yourself. If the supplied character does not represent a valid digit raise an `InvalidDigit` exception. The exception should have one parameter that contains the invalid character, i.e. it is defined as `exception InvalidDigit of char`

Solution:

```
exception InvalidDigit of char;

(* Converts a character representing a digit to an integer *)
fun char_to_int c =
  let
    val res = (ord c) - (ord #"0");
  in
    if res >= 0 andalso res <= 9 then res else raise InvalidDigit(c)
  end;

(* TEST CASES *)
val test1 = char_to_int #"0" = 0;
val test2 = char_to_int #"3" = 3;
val test3 = char_to_int #"9" = 9;
val test4 = char_to_int #"~" = 6 handle InvalidDigit c => true | other => false;
val test5 = char_to_int #"a" = 6 handle InvalidDigit c => true | other => false;
val test6 = char_to_int #"Z" = 6 handle InvalidDigit c => true | other => false;
```

Problem 1.71 (Strings and numbers) 10pt

Write two SML functions

1. `str_to_int = fn : string -> int`
2. `str_to_real = fn : string -> real`

that given a string convert it to an integer or a real respectively. Do not use the built-in functions `Int.fromString, Real.fromString` but do the string parsing yourself.

- Negative numbers begin with a `'~'` character (not '-').
- If the string does not represent a valid integer raise an exception as in the previous exercise. Use the same definition and indicate which character is invalid.
- If the input string is empty raise an exception.
- Examples of valid inputs for the second function are: `~1, ~1.5, 4.63, 0.0, 0, .123`

Solution:

```
(* Converts a list of characters to an integer. The list must be reversed and
  there should be only digit characetrs (no minus). *)
fun inv_pos_charl_to_int nil = 0
  | inv_pos_charl_to_int (a::l) = char_to_int a + 10*inv_pos_charl_to_int(l);

(* Converts a list of characters to a positive or a negative integer. *)
fun charl_to_int (#"~"::l) = ~( inv_pos_charl_to_int(rev l))
  | charl_to_int l = inv_pos_charl_to_int(rev(l));

(* Converts a string to a negative or a positive integer *)
fun str_to_int s = charl_to_int(explode(s));

(* TEST CASES *)
val test1 = str_to_int "0" = 0;
val test2 = str_to_int "1" = 1;
val test3 = str_to_int "234" = 234;
val test4 = str_to_int "~0" = 0;
val test5 = str_to_int "~4" = ~4;
val test6 = str_to_int "~5734" = ~5734;
val test7 = str_to_int "hello" = 6 handle InvalidDigit c => true| other => false;
val test8 = str_to_int "~13.2" = 6 handle InvalidDigit c => true| other => false;
```

Solution:

```
exception NegativeFraction;

(* Splits a character list into two lists delimited by a '.' character *)
fun cl_get_num_parts nil whole _ = (whole,nil)
  | cl_get_num_parts (#"."::l) whole fract = (whole, l)
  | cl_get_num_parts (c::l) whole fract = cl_get_num_parts l (whole @ [c] ) fract;

(* Given a real number makes it into a fraction by dividing by 10 until the
  input is less than 1 *)
fun make_fraction fr =
  if fr < 1.0 then fr else make_fraction (fr / 10.0);

(* Converts a string to a real number. Only decimal dot notation is allowed *)
fun str_to_real s =
  let
    val (w,f) = cl_get_num_parts (explode s) nil nil;
    val is_negative = (length w > 0) andalso (hd w = #"~");
    val whole_r = real( str_to_int (implode w) );
    val fract = real ( str_to_int (implode f) );
    val fract_r = if fract < 0.0
                  then raise NegativeFraction
                  else make_fraction fract;
  in
```

```
    if is_negative then whole_r - fract_r else whole_r + fract_r
  end;

(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;

val test1 = eq ( str_to_real "0") 0.0;
val test2 = eq ( str_to_real "0.156") 0.156;
val test3 = eq ( str_to_real "14.723") 14.723;
val test4 = eq ( str_to_real "~0.123") ~0.123;
val test5 = eq ( str_to_real "~12.789") ~12.789;
val test6 = eq ( str_to_real ".123") 0.123;
val test7 = eq ( str_to_real "hello") 4.2 handle InvalidDigit c => true| other => false;
val test8 = eq ( str_to_real "~13..2") 4.2 handle InvalidDigit c => true| other => false;
val test9 = eq ( str_to_real "~13.~2") 4.2 handle NegativeFraction => true| other => false;
```

Problem 1.72 (Recursive evaluation) 10pt

Write an SML function `evaluate = fn : expression -> real` that takes an expression of the following datatype and computes its value:

```
datatype expression = add of expression*expression (* add *)
                  | sub of expression*expression (* subtract *)
                  | dvd of expression*expression (* divide *)
                  | mul of expression*expression (* multiply *)
                  | num of real;
```

For example we have

```
evaluate(num(1.3)) -> 1.3
evaluate(div(num(2.2),num(1.0))) -> 2.2
evaluate(add(num(4.2),sub(mul(num(2.1),num(2.0)),num(1.4)))) -> 7.0
```

Solution:

```
datatype expression = add of expression*expression (* add *)
                  | sub of expression*expression (* subtract *)
                  | dvd of expression*expression (* divide *)
                  | mul of expression*expression (* multiply *)
                  | num of real;

(* Evaluates an arithmetic expression to a real value *)
fun evaluate (add(x,y)) = (evaluate x) + (evaluate y)
  | evaluate (sub(x,y)) = (evaluate x) - (evaluate y)
  | evaluate (dvd(x,y)) = (evaluate x) / (evaluate y)
  | evaluate (mul(x,y)) = (evaluate x) * (evaluate y)
  | evaluate (num(x)) = x;

(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;

val test1 = eq ( evaluate (num(0.0)) ) 0.0;
val test2 = eq ( evaluate (num(1.23)) ) 1.23;
val test3 = eq ( evaluate (num(~2.78)) ) ~2.78;
val test4 = eq ( evaluate (add(num(1.52),num(~1.78))) ) ~0.26;
val test5 = eq ( evaluate (sub(num(1.52),num(~1.78))) ) 3.3;
val test6 = eq ( evaluate (mul(num(1.5),num(~3.2))) ) ~4.8;
val test7 = eq ( evaluate (dvd(num(3.2),num(~0.5))) ) ~6.4;
val test8 = eq ( evaluate (add(add(add(num(1.0),num(1.0)),num(1.0)),num(1.0)))) 4.0;
val test9 = eq ( evaluate (add(mul(add(num(2.0),num(1.0)), sub(num(9.0),
               mul(num(2.0),add(num(1.0),num(2.0))))),dvd(mul(num(2.0),
               num(4.0)),dvd(add(num(1.0),num(1.0)),num(~4.0)))))) ~7.0;
```

Problem 1.73 (List evaluation) 10pt

Write a new function `evaluate_list = fn : expression list -> real list` that evaluates a list of expressions and returns a list with the corresponding results. Extend the `expression` datatype from the previous exercise by the additional constructor: `var of int`.

The variables here are the final results of previously evaluated expressions. I.e. the first expression from the list should not contain any variables. The second can contain the term `var(0)` which should evaluate to the result from the first expression and so on ... If an expression contains an invalid variable term raise: `exception InvalidVariable of int` that indicates what identifier was used for the variable.

For example we have

```
evaluate_list [num(3.0), num(2.5), mul(var(0),var(1))] -> [3.0,2.5,7.5]
```

Solution:

```
exception InvalidVariable of int;

datatype expression = add of expression*expression (* add *)
                    | sub of expression*expression (* subtract *)
                    | dvd of expression*expression (* divide *)
                    | mul of expression*expression (* multiply *)
                    | num of real
                    | var of int;

(* Evaluates an arithmetic expression to a real value *)
fun evaluate vars (add(x,y)) = (evaluate vars x) + (evaluate vars y)
  | evaluate vars (sub(x,y)) = (evaluate vars x) - (evaluate vars y)
  | evaluate vars (dvd(x,y)) = (evaluate vars x) / (evaluate vars y)
  | evaluate vars (mul(x,y)) = (evaluate vars x) * (evaluate vars y)
  | evaluate _ (num(x)) = x
  | evaluate vars (var(v)) = if v < 0 orelse v>= length vars
                          then raise InvalidVariable(v)
                          else List.nth(vars, v);

fun evaluate_list_helper nil vars = vars
  | evaluate_list_helper (a::l) vars =
     let
       val res = evaluate vars a;
     in
       evaluate_list_helper l (vars @ [res ])
     end;

fun evaluate_list l = evaluate_list_helper l nil;
```

Solution:

```
(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;
fun eql nil nil = true
  | eql l nil = false
  | eql nil l = false
  | eql (a::l) (b::m) = (eq a b) andalso (eql l m);

val test1 = eql ( evaluate_list [num(1.0)] ) [1.0];
val test2 = eql ( evaluate_list [num(1.0),num(~2.3)] ) [1.0,~2.3];
val test3 = eql ( evaluate_list [num(1.0),num(~2.3),add(var(0),var(1))] )
                            [1.0,~2.3,~1.3];
val test4 = eql ( evaluate_list [add(num(1.0),num(4.2)),
                             mul(num(~2.0),sub(num(2.0),num(~5.0))),
                             add(var(0),mul(var(1),num(~1.0)))] )
                             [5.2,~14.0,19.2];

val test5 = eql ( evaluate_list [var(~1)] ) [1.0]
                handle InvalidVariable v => true| other => false;
val test6 = eql ( evaluate_list [var(0)] ) [1.0]
                handle InvalidVariable v => true| other => false;
```

```
val test7 = eql ( evaluate_list [var(1)] ) [1.0]
                handle InvalidVariable v => true| other => false;
val test8 = eql ( evaluate_list [num(1.0),var(1)] ) [1.0]
                handle InvalidVariable v => true| other => false;
```

Problem 1.74 (String parsing) 10pt

Write an SML function `evaluate_str = fn : string list -> real list` that given a list of arithmetic expressions represented as strings returns their values. The strings follow the following conventions:

- strict bracketing: every expression consists of 2 operands joined by an operator and has to be enclosed in brackets, i.e. $1+2+3$ would be represented as `((1+2)+3)` (or `(1+(2+3))`)
- no spaces: the string contains no empty characters

The value of each of the expressions is stored in a variable named *vn* with *n* the position of the expression in the list. These variables can be used in subsequent expressions.

Raise an exception `InvalidSyntax` if any of the strings does not follow the conventions.

For example we have

```
evaluate_str ["((4*.5)-(1+2.5))"] -> [~1.5]
evaluate_str ["((4*.5)-(1+2.5))","(v0*~2)"] -> [~1.5,3.0]
evaluate_str ["(1.8/2)","(1-~3)","(v0+v1)"] -> [0.9,4.0,4.9]
```

Solution:

```
exception InvalidSyntax;

fun parserest [] n = raise InvalidSyntax
  | parserest [#")"] 0 = []
  | parserest (#"("::t) n = #"("::(parserest t (n+1))
  | parserest (#")"::t) n = #")"::(parserest t (n-1))
  | parserest (h::t) n = h::(parserest t n);

fun findop [] n left = raise InvalidSyntax
  | findop (#"+"::t) 0 left = (#"+",left,(parserest t 0))
  | findop (#"-"::t) 0 left = (#"-",left,(parserest t 0))
  | findop (#"*"::t) 0 left = (#"*",left,(parserest t 0))
  | findop (#"/"::t) 0 left = (#"/",left,(parserest t 0))
  | findop (#"("::t) n left = findop t (n+1) (left@[#"("])
  | findop (#")"::t) n left = findop t (n-1) (left@[#")"])
  | findop (h::t) n left = findop t n (left@[h]);

fun charl_to_exp [] = raise InvalidSyntax
  | charl_to_exp (#"("::t) =
      let val (c,x,y) = findop t 0 [];
      in
        if (c = #"+") then add(charl_to_exp x,charl_to_exp y)
        else if (c = #"-") then sub(charl_to_exp x,charl_to_exp y)
        else if (c = #"*") then mul(charl_to_exp x,charl_to_exp y)
        else dvd(charl_to_exp x,charl_to_exp y)
      end
  | charl_to_exp (#"v"::t) = var(str_to_int (implode t))
  | charl_to_exp (h::t) = num(str_to_real (implode(h::t)));

fun str_to_exp str = charl_to_exp (explode str);

fun evaluate_str l = evaluate_list ( map str_to_exp l);
```

Solution:

```
(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a - b) < EPSILON;
fun eql nil nil = true
  | eql l nil = false
  | eql nil l = false
  | eql (a::l) (b::m) = (eq a b) andalso (eql l m);

val test1 = eql (evaluate_str ["0"] ) [0.0];
val test2 = eql (evaluate_str ["1.5"] ) [1.5];
val test3 = eql (evaluate_str [".5"] ) [0.5];
val test4 = eql (evaluate_str ["~1.2"] ) [~1.2];
val test5 = eql (evaluate_str ["(1+3)"] ) [4.0];
val test6 = eql (evaluate_str ["(1.2+3.5)"] ) [4.7];
val test7 = eql (evaluate_str ["(1.2+~3.5)"] ) [~2.3];
val test8 = eql (evaluate_str ["(1.2-~3.5)"] ) [4.7];
val test9 = eql (evaluate_str ["(~1.5+3.2)"] ) [1.7];
val test10 = eql (evaluate_str ["(~1.5*~3.2)"] ) [4.8];
val test11 = eql (evaluate_str ["(5.5/~1.1)"] ) [~5.0];
val test12 = eql (evaluate_str ["(~1.5/3.0)"] ) [~0.5];
val test13 = eql (evaluate_str
      ["(((6.4/~1.6)-7)+((.50-~10)*(20/(2.5/0.5))))"] ) [31.0];
val test14 = eql (evaluate_str ["42.5","v0"] ) [42.5,42.5];
val test15 = eql (evaluate_str
      ["~2","(v0*v0)","(v1*v0)","(v2*v0)"] ) [~2.0,4.0,~8.0,16.0];
val test16 = eql (evaluate_str
      ["~2","(v0*v0)","(v1*(v0+(~2.5/v0)))"] ) [~2.0,4.0,~3.0];
val test17 = eql (evaluate_str ["(((1+2)*3)"] ) [42.5] handle all => true;
val test18 = eql (evaluate_str ["((1+2)3)"] ) [42.5] handle all => true;
val test19 = eql (evaluate_str ["(13"] ) [42.5] handle all => true;
val test20 = eql (evaluate_str ["(((1+2)*3)"] ) [42.5] handle all => true;
val test21 = eql (evaluate_str ["*3)"] ) [42.5] handle all => true;
val test22 = eql (evaluate_str ["(*3)"] ) [42.5] handle all => true;
val test23 = eql (evaluate_str ["(7/3*2)"] ) [42.5] handle all => true;
val test24 = eql (evaluate_str ["((7/3)*(2#6))"] ) [42.5] handle all => true;
val test25 = eql (evaluate_str ["(3-6))"] ) [42.5] handle all => true;
val test26 = eql (evaluate_str ["v0"] ) [42.5] handle all => true;
val test27 = eql (evaluate_str ["0","v1"] ) [42.5] handle all => true;
```

Problem 1.75 (SML File IO) 10pt

Write an SML function `evaluate_file = fn : string -> string -> unit` that performs file IO operations. The first argument is an input file name and the second is an output file name. The input file contains lines which are arithmetic expressions. `evaluate_file` reads all the expressions, evaluates them, and writes the corresponding results to the output file, one result per line.

For example we have

```
evaluate_list "input.txt" "output.txt";

Contents of input.txt:
4.9
0.7
(v0/v1)

Contents of output.txt (after evaluate_list is executed):
4.9
0.7
7.0
```

Solution:

```
fun get_lines istream =
  let
    val line = TextIO.inputLine (istream);
  in
    case line of
         NONE => nil
       | SOME(l) => let
                      val cl = explode l;
                      val cl = List.take(cl, length cl - 1);
                      val l = implode cl;
                    in
                      (l :: (get_lines istream) )
                    end
  end;

fun write_lines nil ostream = true
  | write_lines ((s:real)::l) ostream =
  let
    val _ = TextIO.output (ostream, Real.toString(s));
                val _ = TextIO.output (ostream, "\textbackslash{n}");
  in
    write_lines l ostream
  end;

fun evaluate_file in_filename out_filename =
  let
    val input = TextIO.openIn in_filename;
    val output = TextIO.openOut out_filename;
    val l = evaluate_str ( get_lines input );
                val _ = write_lines l output;
  in
    (TextIO.closeIn input; TextIO.closeOut output)
  end;
```

1.5 Encoding Programs as Strings

1.5.1 Formal Languages

3pt
5min

Problem 1.76: Given the alphabet $A = \{a, b, c\}$ and a $L := \bigcup_{i=1}^{\infty} L_i$, where $L_1 = \{\epsilon\}$ and L_{i+1} contains the strings x, bbx, xac for all $x \in L_i$.

1. Is L a formal language?
2. Which of the following strings are in L? Justify your answer

$s_1 = bbac$	$s_2 = bbacc$	$s_3 = bbbac$
$s_4 = acac$	$s_5 = bbbacac$	$s_6 = bbacac$

Solution:

1. L is a formal language as $L_1 \in A^+$ and every step from L_i to L_{i+1} concatenates only elements from A.
2. $s_1, s_4, s_6 \in L$[3]

---

[3] EdNote: Need a justification here. Please help

Problem 1.77: Given the alphabet $A = \{a, 2, \S\}$. 2pt

1. Determine $k = \#(Q)$ with $Q = \{s \in A^+ \mid |s| \leq 5\}$.
2. Is Q a formal language over A? Justify your results.

Solution:

$$\begin{aligned} k &= \#(\{s \in A^+ \mid |s| = 0\}) + \\ &\ \#(\{s \in A^+ \mid |s| = 1\}) + \ldots + \\ &\ \#(\{s \in A^+ \mid |s| = 5\}) \\ &= 1 + 3^1 + \ldots + 3^5 = 364 \end{aligned}$$

Q is a formal language over A, since $Q \subseteq A^+$.

Problem 1.78: Let $A := \{\mathtt{a}, \mathtt{h}, /, \#, \mathtt{x}\}$ and $\prec$ be the ordering relation on A with $\mathtt{x} \prec \# \prec / \prec \mathtt{h} \prec \mathtt{a}$. Order the following strings in A^* in the lexical order $<_{\text{lex}}$ induced by $\prec$. 3pt 5min

$s_1 = \#\#\#\#$	$s_2 = \#\#\mathtt{x}\#\#\mathtt{h}$	$s_3 = \epsilon$
$s_4 = \#\#\mathtt{h}\#\#\mathtt{x}$	$s_5 = \mathtt{a}\#\#\#\mathtt{a}\#$	$s_6 = \#\#\#\#/$

Problem 1.79 (Lexical Ordering) 20pt

Write a lexical ordering function `lex` on lists in SML, such that `lex` takes three arguments, an ordering relation (i.e. a binary function from list elements to Booleans), and two lists (representing strings over an arbitrary alphabet). Then `lex(o,l,r)` compares lists `l` and `r` in the lexical ordering induced by the character ordering `o`.

We want the function `lex` to return three value strings `"l<r"`, `"r<l"`, and `"l=r"` with the obvious meanings.

Solution:

```
fun lex (ts, nil, nil) = "l=r"
  | lex (ts, nil, (_::_)) = "r<l"
  | lex (ts, (_::_), nil) = "l<r"
  | lex (ts, (h::t), (k::l)) =
       if h=k then lex( ts, t, l)
       else if ts(h,k) then "l<r" else "r<l";
```

1.5.2 Elementary Codes

2pt

Problem 1.80: Given the alphabets $A = \{\mathtt{a}, \mathtt{2}\}$ and $B = \{\mathtt{9}, \#, /\}$.

1. Is c with $c(a) = \#\#$ and $c(2) = \mathtt{9}\#\#\#/$ a character code?
2. Is the extension of c on strings over A a code?

Solution: c is a character code, since $c\colon A \to B$ and $c(\mathtt{a}) \neq c(\mathtt{2})$, so c is injective. Furthermore c is a prefix code, so the extension of c is a code.

Problem 1.81 (Testing for prefix codes) 30pt

Write an SML function `prefix_code` that tests whether a code is a prefix code. The code is given as a list of pairs (SML type `char*string list`).

Example:

```
prefix_code [(#"a","0"), (#"b","1")];
val it = true : bool
```

Hint: You have to test for functionhood, injectivity and the prefix property.

Solution:

```
infix mem (* list membership *)
fun x mem nil = false | x mem (y::l) = (x=y) orelse (x mem l)
(* test for repeated elements in list *)
fun repeat nil = false
  | repeat (h::t) = h mem t orelse repeat(t)
fun function rel = not (repeat (map (fn (x,_) => x ) rel))
fun injective rel = not (repeat (map (fn (_,x) => x ) rel))
(* test whether a list is a prefix of another *)
fun prefix_list _ nil = false
  | prefix_list nil _ = true
  | prefix_list (h::t) (k::l) = if (h = k) then prefix_list t l else false;
(* testing if there is an element with property p in list *)
fun exists p nil = false | exists p (h::t) = p h orelse exists p t;
(* testing for the prefix property *)
fun prefix_prop code =
     exists (fn (_,c) =>
            exists (fn (_,d) =>
                   prefix_list (explode c) (explode d))
                  code)
            code;
(* putting it all together *)
fun prefix_code code = function code
    andalso injective code
    andalso prefix_prop code = false;


(*Test cases:*)
val test1 = prefix_code [(#"a","0"), (#"b","10")] = true;
val test2 = prefix_code [(#"a","0"), (#"b","1")] = true;
val test3 = prefix_code [(#"a","0"), (#"b","10"), (#"c", "110")]=true;
val test4 = prefix_code [(#"a","0"), (#"a","10")]=false;
val test5 = prefix_code [(#"a", "0"), (#"b", "01")]=false;
val test6 = prefix_code [(#"a", "10"), (#"b", "101"), (#"c", "01")]=false;
val test7 = prefix_code [(#"a", "10"), (#"b", "11")]=true;
val test8 = prefix_code [(#"a","0")]=true;
val test9 = prefix_code []=true;
```

Problem 1.82: Let $A := \{a, b, c, d, e, f, g, h\}$ and $\mathbb{B} := \{0, 1\}$, and

$c(a)$:=010010010101001	$c(b)$:=010110010101001
$c(c)$:=010011110101001	$c(d)$:=010010011101001
$c(e)$:=010010010110001	$c(f)$:=010010010101101
$c(g)$:=010011110101000	$c(h)$:=011111110101000

Is c a character code? Does it induce a code on strings?

Problem 1.83 (Morse Code Translator) 40pt

Write an SML program that transforms arbitrary strings into Morse Code. Write a translation function from Morse code to regular strings and show on some examples that the translators are inverses.

Hint: The Morse codes are multi-character strings. In the Morse representation of the string, these codes should be separated by space characters. This makes a back-translation possible.

Solution: The first task is to program the character-level translation procedures, to make things simple, we will represent the translation table in a list of pairs and use functions `assoc` and `rassoc` to do the lookup. Note that we have conveniently added the separating blanks to the table. Then translating to Morse code is just a simple call to the `mapcan` function from ??.

```
val table = [(#"A",".- "),(#"B","-... "),\ldots,(#"0","----- ")]
exception Lookup
fun assoc (k,nil) = raise Lookup
  | assoc (k,(key,value)::t) = if key = k then value else assoc(k,t);
fun rassoc (k,nil) = raise Lookup
  | rassoc (k,(value,key)::t) = if key = k then value else rassoc(k,t)
fun morse s = implode(mapcan(fn c => explode(assoc(c,table)),explode(s)));
```

The translation back is more involved, since we cannot just "`explode`" the string into the right pieces (which we call the tokens); we have to compute the tokens first. Armed with this procedure, we can proceed almost like above:

```
fun tok(nil,chars,strings) = implode(chars)::strings
  | tok(h::t,chars,strings) =
      if h = #" "
      then tok(t,nil, implode(chars)::strings)
      else tok(t,h::chars,strings)
fun tokenize(s) = tok(explode(s),nil,nil)
fun demorse s = implode(map (fn s => rassoc(s,table)) (tokenize s));
```

Problem 1.84 (Morse Code again) 20pt

With what you know about codes now, is the Morse Code (without the blank characters as stop symbols) a code on strings? Give a proof for your answer.

Solution: The Morse code is not a code on strings without stop characters: We have $morse(IE) = .. + . = ... = morse(S)$, so the Morse code is not injective.

Problem 1.85 (String Decoder without Stop Characters) 30pt

Write a general string decoder that takes as the first argument a code (in the representation you developed in ??) and decodes strings with respect to this code if possible and raises and exception otherwise.

Solution: The algorithm for decoding works as follows, we find a prefix of the coded string that is a codeword, decode that and add it to the result string and recurse on the rest string.

```
(* drop the first n elements from a list of length >= n *)
exception too_short
fun drop n nil = raise too_short
  | drop 0 l = l
  | drop n (h::t) = drop (n-1) t
exception invalid
(* find a code word in a coded string as a prefix*)
fun find code nil = raise invalid
  | find ((char,cw)::t) coded =
      if prefix_list (explode cw) coded
      then cw
      else decode_one t coded
exception Lookup
fun rassoc (k,nil) = raise Lookup
  | rassoc (k,(value,key)::t) = if key = k then value else rassoc(k,t)
fun decode code nil = nil
  | decode code coded =
      let cw = find code coded (* raises exception if not found *)
      in (rassoc cw code) @ (decode code (drop (length cw) coded))
      end
```

1.5.3 Character Codes in the Real World

No problems supplied yet.

1.5.4 Formal Languages and Meaning

No problems supplied yet.

1.6 Boolean Algebra

1.6.1 Boolean Expressions and their Meaning

15pt

Problem 1.86 (Boolean complements)
Prove or refute that the following is a theorem of Boolean Algebra:

> For all $a, b \in \mathbb{B}$, if both $a + b = 1$ and $a * b = 0$, we obtain $b = \overline{a}$. (That is, any $b \in \mathbb{B}$ has a unique complement, regardless of whether we're considering Boolean sums or products.)

Observation: You are not allowed to use truth tables in this proof. Give a solution that is only based on Boolean Algebra rules and theorems.

Solution: Source: [MM00]
Let $a + b = 1$ and $a * b = 0$. Then:

$$\begin{aligned} \overline{a} &= 1 * \overline{a} = (a + b) * \overline{a} = b * \overline{a} \\ \overline{a} &= 0 + \overline{a} = a * b + \overline{a} = b + \overline{a} \end{aligned}$$

Now we replace $\overline{a}$ in the rightmost term in (2) by the right side of (1), $b * \overline{a}$, and obtain

$$\overline{a} = b + b * \overline{a} = b$$

Problem 1.87: Give a model for C_{bool}, where the following expression are theorems: $a*\overline{a}$, $a+\overline{a}$, $a*a$, $\overline{a+a}$. 10pt

Hint: Give the truth tables for the Boolean functions.

Solution: Let $\mathcal{U} := \mathbb{B}$, and $\mathcal{I}(0) = \mathsf{F}$, $\mathcal{I}(1) = \mathsf{T}$, and

$\mathcal{I}(+)$	T	F
T	F	T
F	T	F

$\mathcal{I}(*)$	T	F
T	T	T
F	T	T

$\mathcal{I}(-)$	
T	F
F	T

With this, we have the truth tables

a	$\overline{a}$	$a*\overline{a}$
T	F	T
F	T	T

a	$\overline{a}$	$a+\overline{a}$
T	F	T
F	T	T

a	$a*a$
T	T
F	T

a	$a+a$	$\overline{a+a}$
T	F	T
F	F	T

which verify that we have indeed found the desired model.

Problem 1.88 (Partial orders in a Boolean algebra) 15pt

For a given boolean algebra with a universe $\mathbb{B}$ and $a, b \in \mathbb{B}$, we define that the relation $a \leq b$ holds iff $a + b = b$. Prove for refute that $\leq$ is a partial order on $\mathbb{B}$.

Note: There are boolean algebras with a universe $\mathbb{B}$ larger than just $\{0, 1\}$. We are not going to consider them in the scope of this lecture, but still try to keep your proof as generic as possible. That is, assume that a, b are *arbitrary* elements of $\mathbb{B}$ instead of just distinguishing the cases $a/b = 0$ and $a/b = 1$.

Solution: Source: Meinel/Mundhenk: Mathematische Grundlagen der Informatik. Teubner, 2000. ISBN 3-519-02949-9.

reflexive: because of idempotence

transitive: let $x \leq y$ and $y \leq z$ for arbitrary $x, y, z \in \mathbb{B}$. Then, $x + y = y$ and $y + z = z$ by definition. We obtain:

$$x + z = x + (y + z) = (x + y) + z = y + z = z$$

i. e. $x \leq z$.

antisymmetric: Let $x \leq y$ and $y \leq x$ for arbitrary $x, y \in \mathbb{B}$. That implies $x + y = y$ and $y + x = x$ by definition, and we obtain:

$$x = y + x = x + y = y$$

Problem 1.89: Given the following SML data types for Boolean formulae and truth values 20pt

```
datatype boolexp = zero | one
                 | plus of boolexp * boolexp
                 | times of boolexp * boolexp
                 | compl of boolexp
                 | var of int
datatype mybool = mytrue | myfalse
```

write a (cascading) evaluation function `eval : (int -> mybool) -> boolexp -> mybool` that takes an assignment φ and a Boolean formula e and returns $\mathcal{I}_\varphi(e)$ as a value.

Solution:

```
fun eval1(_,zero) = myfalse
  | eval1(_,one) = mytrue
  | eval1(f,plus(x,y)) =
    if (eval1(f,x) = mytrue orelse eval1(f,y) = mytrue)
    then mytrue else myfalse
  | eval1(f,times(x,y)) =
    if (eval1(f,x) = mytrue andalso eval1(f,y) = mytrue)
    then mytrue else myfalse
  | eval1(f,compl(x)) = if eval1(f,x) = myfalse
                   then mytrue else myfalse
  | eval1(f,var(n)) = f(n)

fun eval f e = eval1(f,e);
```

Problem 1.90: Given the SML data types from ??, write a simplified version of the function using the built-in truth values in SML, i.e. an evaluation function `evalbib : (int -> bool) -> boolexp -> bool`. This function should not use any `if` constructs. 20pt

Solution:

```
fun evalbib(_,bez) = myfalse
  | evalbib(_,beo) = mytrue
  | evalbib(f,bep(x,y)) = evalbib(f,x) orelse evalbib(f,y)
  | evalbib(f,bet(x,y)) = evalbib(f,x) andalso evalbib(f,y)
  | evalbib(f,bec(x)) = not evalbib(f,x)
  | evalbib(f,bev(n)) = f(n)
```

Problem 1.91 (Parsing boolean expressions) 40pt

Given the following SML data types for Boolean formulae

```
datatype boolexp = bez | beo (* 0 and 1 *)
                 | bep of boolexp * boolexp (* plus *)
                 | bet of boolexp * boolexp (* times *)
                 | bec of boolexp (* complement *)
                 | bev of int (* variables *)
```

write an SML function `beparse : string -> boolexp` that takes a string as input and transforms it into an `boolexp` representation of this formula, if it is in E_{bool} and raises an exception if not.

Note: As there is no ASCII representation for the complement operation we used in the definition in class, we use `-(x)` for the complement of `x` in the input syntax. So the relevant clause in the definition is now:

- $E_{\text{bool}}^{i+1} := \{a, -(a), (a+b), (a*b) \mid a, b \in E_{\text{bool}}^{i}\}$

Hint: For this you will need to write a couple of auxiliary functions, e.g. to convert lists of characters into integers and strings. A main function will have to look at all the characters in turn and decide what to do next.

Solution: We will need some auxiliary functions `take` and `drop` for manipulating lists of characters:

```
exception parse_error;

fun take(nil,n) = if n=0 then nil else raise parse_error
  | take(h::t,n) = if n<0 then raise parse_error
                  else if n=0 then nil else h::take(t,n-1);

fun drop(nil,n) = if n=0 then nil else raise parse_error
  | drop(h::t,n) = if n<0
                    then raise parse_error
                    else if n=0 then(h::t) else drop(t,n-1);
```

furthermore, we need functions to convert lists of characters to integers and strings

```
fun to_int(l) =
    if length(l)=0 then raise parse_error
    else let fun to_int_rev(nil) = 0
              | to_int_rev(h::t) = if h>=(#"0")
                                   then if h<=(#"9")
                                        then to_int_rev(t)*10 + ord(h)-ord(#"0")
                                        else raise parse_error
                                   else raise parse_error
         in to_int_rev(rev l) end;
```

The following function finds out the head symbol of the expression

```
fun find_sign(nil,_,_) = raise parse_error
  | find_sign(h::t,np,pos) =
      if h=(#"(") then find_sign(t,np+1,pos+1)
      else if h=(#")") then find_sign(t,np-1,pos+1)
           else if (h=(#"~") orelse h=(#"+") orelse h=(#"*")) andalso np=0
                then (pos,h)
                else find_sign(t,np,pos+1);
```

With these, we can finally build the main processing function

```
fun process(nil) = raise parse_error
  | process(h::t) =
     case (h) of
          (#"X") => if hd(t)=(#"0") then raise parse_error
                    else bev(str2int(implode(t)))
        | (#"0") => if t=nil then bez else raise parse_error
        | (#"1") => if t=nil then beo else raise parse_error
        | (#"(") =>
           let
             val lst = if hd(rev t)=(#")") then take(t,length(t)-1)
                       else raise parse_error
```

```
            val (p,s)=find_sign(lst,0,1)
            (* we find the next sign to be interpreted, and its position *)
           in
            case (s) of
                 (#"+") => bep(process(take(lst,p-1)),process(drop(lst,p)))
               | (#"*") => bet(process(take(lst,p-1)),process(drop(lst,p)))
               | (#"~") => bec(process(drop(lst,1)))
               |(_) => raise parse_error end (* to surpress the warning *)
       |(_) => raise parse_error;
```

With this, the main parsing function is simply

```
fun beparse(s) = process(explode(s));
```

Problem 1.92: Write a function `beprint : boolexp -> string` that converts `boolexp` formulae from ?? to E_{bool} strings. This should be the inverse function to the function `beparse` from ??. 20pt

Test your implementation by round-tripping (check on some examples whether `beparse(beprint(x))=x` and `beprint(beparse(x))=x`). Exhibit at least three examples with at least 8 operators each, and show the results on them.

Solution: For the inverse function `beprint` we will need a function that converts an integer to a string.

```
fun to_string(v) = if v<0 then "~"^to_string((~1)*v)
                  else if v>9
                       then to_string(v div 10)^to_string(v mod 10)
                       else implode([chr(v+48)]);
```

With this, the print function is a simple recursion over the structure of the object

```
fun beprint(bez) = "0" | beprint(beo) = "1"
  | beprint(bec(e)) = "(~"^beprint(e)^")"
  | beprint(bep(e1,e2)) = "("^beprint(e1)^"+"^beprint(e2)^")"
  | beprint(bet(e1,e2)) = "("^beprint(e1)^"*"^beprint(e2)^")"
  | beprint(bev(v)) = "X"^to_string(v);
```

To test that this is really an inverse, we have

```
- beprint(beparse("((X200+X100)*(X1+X2))"));
val it= "((X200+X100)*(X1+X2))"" : string
```

Problem 1.93: Is the expression $e := \overline{x123 * x72} + x123 * x4$ valid, satisfiable, unsatisfiable, falsifiable? Justify your answer. 3pt 5min

Solution: To determine the class of e, we determine its value under all assignments in a truth table,

assignments			intermediate results			full
$x4$	$x72$	$x123$	$e_1 := x123 * x72$	$e_2 := \overline{e_1}$	$e_3 := x123 * x4$	$e_2 + e_3$
F	F	F	F	T	F	T
F	F	T	F	T	F	T
F	T	F	F	T	F	T
F	T	T	T	F	F	F
T	F	F	F	T	F	T
T	F	T	F	T	T	T
T	T	F	F	T	F	T
T	T	T	T	F	T	T

Ergo, e is satisfiable and falsifiable.

Problem 1.94 (Evaluating Expressions) 2pt 7min

Let $e := \overline{x_1 + x_2} + (\overline{\overline{x_2} * x_3} + x_3 * x_4)$ and $\varphi := [F/x_1], [F/x_2], [T/x_3], [F/x_4]$, compute the value $\mathcal{I}_\varphi(e)$, give a (partial) trace of the computation.

Solution:

$$\begin{aligned}
& \mathcal{I}_\varphi(\overline{x_1 + x_2} + (\overline{(\overline{x_2} * x_3)} + (x_3 * x_4))) \\
= \quad & \mathcal{I}_\varphi(\overline{x_1 + x_2}) \vee \mathcal{I}_\varphi(\overline{(\overline{x_2} * x_3)} + (x_3 * x_4)) \\
= \quad & \neg\mathcal{I}_\varphi(x_1 + x_2) \vee \mathcal{I}_\varphi(\overline{(\overline{x_2} * x_3)}) \vee \mathcal{I}_\varphi(x_3 * x_4)) \\
= \quad & \neg(\mathcal{I}_\varphi(x_1) \vee \mathcal{I}_\varphi(x_2)) \vee (\neg(\mathcal{I}_\varphi(\overline{x_2} * x_3)) \vee \mathcal{I}_\varphi(x_3 * x_4)) \\
= \quad & \neg(\varphi(x_1) \vee \varphi(x_2)) \vee (\neg(\mathcal{I}_\varphi(\overline{x_2}) \wedge \mathcal{I}_\varphi(x_3)) \vee (\mathcal{I}_\varphi(x_3) \wedge \mathcal{I}_\varphi(x_4))) \\
= \quad & \neg(\mathsf{F} \vee \mathsf{F}) \vee (\neg(\neg\mathcal{I}_\varphi(x_2) \wedge \varphi(x_3)) \vee (\varphi(x_3) \wedge \varphi(x_4))) \\
= \quad & \neg\mathsf{F} \vee (\neg(\neg\varphi(x_2) \wedge \mathsf{T}) \vee (\mathsf{T} \wedge \mathsf{F})) \\
= \quad & \mathsf{T} \vee (\neg(\neg\mathsf{F} \wedge \mathsf{T}) \vee \mathsf{F}) \\
= \quad & \mathsf{T} \vee (\neg(\mathsf{T} \wedge \mathsf{T}) \vee \mathsf{F}) \\
= \quad & \mathsf{T} \vee (\neg\mathsf{T} \vee \mathsf{F}) \\
= \quad & \mathsf{T} \vee (\mathsf{F} \vee \mathsf{F}) \\
= \quad & \mathsf{T} \vee \mathsf{F} = \mathsf{T}
\end{aligned}$$

Problem 1.95 (Boolean Equivalence)
Prove or refute the following equivalence:

$$\overline{x_1 * x_1 + \overline{\overline{x_1} + x_2}} \equiv (\overline{x_1} + x_2) * ((\overline{x_1} + \overline{x_2}) * (\overline{x_1} + \overline{x_1}))$$

For each step write down which equivalence rule you used (by equivalence rules we mean commutativity, associativity, etc.).

Solution:

$$\begin{aligned}
\overline{x_1 * x_1 + \overline{\overline{x_1} + x_2}} &\equiv \overline{\overline{\overline{x_1} + x_2} + x_1 * x_1} && \text{(commutativity)} \\
&\equiv \overline{x_1 * \overline{x_2} + x_1 * x_1} && \text{(De Morgan)} \\
&\equiv \overline{x_1 * (\overline{x_2} + x_1)} && \text{(distributivity)} \\
&\equiv \overline{x_1} && \text{(covering)}
\end{aligned}$$

$$\begin{aligned}
(\overline{x_1} + x_2) * ((\overline{x_1} + \overline{x_2}) * (\overline{x_1} + \overline{x_1})) &\equiv (\overline{x_1} + x_2) * (\overline{x_1} + \overline{x_2}) && \text{(consensus)} \\
&\equiv \overline{x_1} && \text{(combining)}
\end{aligned}$$

Since both expressions are equivalent to $\overline{x_1}$, they are equivalent to each other.

1.6.2 Boolean Functions

10pt

Problem 1.96 (Induced Boolean Function)
Determine the Boolean function f_e induced by the Boolean expression $e := (x1 + x2) * \overline{x1 * x3}$. Moreover determine the CNF and DNF of f_e.

Solution:

argument	*value*	*argument*	*value*
$\langle$F, F, F$\rangle$	T	$\langle$T, F, F$\rangle$	T
$\langle$F, F, T$\rangle$	T	$\langle$T, F, T$\rangle$	T
$\langle$F, T, F$\rangle$	T	$\langle$T, T, F$\rangle$	F
$\langle$F, T, T$\rangle$	T	$\langle$T, T, T$\rangle$	T

Problem 1.97 (CNF and DNF)
Write the CNF and DNF of the boolean function that corresponds to the truth table below.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Solution:

DNF: $\overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}\,x_2\,\overline{x_3} + x_1\,\overline{x_2}\,\overline{x_3} + x_1\,\overline{x_2}\,x_3 + x_1\,x_2\,\overline{x_3}$

CNF: $(x_1 + x_2 + \overline{x_3})\,(x_1 + \overline{x_2} + \overline{x_3})\,(\overline{x_1} + \overline{x_2} + \overline{x_3})$

1.6.3 Complexity Analysis for Boolean Expressions

Problem 1.98 (Landau sets)
Order the landau sets below by specifying which ones are subsets and which ones are equal (e.g.: $O(a) \subset O(b) \subset O(c) \equiv O(d) \subset O(e)...$)

$$O(n^2);\ O((n)!);\ O(|\sin n|);\ O(n^n);\ O(1);\ O(2^n);\ O(2n^2 + 2^{72})$$

Solution: $O(|\sin n|) \subset O(1) \subset O(2n^2 + 2^{72}) \equiv O(n^2) \subset O(2^n) \subset O((n)!) \subset O(n^n)$

Problem 1.99 (Relations among polynomials) 5pt 0min

Prove or refute that $O(n^i) \subseteq O(n^j)$ for $0 \leq i < j, n$ $(i, j, n \in \mathbb{N})$.

Problem 1.100: Determine for the following functions f and g whether $f \in O(g)$, or $f \in \Omega(g)$, or $f \in \Theta(g)$, explain your answers. 3pt 10min

f	g	f	g
4572	84	$n^3 + 3 * n$	n^3
$\log(n^3)$	$\log(n)$	$(n^2) - 2^2$	n^3
16^n	2^n	n^n	2^{n+1}

Solution: The following table summarizes the results.

Fact	Explanation
$4572 \in \Theta(84)$	For all $n \in \mathbb{N}$ we have $1000 \cdot 84 \leq 4572$ and $0.001 \cdot 4572 \leq 84$
$(n^3) + 3 * n \in \Omega(n^3)$	For all n: If $c = 1$ then $n^3 + 3 * n \geq n^3$ and if $c = 10$ then $n^3 + 3 * n \leq n^3$.
$(log(n^3)) \in \Theta(log(n))$	Since $log(n^3) = 3 \cdot log(n)$
$((n^2) - 2^2) \in \Omega(n^3)$	larger exponents win
$(16^n) \in \Theta(2^n)$	For all c there is an n such that $16^n \geq c \cdot 2^n$; just take n for a given c such that $8^n \geq c$.
$(n^n) \in O(2^{n+1})$	For $c = 2$ and $n > 1$ we have $2^{n+1} = 2 * 2^n \leq c \cdot n^n$

Problem 1.101 (Upper and lower bounds)
For each of the functions below determine whether $f \in O(g)$, $f \in \Omega(g)$ or $f \in \Theta(g)$. Briefly explain your answers.

1. $f(n) = 235,\ g(n) = 12$
2. $f(n) = n,\ g(n) = 16n$
3. $f(n) = \log_{10}(n),\ g(n) = 7n + 2$
4. $f(n) = 7n^3 + 4n - 2,\ g(n) = 3n^4 + 1$
5. $f(n) = \frac{\log_2(n)}{n},\ g(n) = \frac{n}{\log_2(n)}$
6. $f(n) = 8^n,\ g(n) = 2^n$
7. $f(n) = n^{\log_n(5)},\ g(n) = 2^n$
8. $f(n) = n^n,\ g(n) = (\log_n(3))(n)!$
9. $f(n) = \binom{n}{2},\ g(n) = \binom{n}{4}$

Solution:

1. $f \in \Theta(g)$ both are constants.
2. $f \in \Theta(g)$ the leading terms of the polynomial are of the same order.
3. $f \in O(g)$ n grows faster than $\log_2(n)$ as in the slides.
4. $f \in O(g)$ the leading term of f n^3 grows slower than n^4 from g.
5. $f \in O(g)$ The numerator of f grows slower that the numerator of g and the denominator of f grows faster than the one from g. There fore f clearly grows slower.
6. $f \in \Omega(g)$ $8^n = 2^{3n}$ which clearly grows faster than 2^n.
7. $f \in O(g)$ 2^n is or a higher rank (see slides) and grows much faster.
8. $f \in \Omega(g)$ n^n is clearly asymptotically bigger than $(n)!$. And the logarithm in front plays an insignificant role when n is large.
9. $f \in O(g)$ The first is a polynomial of degree 2 while the second is a polynomial of degree 4.

Problem 1.102: What is the time complexity of the following SML function? Take one evaluation step to be a creation of a head in function `unwork` and disregard other operations.

```
fun gigatwist lst = let

        fun unwork nil = nil |
            unwork(hd::tl) = hd::unwork(tl)

        fun nextwork(nil, _) = nil |
            nextwork(hd::tl, fnc) = fnc(lst)@nextwork(tl, fnc)

        fun nthwork 1 = unwork |
            nthwork n = let
               fun work arg = nextwork(arg, nthwork(n-1))
            in
             work
            end
in
         nthwork(length lst) lst
end
```

Solution: Time complexity is $\Theta(n^n)$.

Problem 1.103 (Proof of Membership in Landau Set) 30pt 10min

Prove by induction or refute: the function $f(n) := n^n$ is in $O((n)!^2)$; i.e. there is a constant c such that $n^n \leq (n)!^2$ for sufficiently large n.

Hint:

Solution: We choose $c = 1$. Induction step: Show $(n+1)^{(n+1)} \leq (n+1)!^2$ under the induction hypothesis (IH) is $n^n \leq (n)!^2$. We have $(n+1)!^2 = n+1^2 * (n)!^2 \geq n+1^2n^n$ by (IH). Hence, we have to show that $n+1^2n^n \geq n+1^{n+1}$ which we do by the equivalence transformations $n+1^2n^n \geq n+1^{n+1}$ and $n^n \geq n+1^{n-1}$ and now??? TODO !!!!!!

1.6.4 The Quine-McCluskey Algorithm

14pt

Problem 1.104 (Quine-McCluskey)
Execute the QMC algorithm for the following function:

x_1	x_2	x_3	f
F	F	F	T
F	F	T	T
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	F
T	T	F	T
T	T	T	T

Moreover you are required to find the solution with minimal cost where each operation (and, not, or) adds 1 to the cost. E.g. the cost of $(\overline{x_1} + x_3)\,(x_3)$ is 3.

Solution:

QMC_1 :

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

QMC_2 :

	FFF	FFT	FTT	TFF	TTF	TTT
$\overline{x_1}\,\overline{x_2}$	T	T	F	F	F	F
$\overline{x_2}\,\overline{x_3}$	T	F	F	T	F	F
$\overline{x_1}\,x_3$	F	T	T	F	F	F
$x_2\,x_3$	F	F	T	F	F	T
$x_1\,x_2$	F	F	F	F	T	T
$x_1\,\overline{x_3}$	F	F	F	T	T	F

Final result: There are two solutions with optimal cost 8 which are actually the only solutions with three polynomials:

1. $f = x_2\,x_3 + \overline{x_1}\,\overline{x_2} + x_1\,\overline{x_3}$
2. $f = x_2\,x_3 + \overline{x_2}\,\overline{x_3} + \overline{x_1}\,x_3$

Problem 1.105: Use the algorithm of Quine-McCluskey to determine the minimal polynomial of the following functions: 35pt

$x1$	$x2$	$x3$	$x4$	f_1
F	F	F	F	F
F	F	F	T	F
F	F	T	F	T
F	F	T	T	T
F	T	F	F	T
F	T	F	T	T
F	T	T	F	T
F	T	T	T	T
T	F	F	F	T
T	F	F	T	F
T	F	T	F	F
T	F	T	T	T
T	T	F	F	T
T	T	F	T	F
T	T	T	F	F
T	T	T	T	F

$x1$	$x2$	$x3$	$x4$	f_2
F	F	F	F	T
F	F	F	T	F
F	F	T	F	T
F	F	T	T	F
F	T	F	F	F
F	T	F	T	F
F	T	T	F	F
F	T	T	T	T
T	F	F	F	T
T	F	F	T	T
T	F	T	F	F
T	F	T	T	F
T	T	F	F	F
T	T	F	T	F
T	T	T	F	F
T	T	T	T	T

Solution: For f_1, we first enter the monomials and delete the rows that do not result in a monomial:

$x1$	$x2$	$x3$	$x4$	$Monomials$
F	F	T	F	$x1^0\,x2^0\,x3^1\,x4^0$
F	F	T	T	$x1^0\,x2^0\,x3^1\,x4^1$
F	T	F	F	$x1^0\,x2^1\,x3^0\,x4^0$
F	T	F	T	$x1^0\,x2^1\,x3^0\,x4^1$
F	T	T	F	$x1^0\,x2^1\,x3^1\,x4^0$
F	T	T	T	$x1^0\,x2^1\,x3^1\,x4^1$
T	F	F	F	$x1^1\,x2^0\,x3^0\,x4^0$
T	F	T	T	$x1^1\,x2^0\,x3^1\,x4^1$
T	T	F	F	$x1^1\,x2^1\,x3^0\,x4^0$

The next two tables show each step in the Quine McCluskey Algorithm.

$x1$	$x2$	$x3$	$x4$	$Monomials$
F	F	T	X	$x1^0\,x2^0\,x3^1$
F	X	T	F	$x1^0\,x3^1\,x4^0$
F	X	T	T	$x1^0\,x3^1\,x4^1$
X	F	T	T	$x2^0\,x3^1\,x4^1$
X	T	F	F	$x2^1\,x3^0\,x4^0$
F	T	X	T	$x1^0\,x2^1\,x4^0$
F	T	T	X	$x1^0\,x2^1\,x3^1$
F	T	F	X	$x1^0\,x2^1\,x3^0$
T	X	F	F	$x1^1\,x3^0\,x4^0$
F	T	X	F	$x1^0\,x2^1\,x4^0$

$x1$	$x2$	$x3$	$x4$	$Monomials$
F	X	T	X	$x1^0\,x3^1$
F	T	X	X	$x1^0\,x2^1$
X	F	T	T	$x2^0\,x3^1\,x4^1$
X	T	F	F	$x2^1\,x3^0\,x4^0$
T	X	F	F	$x1^1\,x3^0\,x4^0$

Finally, we have to determine the prime implicants that form the minimal polynomial.

	FFTF	FFTT	FTFF	FTFT	FTTF	FTTT	TFFF	TFTT	TTFF
$\overline{x1}\,x3$	T	T	F	F	T	T	F	F	F
$\overline{x1}\,x2$	F	F	T	T	T	T	F	F	F
$\overline{x2}\,x3\,x4$	F	T	F	F	F	F	F	T	F
$x2\,\overline{x3}\,x4$	F	F	T	F	F	F	F	F	T
$x1\,\overline{x3}\,\overline{x4}$	F	F	F	F	F	F	T	F	T

All prime implicants but the last one are essential. Hence, the minimal polynomial of f_1 is:

$$f_1 = \overline{x1}\,x3 + \overline{x1}\,x2 + \overline{x2}\,x3\,x4 + x2\,\overline{x3}\,x4$$

For f_2, we first enter the monomials and delete all rows which do not result in a monomial and get the following table from which we can start the algorithm from.

$x1$	$x2$	$x3$	$x4$	$Monomials$
F	F	F	F	$x1^0\,x2^0\,x3^0\,x4^0$
F	F	T	F	$x1^0\,x2^0\,x3^1\,x4^0$
F	T	T	T	$x1^0\,x2^1\,x3^1\,x4^1$
T	F	F	F	$x1^1\,x2^0\,x3^0\,x4^0$
T	F	F	T	$x1^1\,x2^0\,x3^0\,x4^1$
T	T	T	T	$x1^1\,x2^1\,x3^1\,x4^1$

The next table shows the only step in the Quine McCluskey Algorithm that can be made for this function.

$x1$	$x2$	$x3$	$x4$	$Monomials$
F	F	X	F	$x1^0\,x2^0\,x4^0$
X	F	F	F	$x2^0\,x3^0\,x4^0$
T	F	F	X	$x1^1\,x2^0\,x3^0$
X	T	T	T	$x2^1\,x3^1\,x4^1$

We are already done after one step. Now, we have to find out the prime implicants that form the minimal polynomial.

	FFFF	FFTF	TFFF	TFFT	FTTT	TTTT
$\overline{x1}\,\overline{x2}\,\overline{x4}$	T	T	F	F	F	F
$\overline{x2}\,\overline{x3}\,\overline{x4}$	T	F	T	F	F	F
$x1\,\overline{x2}\,\overline{x3}$	F	F	T	T	F	F
$x2\,x3\,x4$	F	F	F	F	T	T

We see that all of the prime implicants but the second one are needed for the minimal polynomial. Hence, we are finished and can write the polynomial. Our resulting polynomial is:

$$f_2 = \overline{x1}\,\overline{x2}\,\overline{x4} + x1\,\overline{x2}\,\overline{x2} + x1\,x3\,x4$$

Problem 1.106 (Quine-McCluskey with Don't-Cares) 15pt

How can the Quine-McCluskey algorithm be modified to take advantage of don't-cares? Find out which steps of the algorithm are affected by this modification and explain how they change by showing the respective steps of applying the algorithm to the function $f(x1, x2, x3, x4)$ that yields T for $x1^0\, x2^1\, x3^0\, x4^0$, $x1^0\, x2^1\, x3^0\, x4^1$, $x1^0\, x2^1\, x3^1\, x4^0$, $x1^1\, x2^0\, x3^0\, x4^0$, $x1^1\, x2^0\, x3^0\, x4^1$, $x1^1\, x2^0\, x3^1\, x4^0$, $x1^1\, x2^1\, x3^0\, x4^1$, "don't care" for $x1^0\, x2^0\, x3^0\, x4^0$, $x1^0\, x2^1\, x3^1\, x4^1$, $x1^1\, x2^1\, x3^1\, x4^1$, and F for the other inputs.

Solution: A nice explanation for the same function can be found at `http://www-static.cc.gatech.edu/classes/AY2005/cs3220_spring/quine-mccluskey.pdf`. One basically takes all all inputs with a don't-care output into account in QMC_1, where the prime implicants are determined. In the top row of the table used for QMC_2, the don't-cares are not included.

Problem 1.107 (CNF with Quine-McCluskey) 14pt 12min

In class you have learned how to derive the optimal formula for a given function in DNF form using the Quine-McCluskey algorithm. It appears that the same algorithm could be applied to find the optimal formula in CNF form. Think of how this can be done and apply it on the function defined by the following table:

x_1	x_2	x_3	f
F	F	F	T
F	F	T	T
F	T	F	T
F	T	T	F
T	F	F	T
T	F	T	T
T	T	F	F
T	T	T	F

Hint:

The basic rule used in the QMC algorithm: $a\,x + a\,\overline{x} = a$ also applies for formulas in CNF: $(a + x)\,(a + \overline{x}) = (a)$

Solution:

QMC_1 :

$$\begin{aligned} C_0 &= \{x_1 + \overline{x_2} + \overline{x_3}, \overline{x_1} + \overline{x_2} + x_3, \overline{x_1} + \overline{x_2} + \overline{x_3}\} \\ P_1 &= \emptyset \\ C_1 &= \{\overline{x_2} + \overline{x_3}, \overline{x_1} + \overline{x_2}\} \\ P_2 &= \{\overline{x_2} + \overline{x_3}, \overline{x_1} + \overline{x_2}\} \end{aligned}$$

QMC_2 :

	FTT	TTF	TTT
$\overline{x_2} + \overline{x_3}$	F	T	F
$\overline{x_1} + \overline{x_2}$	T	F	F

Final result:

$$f = (\overline{x_2} + \overline{x_3})\,(\overline{x_1} + \overline{x_2})$$

1.6.5 A simpler Method for finding Minimal Polynomials

10pt

Problem 1.108 (Karnaugh-Veitch Minimization)

Given the boolean function $f = B * \overline{D+C} + \overline{B} * \overline{(D+\overline{A}) * (A+D)}$:

1. Use a KV map to determine the minimal polynomial for the function.
2. Try to further reduce the cost of the resulting polynomial using boolean equivalences. The result does not need to be a polynomial.
3. Using boolean equivalences, transform the original expression into the the result from (2). Show all intermediate steps.

Solution:

1. The KV map looks like this:

	$\overline{AB}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{CD}$	T	T	T	T
$\overline{C}D$	F	F	F	F
CD	F	F	F	F
$C\overline{D}$	T	F	F	T

   The minimal polynomial is: $\overline{D}\,\overline{B} + \overline{D}\,\overline{C}$

2. We can reduce the cost by 1 if we use the following expression: $f = \overline{D} * (\overline{B} + \overline{C})$

3.

$$\begin{aligned}
f = \quad & B * \overline{D+C} + \overline{B} * \overline{(D+\overline{A}) * (A+D)} \\
= \quad & B * (\overline{D} * \overline{C}) + \overline{B} * (\overline{D+\overline{A}} + \overline{A+D}) \\
= \quad & B * (\overline{D} * \overline{C}) + \overline{B} * (\overline{D} * A + \overline{A} * \overline{D}) \\
= \quad & B * (\overline{D} * \overline{C}) + (\overline{B} * (\overline{D} * A) + \overline{B} * (\overline{A} * \overline{D})) \\
= \quad & B * (\overline{D} * \overline{C}) + \overline{B} * \overline{D} \\
= \quad & \overline{D} * (B * \overline{C} + \overline{B}) \\
= \quad & \overline{D} * (\overline{C} + \overline{B})
\end{aligned}$$

Problem 1.109 (Karnaugh-Veitch Diagrams) 10min

1. Use a KV map to determine all possible minimal polynomials for the function defined by the following truth table:

A	B	C	D	f
F	F	F	F	F
F	F	F	T	T
F	F	T	F	T
F	F	T	T	F
F	T	F	F	T
F	T	F	T	F
F	T	T	F	T
F	T	T	T	T
T	F	F	F	T
T	F	F	T	T
T	F	T	F	F
T	F	T	T	T
T	T	F	F	T
T	T	F	T	T
T	T	T	F	F
T	T	T	T	T

2. How would you use a KV map to find a minimal polynomial for a function with 5 variables? What does your map look like? Which borders in the map are virtually connected? (A simple but clear explanation suffices.)

Solution:

1. The resulting KV Map is:

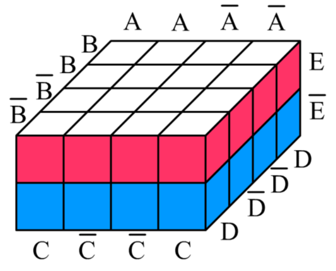
	$\overline{A}\overline{B}$	$A\overline{B}$	AB	$\overline{A}B$
$\overline{C}\overline{D}$	F	T	T	T
$C\overline{D}$	T	F	F	T
CD	F	T	T	T
$\overline{C}D$	T	T	T	F

The two possible minimal polynomials are:

(a) $A\,D + A\,\overline{C} + \overline{B}\,\overline{C}\,D + \overline{A}\,C\,\overline{D} + B\,C\,D + B\,\overline{C}\,\overline{D}$

(b) $A\,D + A\,\overline{C} + \overline{B}\,\overline{C}\,D + \overline{A}\,C\,\overline{D} + B\,C\,D + \overline{A}\,B\,\overline{D}$

2. The picture below should be self explanatory:



Problem 1.110 (CNF with Karnaugh-Veitch Diagrams) 15pt 6min

KV maps can also be used to compute a minimal CNF for a Boolean function. Using the function $f(x1, x2, x3)$ that yields T for $x1^0\, x2^0\, x3^0$, $x1^0\, x2^1\, x3^0$, $x1^0\, x2^1\, x3^1$, $x1^1\, x2^0\, x3^0$, and F for the other inputs, develop an idea (and verify it for this example!) how to do this.

Hint: Start by grouping F-cells together.

Solution: Grading: Assuming 3 pt = 100%:

- 1 pt for the map
- 0.5 pt for correct grouping
- 1 pt for a reasonable description of the procedure
- 0.5 pt for a correct minimal CNF

Problem 1.111 (Karnaugh-Veitch Diagrams with Don't-Cares) 10pt

In some cases, there is an input $d \in \mathbf{dom}(f)$ to a boolean function $f \colon \mathbb{B}^n \to \mathbb{B}$ for which no output is specified — because the input is invalid or it would never occur. In a truth table for f, a function value $f(d)$ would be written as X instead of F or T, which means, "Don't care!"

Describe how don't-cares can be utilized when determining the minimal polynomial of a Boolean function using a KV map.

Note: Considering don't-cares is particularly beneficial when designing digital circuits. This will be done in GenCS 2. Just consider an electronic device with six states, which we can conveniently encode by using three boolean memory elements, which leads to $2^3 - 6 =$ two leftover "don't-care" states.

Solution: One tries to assign values, either F or T, to the don't-care fields that lead to maximal groups in the KV map.

Problem 1.112 (Don't-Care Minimization) 10pt

1. Devise a concrete Boolean function $f\colon \mathbb{B}^4 \to \mathbb{B}$ that gives T for 6 of the 16 possible inputs, F for 7 inputs, and "don't care" for the remaining 3 possible inputs.
2. Apply the don't-care minimization algorithm from the previous exercise to it.
3. Then replace all don't-cares by T, do minimization without don't-cares, compare, and give a short comment.

1.7 Propositional Logic

1.7.1 Boolean Expressions and Propositional Logic

Problem 1.113 (The *Nor* Connective) 2pt 7min

All logical binary connectives can be expressed by the $\downarrow$ (*nor*) connective which is defined as $\mathbf{A} \downarrow \mathbf{B} := \neg(\mathbf{A} \vee \mathbf{B})$. Rewrite $\mathbf{P} \vee \neg\mathbf{P}$ (tertium non datur) into an expression containing only $\downarrow$ as a logical connective.

Hint: Recall that $\neg\mathbf{A} \Leftrightarrow \mathbf{A} \downarrow \mathbf{A}$.

Solution: $P \vee \neg P = \neg\neg(P \vee \neg P) = \neg(P \downarrow \neg P) = (P \downarrow (P \downarrow P)) \downarrow (P \downarrow (P \downarrow P))$

1.7.2 Logical Systems and Calculi

Problem 1.114 (Calculus Properties)
Explain briefly what the following properties of calculi mean:

- correctness
- completeness

Solution:

- correctness ($\mathcal{H} \vdash \mathbf{B}$ implies $\mathcal{H} \models \mathbf{B}$) - A calculus is correct if any derivable(provable) formula is also a valid formula.
- completeness ($\mathcal{H} \models \mathbf{B}$ implies $\mathcal{H} \vdash \mathbf{B}$) - A calculus is complete if any valid formula can also be derived(proven).

1.7.3 Proof Theory for the Hilbert Calculus

5pt

Problem 1.115: We have proven the correctness of the Hilbert calculus $\mathcal{H}^0$ in class. The problems of this quiz is about two incorrect calculi $\mathcal{C}^1$ and $\mathcal{C}^2$ which differ only slightly from $\mathcal{H}^0$. What makes them incorrect?

Hint: The fact that $\mathcal{H}^0$ has two axioms, but each of $\mathcal{C}^1$ and $\mathcal{C}^2$ only have one is not the point. Remember the properties of axioms and inference rules which are preconditions for a correct calculus.

Why is this calculus $\mathcal{C}^1$ incorrect?

- $\mathcal{C}^1$ Axiom:$P \Rightarrow P \wedge Q$
- $\mathcal{C}^1$ Inference Rules: $\dfrac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}\,\mathrm{MP} \qquad \dfrac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\,\mathrm{Subst}$

Why is this calculus $\mathcal{C}^2$ incorrect?

- $\mathcal{C}^2$ Axiom: $P \Rightarrow (Q \Rightarrow P)$
- $\mathcal{C}^2$ Inference Rules: $\dfrac{\mathbf{A} \vee \mathbf{B} \quad \mathbf{A}}{\mathbf{A} \wedge \mathbf{B}}\,R2 \qquad \dfrac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\,\mathrm{Subst}$

Solution: A correct calculus requires valid axioms.
However the Axiom of $\mathcal{C}^1$ is not valid since the assignment $\varphi = [T/P], [T/Q], [F/R]$ makes it false.

Problem 1.116 (Almost a Proof)
Please consider the following sequence of formulae: it pretends to be a proof of the formula $\mathbf{A} \Rightarrow \mathbf{A}$ in $\mathcal{H}^0$. For each line annotate how it is derived by the inference rules from proceeding lines or axioms. If a line is not derivable in such a manner then mark it as underivable and explain what went wrong.

Use the aggregate notation we used in the slides for derivations with multiple steps (e.g. an axiom with multiple applications of the Subst rule)

1. $\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A})$
2. $\mathbf{B} \Rightarrow \mathbf{A}$
3. $\mathbf{B} \Rightarrow (\mathbf{A} \Rightarrow \mathbf{B})$
4. $\mathbf{A} \Rightarrow \mathbf{B}$
5. $(\mathbf{B} \Rightarrow \mathbf{A}) \Rightarrow (\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A}))$
6. $(\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A})) \Rightarrow ((\mathbf{A} \Rightarrow \mathbf{B}) \Rightarrow (\mathbf{A} \Rightarrow \mathbf{A}))$
7. $(\mathbf{A} \Rightarrow \mathbf{B}) \Rightarrow (\mathbf{A} \Rightarrow \mathbf{A})$
8. $\mathbf{A} \Rightarrow \mathbf{A}$

Solution:

1.	$\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A})$	Ax1 with $[\mathbf{A}/P]$ and $[\mathbf{B}/Q]$
2.	$\mathbf{B} \Rightarrow \mathbf{A}$	underivable
3.	$\mathbf{B} \Rightarrow (\mathbf{A} \Rightarrow \mathbf{B})$	Ax1 with $[\mathbf{B}/P]$ and $[\mathbf{A}/Q]$
4.	$\mathbf{A} \Rightarrow \mathbf{B}$	underivable
5.	$(\mathbf{B} \Rightarrow \mathbf{A}) \Rightarrow (\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A}))$	Ax1 with $[\mathbf{B} \Rightarrow \mathbf{A}/P]$ and $[\mathbf{A}/Q]$
6.	$(\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{A})) \Rightarrow ((\mathbf{A} \Rightarrow \mathbf{B}) \Rightarrow (\mathbf{A} \Rightarrow \mathbf{A}))$	Ax2 with $[\mathbf{A}/P]$, $[\mathbf{B}/Q]$ and $[\mathbf{A}/R]$
7.	$(\mathbf{A} \Rightarrow \mathbf{B}) \Rightarrow (\mathbf{A} \Rightarrow \mathbf{A})$	MP16
8.	$\mathbf{A} \Rightarrow \mathbf{A}$	MP47

Problem 1.117: We have proven the correctness of the Hilbert calculus $\mathcal{H}^0$ in class. The problems of this quiz is about two incorrect calculi $\mathcal{C}^1$ and $\mathcal{C}^2$ which differ only slightly from $\mathcal{H}^0$. What makes them incorrect?

Hint: The fact that $\mathcal{H}^0$ has two axioms, but each of $\mathcal{C}^1$ and $\mathcal{C}^2$ only have one is not the point. Remember the properties of axioms and inference rules which are preconditions for a correct calculus.

Why is this calculus $\mathcal{C}^1$ incorrect?

- $\mathcal{C}^1$ Axiom: $P \Rightarrow (Q \Rightarrow R)$
- $\mathcal{C}^1$ Inference Rules: $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}\,\mathrm{MP}$ $\qquad$ $\frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\,\mathrm{Subst}$

Solution: A correct calculus requires valid axioms.
However the Axiom of $\mathcal{C}^1$ is not valid since the assignment $\varphi = [T/P], [T/Q], [F/R]$ makes it false.

Problem 1.118 (Alternative Calculus)
Consider a calculus given by the axioms $\mathbf{A} \vee \neg\mathbf{A}$ and $\mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{B} \wedge \mathbf{A}$ and the following rules:

$$\frac{\mathbf{A} \Rightarrow \mathbf{B}}{\neg\mathbf{B} \Rightarrow \neg\mathbf{A}}\, Transp \qquad \frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\, \text{Subst}$$

Prove that the calculus is sound.

Solution: First we show that the axioms are theorems by constructing their truth tables:

$\mathbf{A}$	$\neg\mathbf{A}$	$\mathbf{A} \vee \neg\mathbf{A}$
0	1	1
1	0	1

$\mathbf{A}$	$\mathbf{B}$	$\mathbf{A} \vee \mathbf{B}$	$\mathbf{B} \vee \mathbf{A}$	$\mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{B} \wedge \mathbf{A}$
0	0	0	0	1
0	1	1	1	1
1	0	1	1	1
1	1	1	1	1

The substitutionn rule is shown to be sound in the slides, so we are left to show that transposition is sound. We use a truth table to show that its outcome is true whenever the precondition is true.

$\mathbf{A}$	$\mathbf{B}$	$\neg\mathbf{A}$	$\neg\mathbf{B}$	$\mathbf{A} \Rightarrow \mathbf{B}$	$\neg\mathbf{B} \Rightarrow \neg\mathbf{A}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	0	0
1	1	0	0	1	1

All axioms and rules were shown to be sound, thus we can conclude that the calculus is sound.

Problem 1.119 (A calculus for propositional logic) 10pt 10min

Let us assume a calculus for propositional logic that consists of the single axiom $\mathbf{A} \Rightarrow \mathbf{A}$ and the inference rule:

$$\frac{\mathbf{A} \Rightarrow (\mathbf{B} \Rightarrow \mathbf{C})}{\mathbf{A} \wedge \mathbf{B} \Rightarrow \mathbf{C}} \qquad\qquad \frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\,\text{Subst}$$

1. Show that this calculus is sound (i. e. correct).
2. Prove the formula $((P \Rightarrow Q) \wedge P) \Rightarrow Q$ using this calculus.

Solution:

1. Induction over proof length:
   - The axiom is valid.
   - The first inference rule is valid.
   - The substitution inference rule is valid (see lecture).
2. $$\begin{array}{ll} \vdash & \mathbf{A} \Rightarrow \mathbf{A} \\ \vdash & (P \Rightarrow Q) \Rightarrow (P \Rightarrow Q) \\ \vdash & ((P \Rightarrow Q) \wedge P) \Rightarrow Q \end{array}$$

Problem 1.120 (Hilbert Calculus)
Prove the following theorem using $\mathcal{H}^0$: $((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{A}) \Rightarrow ((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A}))$

Solution:
Proof:

P.1 $((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow (\mathbf{A} \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A}))) \Rightarrow (((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{A}) \Rightarrow ((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A})))$ ($\mathbf{S}$ with $[\mathbf{A} \Rightarrow \mathbf{C}/P], [\mathbf{A}/Q], [($

P.2 $\mathbf{A} \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A})$ ($\mathbf{K}$ with $[\mathbf{A}/P], [\mathbf{B} \Rightarrow \mathbf{B}/Q]$)

P.3 $(\mathbf{A} \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A})) \Rightarrow ((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow (\mathbf{A} \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A})))$ ($\mathbf{K}$ with $[\mathbf{A} \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A})/P], [\mathbf{A} \Rightarrow \mathbf{C}/Q]$)

P.4 $(\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow (\mathbf{A} \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A}))$ (MP on P.2 and P.3)

P.5 $((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow \mathbf{A}) \Rightarrow ((\mathbf{A} \Rightarrow \mathbf{C}) \Rightarrow ((\mathbf{B} \Rightarrow \mathbf{B}) \Rightarrow \mathbf{A}))$ (MP on P.1 and P.4)

□

Problem 1.121 (A Hilbert Calculus) 20pt

Consider the Hilbert-style calculus given by the following axioms:

1. $(\mathbf{F} \vee \mathbf{F}) \Rightarrow \mathbf{F}$ (idempotence of disjunction)
2. $\mathbf{F} \Rightarrow (\mathbf{F} \vee \mathbf{G})$ (weakening)
3. $(\mathbf{G} \vee \mathbf{F}) \Rightarrow (\mathbf{F} \vee \mathbf{G})$ (commutativity)
4. $(\mathbf{G} \Rightarrow \mathbf{H}) \Rightarrow ((\mathbf{F} \vee \mathbf{G}) \Rightarrow (\mathbf{F} \vee \mathbf{H}))$

and the identities

1. $\mathbf{A} \Rightarrow \mathbf{B} = \neg\mathbf{A} \vee \mathbf{B}$
2. $\mathbf{F} \wedge \mathbf{G} = \neg(\neg\mathbf{F} \vee \neg\mathbf{G})$

You can use the MP and substitution as inference rules:

$$\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}\,\mathrm{MP} \qquad \frac{\mathbf{A}}{[\mathbf{B}/X](\mathbf{A})}\,\mathrm{Subst}$$

Prove the formula $\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\neg\mathbf{P} \vee \neg\mathbf{Q}))$

Solution:

Proof:

P.1 $(((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P}) \Rightarrow (\mathbf{P} \vee (\mathbf{P} \Rightarrow \neg\mathbf{Q}))) \Rightarrow ((\mathbf{P} \wedge \mathbf{Q} \vee ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})) \Rightarrow (\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\mathbf{P} \Rightarrow \neg\mathbf{Q}))))$ (ax.4 with $[\mathbf{P} \wedge \mathbf{Q}/F], [(\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P}/G], [\mathbf{P} \vee (\mathbf{P} \Rightarrow \neg\mathbf{Q})/H]$)

P.2 $((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P}) \Rightarrow (\mathbf{P} \vee (\mathbf{P} \Rightarrow \neg\mathbf{Q}))$ (ax.3 with $[\mathbf{P}/F], [\mathbf{P} \Rightarrow \neg\mathbf{Q}/G]$)

P.3 $(\mathbf{P} \wedge \mathbf{Q} \vee ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})) \Rightarrow (\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\mathbf{P} \Rightarrow \neg\mathbf{Q})))$ (MP on P.1 and P.2)

P.4 $(\mathbf{P} \wedge \mathbf{Q} \vee ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})) \Rightarrow (\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\neg\mathbf{P} \vee \neg\mathbf{Q})))$ (Identity 1.)

P.5 $(\neg(\neg\mathbf{P} \vee \neg\mathbf{Q}) \vee ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})) \Rightarrow (\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\neg\mathbf{P} \vee \neg\mathbf{Q})))$ (Identity 2.)

P.6 $(\neg(\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})) \Rightarrow (\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\neg\mathbf{P} \vee \neg\mathbf{Q})))$ (Identity 1.)

P.7 $((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \Rightarrow ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})) \Rightarrow (\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\neg\mathbf{P} \vee \neg\mathbf{Q})))$ (Identity 1.)

P.8 $(\mathbf{P} \Rightarrow \neg\mathbf{Q}) \Rightarrow ((\mathbf{P} \Rightarrow \neg\mathbf{Q}) \vee \mathbf{P})$ (ax.2 with $[\mathbf{P} \Rightarrow \neg\mathbf{Q}/F], [\mathbf{P}/G]$)

P.9 $\mathbf{P} \wedge \mathbf{Q} \vee (\mathbf{P} \vee (\neg\mathbf{P} \vee \neg\mathbf{Q}))$ (MP on P.7 and P.8)

□

1.7.4 The Calculus of Natural Deduction

No problems supplied yet.

1.8 Machine-Oriented Calculi

1.8.1 Calculi for Automated Theorem Proving: Analytical Tableaux

Problem 1.122: Prove the Hilbert-Calculus axioms $P \Rightarrow (Q \Rightarrow P)$, and $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))$

Solution:

$$\begin{array}{c} P \Rightarrow (Q \Rightarrow P)^{\mathsf{F}} \\ P^{\mathsf{T}} \\ Q \Rightarrow P^{\mathsf{F}} \\ Q^{\mathsf{T}} \\ P^{\mathsf{F}} \\ \bot \end{array}$$

$$\begin{array}{c} (P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \Rightarrow Q) \Rightarrow (P \Rightarrow R))^{\mathsf{F}} \\ P \Rightarrow (Q \Rightarrow R)^{\mathsf{T}} \\ (P \Rightarrow Q) \Rightarrow (P \Rightarrow R)^{\mathsf{F}} \\ P \Rightarrow Q^{\mathsf{T}} \\ P \Rightarrow R^{\mathsf{T}} \\ P^{\mathsf{T}} \\ R^{\mathsf{T}} \\ \begin{array}{c|c} \begin{array}{c} P^{\mathsf{F}} \\ \bot \end{array} & \begin{array}{c} Q \Rightarrow R^{\mathsf{T}} \\ \begin{array}{c|c} \begin{array}{c} Q^{\mathsf{F}} \\ \begin{array}{c|c} P^{\mathsf{F}} & Q^{\mathsf{T}} \\ \bot & \bot \end{array} \end{array} & \begin{array}{c} R^{\mathsf{T}} \\ \bot \end{array} \end{array} \end{array} \end{array} \end{array}$$

Problem 1.123: Prove the associative law for disjunction $(P \lor Q) \lor R \Leftrightarrow P \lor (Q \lor R)$[2] with the tableau method.

Solution:

$$\begin{array}{c}
(P \lor Q) \lor R \Leftrightarrow P \lor (Q \lor R)^{\mathsf{F}} \\
\begin{array}{c|c}
\begin{array}{c}
(P \lor Q) \lor R^{\mathsf{F}} \\
P \lor (Q \lor R)^{\mathsf{T}} \\
P^{\mathsf{F}} \\
Q^{\mathsf{F}} \\
R^{\mathsf{F}} \\
\begin{array}{c|c|c}
P^{\mathsf{T}} & Q^{\mathsf{T}} & R^{\mathsf{T}} \\
\bot & \bot & \bot
\end{array}
\end{array}
&
\begin{array}{c}
(P \lor Q) \lor R^{\mathsf{T}} \\
P \lor (Q \lor R)^{\mathsf{F}} \\
P^{\mathsf{F}} \\
Q^{\mathsf{F}} \\
R^{\mathsf{F}} \\
\begin{array}{c|c|c}
P^{\mathsf{T}} & Q^{\mathsf{T}} & R^{\mathsf{T}} \\
\bot & \bot & \bot
\end{array}
\end{array}
\end{array}
\end{array}$$

[2]Proving this in the Hilbert calculus from ?? takes about 300 steps.

Problem 1.124 (Tableau Calculus) 0pt 10min

1. Explain the difference between tableau proof of validity and model generation.
2. Derive a tableau inference rule for $A \Leftrightarrow B^{\mathsf{T}}$. Show the derivation.
3. Generate all models of the following expression: $\neg Q \wedge P \Leftrightarrow Q \wedge \neg P$

Solution:

1. Tableau proof of validity is done by assuming the expression to be false and then refuting the assumption by showing that all branches get closed. On the other hand, model generation starts from the assumption that the expression is true and proceeds until all branches are saturated. All open saturated branches lead to models.
2. $A \Leftrightarrow B \equiv (A \Rightarrow B) \wedge (B \Rightarrow A)$, then:

$$\begin{array}{c}
(A \Rightarrow B) \wedge (B \Rightarrow A)^{\mathsf{T}} \\
A \Rightarrow B^{\mathsf{T}} \\
B \Rightarrow A^{\mathsf{T}} \\
\begin{array}{c|c}
\begin{array}{c} A^{\mathsf{F}} \\ \begin{array}{c|c} B^{\mathsf{F}} & \begin{array}{c} A^{\mathsf{T}} \\ \bot \end{array} \end{array} \end{array} &
\begin{array}{c} B^{\mathsf{T}} \\ \begin{array}{c|c} \begin{array}{c} B^{\mathsf{F}} \\ \bot \end{array} & A^{\mathsf{T}} \end{array} \end{array}
\end{array}
\end{array}$$

Thus the rule is:

$$\frac{\mathbf{A} \Leftrightarrow \mathbf{B}^{\mathsf{T}}}{\begin{array}{c|c} \mathbf{A}^{\mathsf{T}} & \mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{B}^{\mathsf{F}} \end{array}}$$

3. We generate models by assuming the expression to be true:

$$\begin{array}{c}
\neg Q \wedge P \Leftrightarrow Q \wedge \neg P^{\mathsf{T}} \\
\begin{array}{c|c}
\neg Q \wedge P^{\mathsf{T}} & \neg Q \wedge P^{\mathsf{F}} \\
Q \wedge \neg P^{\mathsf{T}} & Q \wedge \neg P^{\mathsf{F}} \\
\neg Q^{\mathsf{T}} & \neg Q^{\mathsf{F}} \\
P^{\mathsf{T}} & P^{\mathsf{F}} \\
Q^{\mathsf{T}} & Q^{\mathsf{F}} \\
\neg P^{\mathsf{T}} & \neg P^{\mathsf{F}} \\
Q^{\mathsf{F}} & Q^{\mathsf{T}} \\
P^{\mathsf{F}} & P^{\mathsf{T}} \\
\bot & \bot
\end{array}
\end{array}$$

Clearly, the expression $\neg Q \wedge P \Leftrightarrow Q \wedge \neg P$ has no model. It is unsatisfiable.

Problem 1.125 (Refutation and model generation in Tableau Calculus) 11pt

1. Prove the following proposition:

$$\models \neg A \wedge \neg B \Rightarrow \neg(A \vee B)$$

2. Find all models for the following proposition:

$$\models (A \Rightarrow B) \wedge (B \Rightarrow A \wedge B)$$

Hint: You may use derived rules for implication and disjunction.

Solution:

1.

$$\begin{array}{c} \neg A \wedge \neg B \Rightarrow \neg(A \vee B)^{\mathsf{F}} \\ \neg A \wedge \neg B^{\mathsf{T}} \\ A^{\mathsf{F}} \\ B^{\mathsf{F}} \\ \neg(A \wedge B)^{\mathsf{F}} \\ A \vee B^{\mathsf{T}} \\ \begin{array}{c|c} A^{\mathsf{T}} & B^{\mathsf{T}} \\ \bot & \bot \end{array} \end{array}$$

2.

$$\begin{array}{c} (A \Rightarrow B) \wedge (B \Rightarrow A \wedge B)^{\mathsf{T}} \\ A \Rightarrow B^{\mathsf{T}} \\ B \Rightarrow A \wedge B^{\mathsf{T}} \\ \begin{array}{c|c} \begin{array}{c} B^{\mathsf{F}} \\ A \Rightarrow B^{\mathsf{T}} \\ \begin{array}{c|c} A^{\mathsf{F}} & B^{\mathsf{T}} \\ & \bot \end{array} \end{array} & \begin{array}{c} A \wedge B^{\mathsf{T}} \\ A^{\mathsf{T}} \\ B^{\mathsf{T}} \\ A \Rightarrow B^{\mathsf{T}} \\ \begin{array}{c|c} A^{\mathsf{F}} & B^{\mathsf{T}} \\ \bot & \end{array} \end{array} \end{array} \end{array}$$

That yields the models $\varphi := \{A \mapsto \mathsf{F}, B \mapsto \mathsf{F}\}$ and $\psi := \{A \mapsto \mathsf{T}, B \mapsto \mathsf{T}\}$.

Problem 1.126 (Tableau Calculus) 14pt

Prove or refute that the following proposition is valid using a tableaux:

$$(P \Rightarrow Q) \vee R \Leftrightarrow \neg R \wedge Q \Rightarrow S$$

Solution:

$$\begin{array}{c}
(P \Rightarrow Q) \vee R \Leftrightarrow \neg R \wedge Q \Rightarrow S^{\mathsf{F}} \\
\begin{array}{c|c}
\begin{array}{c}
(P \Rightarrow Q) \vee R^{\mathsf{T}} \\
\neg R \wedge Q \Rightarrow S^{\mathsf{F}} \\
\neg R \wedge Q^{\mathsf{T}} \\
S^{\mathsf{F}} \\
\neg R^{\mathsf{T}} \\
Q^{\mathsf{T}} \\
R^{\mathsf{F}} \\
\begin{array}{c|c}
\begin{array}{c} P \Rightarrow Q^{\mathsf{T}} \\ \begin{array}{c|c} P^{\mathsf{F}} & Q^{\mathsf{T}} \end{array} \end{array} & \begin{array}{c} R^{\mathsf{T}} \\ \bot \end{array}
\end{array}
\end{array}
&
\begin{array}{c}
(P \Rightarrow Q) \vee R^{\mathsf{F}} \\
\neg R \wedge Q \Rightarrow S^{\mathsf{T}} \\
P \Rightarrow Q^{\mathsf{F}} \\
R^{\mathsf{F}} \\
P^{\mathsf{T}} \\
Q^{\mathsf{F}} \\
\begin{array}{c|c}
\begin{array}{c} \neg R \wedge Q^{\mathsf{F}} \\ R^{\mathsf{T}} \\ \bot \end{array} & S^{\mathsf{T}}
\end{array}
\end{array}
\end{array}
\end{array}$$

Based on the tableaux we find the following assignments that make the expression false therefore it is not valid:

1. $\varphi := \{P \mapsto \mathsf{T}, Q \mapsto \mathsf{T}, R \mapsto \mathsf{F}, S \mapsto \mathsf{F}\}$
2. $\varphi := \{P \mapsto \mathsf{F}, Q \mapsto \mathsf{T}, R \mapsto \mathsf{F}, S \mapsto \mathsf{F}\}$
3. $\varphi := \{P \mapsto \mathsf{T}, Q \mapsto \mathsf{F}, R \mapsto \mathsf{F}, S \mapsto \mathsf{T}\}$

Problem 1.127 (A *Nor* Tableau Calculus) 10pt 15min

Develop a variant of the tableau calculus presented in class for propositional formulae expressed with $\downarrow$ (i.e. "not or") as the only logical connective.

Complete the following scheme of inference rules for such a tableau calculus and proof its correctness

$$\frac{\mathbf{A}\downarrow\mathbf{B}^{\mathsf{T}}}{?}\qquad\frac{\mathbf{A}\downarrow\mathbf{B}^{\mathsf{F}}}{?}\qquad\frac{\begin{array}{l}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\end{array}\quad\alpha\neq\beta}{\perp}$$

Prove the formula $(P\downarrow(P\downarrow P))\downarrow(P\downarrow(P\downarrow P))$ in your new tableau calculus.

Solution: The completed *Nor*-tableau calculus is the following.

$$\frac{\mathbf{A}\downarrow\mathbf{B}^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{F}}\\ \mathbf{B}^{\mathsf{F}}\end{array}}\qquad\frac{\mathbf{A}\downarrow\mathbf{B}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}}\;\Big|\;\mathbf{B}^{\mathsf{T}}}\qquad\frac{\begin{array}{l}\mathbf{A}^{\alpha}\\ \mathbf{A}^{\beta}\end{array}\quad\alpha\neq\beta}{\perp}$$

And the proof of the formula is

$$\begin{array}{c|c}\multicolumn{2}{c}{(P\downarrow(P\downarrow P))\downarrow(P\downarrow(P\downarrow P))^{\mathsf{F}}}\\ P\downarrow(P\downarrow P)^{\mathsf{T}} & P\downarrow(P\downarrow P)^{\mathsf{T}}\\ P^{\mathsf{F}} & P^{\mathsf{F}}\\ P\downarrow P^{\mathsf{F}} & P\downarrow P^{\mathsf{F}}\\ P^{\mathsf{T}} & P^{\mathsf{T}}\\ \perp & \perp\end{array}\qquad(1)$$

Problem 1.128 **(Tableau Construction)** 35pt

Write an SML function that computes a complete tableau for a labeled formula. Use the data type `prop` for formulae and the datatype `tableau` for tableaux.

```
datatype prop = tru | fals (* true and false *)
              | por of prop * prop (* disjunction *)
              | pand of prop * prop (* conjunction *)
              | pimpl of prop * prop (* implication *)
              | piff of prop * prop (* biconditional *)
              | pnot of prop (* negation *)
              | var of int (* variables *)
datatype label = prove | refute
datatype tableau = ext of prop * label * tableau (* extension by a formula *)
                 | cases of tableau * tableau (* two branches *)
                 | complete (* branch completehalt *)
```

Hint: Write a recursive function `ctab` that takes a list of (unresolved) proposition/label pairs as an input, goes through them, extending the tableau as needed.

Solution: We proceed like the hint tells us. The main idea is to write a large case distinction; one for every rule.

```
fun ctab (nil) = complete
  | ctab ((tru,prove)::PL) = ext(tru,prove,ctab(PL))
  | ctab ((tru,refute)::PL) = ext(tru,refute,ctab((fals,prove)::PL))
  | ctab ((fals,prove)::PL) = ext(fals,prove,ctab(PL))
  | ctab ((fals,refute)::PL) = ext(fals,refute,ctab((tru,prove)::PL))
  | ctab ((var(X),L)::PL) = ext(var(X),L,ctab(PL))
  | ctab ((por(X,Y),prove)::PL) = ext(por(X,Y),prove,
                                  cases(ctab((X,prove)::PL),
                                        ctab((Y,prove)::PL)))
  | ctab ((por(X,Y),refute)::PL) = ext(por(X,Y),refute,
                                   ctab((X,refute)::(Y,refute)::PL))
  | ctab ((pand(X,Y),prove)::PL) = ext(pand(X,Y),prove,
                                   ctab((X,prove)::(Y,prove)::PL))
  | ctab ((pand(X,Y),refute)::PL) = ext(pand(X,Y),refute,
                                    cases(ctab((X,refute)::PL),
                                          ctab((Y,refute)::PL)))
  | ctab ((pimpl(X,Y),prove)::PL) = ext(pimpl(X,Y),prove,
                                    cases(ctab((X,refute)::PL),
                                          ctab((Y,prove)::PL)))
  | ctab ((pimpl(X,Y),refute)::PL) = ext(pimpl(X,Y),refute,
                                     ctab((X,prove)::(Y,refute)::PL))
  | ctab ((piff(X,Y),prove)::PL) = ext(piff(X,Y),prove,
                                   cases(ctab((X,refute)::(Y,refute)::PL),
                                         ctab((X,prove)::(Y,prove)::PL)))
  | ctab ((piff(X,Y),refute)::PL) = ext(piff(X,Y),prove,
                                    cases(ctab((X,prove)::(Y,refute)::PL),
                                          ctab((X,refute)::(Y,prove)::PL)))
  | ctab ((pnot(X),prove)::PL) = ext(pnot(X),prove,ctab((X,refute)::PL))
  | ctab ((pnot(X),refute)::PL) = ext(pnot(X),refute,ctab((X,prove)::PL));
```

Problem 1.129 (Automated Theorem Prover) 30pt

Building on the tableau procedure from ?? build an automated theorem prover for propositional logic. Concretely build an SML function `prove` that given a formula F outputs `valid`, if F is valid, and returns a counterexample otherwise (i.e. an interpretation of the variables that satisfy F^{T}).

Solution: Given a formula F, we have to examine the refutation tableau constructed by `ctab` to see if it is closed. The first step is to write a function that detects contradictions on a list of positive and negative literals

```
fun exists (_,nil) = false
  | exists (f,h::t) = f(h) orelse exists(f,t);

fun contradiction (pos,neg) =
    exists ((fn (x) => exists( (fn (y) => x = y), pos)), neg);
```

Then we build some infrastructure for outputting interpretations anything will do.

```
fun to_string(v) = if v<0 then
                     "~"^to_string((~1)*v)
                   else
                     if v>9 then
                       to_string(v div 10)^to_string(v mod 10)
                     else
                       implode([chr(v+48)]);
 fun istring(nil) = ""
  | istring((x,true)::t) = to_string(x)^"=true, " ^ istring(t)
  | istring((x,false)::t) = to_string(x)^"=false, " ^ istring(t);
fun interpretation (pos,neg) =
      (map (fn (x) => (x,true)) pos) @ (map (fn (x) => (x,false)) neg)
```

building on this, we walk the tableau and see whether all the branches are closed.

```
exception invalid;

fun closed (complete,pos,neg) = (*check at the leaves *)
    if (contradiction(pos,neg)) then true
    else
      let
      in
      print (istring(interpretation(pos,neg)));
      raise invalid
      end
  | closed (ext(var(n),prove,t),pos,neg) = closed(t,n::pos,neg)
  | closed (ext(var(n),refute,t),pos,neg) = closed(t,pos,n::neg)
  | closed (ext(tru,refute,_),pos,neg) = true
  | closed (ext(fals,prove,_),pos,neg) = true
  | closed (ext(_,_,t),pos,neg) = closed(t,pos,neg) (* only need literals *)
  | closed (cases(X,Y),pos,neg) = closed(X,pos,neg) andalso closed(Y,pos,neg);
```

Now, the function `prove` can be built by collecting the pieces.

```
  fun prove(X) = closed(ctab[(X,refute)],nil,nil)
```

Problem 1.130 (Testing the ATP) 30pt

Use the random formula generators from ?? to test your tableau implementation. Run experiments on large sets (e.g. 100) of random formulae with differing depths and plot the runtimes, percentages of valid formulae, over depths, and weights, and variable numbers. Interpret the results briefly.

Hint: You can use any plotting software you are familiar with, e.g. Excel or gnuplot. If you are not familiar with any, use pen and paper. Do not waste time on the plotting aspect.

G

1.8.2 Resolution for Propositional Logic

10pt

Problem 1.131: Compute the Clause normal form of $(P \Leftrightarrow Q) \Leftrightarrow (R \Leftrightarrow P)$ with and without using the derived rules.

Problem 1.132: Prove in the resolution calculus using derived rules:

$$\models A \wedge (B \vee C) \Rightarrow (A \wedge B \vee A \wedge C)$$

Solution: Clause Normal Form transformation

$$\cfrac{\cfrac{\cfrac{A \wedge (B \vee C) \Rightarrow (A \wedge B \vee A \wedge C)^{\mathsf{F}}}{A \wedge (B \vee C)^{\mathsf{T}}; A \wedge B \vee A \wedge C^{\mathsf{F}}}}{A^{\mathsf{T}}; B^{\mathsf{T}} \vee C^{\mathsf{T}}; A \wedge B^{\mathsf{F}}; A \wedge C^{\mathsf{F}}}}{A^{\mathsf{T}}; B^{\mathsf{T}} \vee C^{\mathsf{T}}; A^{\mathsf{F}} \vee B^{\mathsf{F}}; A^{\mathsf{F}} \vee C^{\mathsf{F}}}$$

Resolution Proof

1	A^{T}	initial
2	$B^{\mathsf{T}} \vee C^{\mathsf{T}}$	initial
3	$A^{\mathsf{F}} \vee B^{\mathsf{F}}$	initial
4	$A^{\mathsf{F}} \vee C^{\mathsf{F}}$	initial
5	B^{F}	with 1 and 3
6	C^{F}	with 1 and 4
7	C^{T}	with 2 and 5
8	$\square$	with 6 and 7

Problem 1.133 (Basics of Resolution) 4pt 8min

What are the principal steps when you try to prove the validity of a propositional formula by means of resolution calculus? In case you succeed deriving the empty clause, why does this mean you have found a proof for the validity of the initial formula?

Problem 1.134 (Resolution Calculus with Nand Connective) 5pt 10min

Develop a variant *PropCNFCalcNAND* of the CNF transformation calculus presented in class that transforms propositional formulae expressed with *NAND* (denoted by $\uparrow$) as the only logical connective. To do so just complete the scheme of inference rules given here:

$$\frac{\mathbf{C} \vee \mathbf{A} \uparrow \mathbf{B}^{\mathsf{T}}}{?} \qquad \frac{\mathbf{C} \vee \mathbf{A} \uparrow \mathbf{B}^{\mathsf{F}}}{?}$$

With this variant $\mathcal{CNF}^{\uparrow}$ together with the usual inference rule from resolution calculus conduct a resolution proof to verify the formula $(A \uparrow A) \uparrow ((A \uparrow B) \uparrow (A \uparrow B))$

Solution:

$$\frac{\mathbf{C} \vee \mathbf{A} \uparrow \mathbf{B}^{\mathsf{T}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{F}} \vee \mathbf{B}^{\mathsf{F}}} \qquad \frac{\mathbf{C} \vee \mathbf{A} \uparrow \mathbf{B}^{\mathsf{F}}}{\mathbf{C} \vee \mathbf{A}^{\mathsf{T}}; \mathbf{C} \vee \mathbf{B}^{\mathsf{T}}}$$

Problem 1.135: Use the resolution method to prove the formulae from ??: 25pt

1. $(\neg P \Rightarrow Q) \Rightarrow ((P \Rightarrow Q) \Rightarrow Q)$
2. $(P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow \neg(\neg R \wedge P)$

You may use any derived correctly derived inference rules such as for instance:

$$\frac{\mathbf{A} \Rightarrow \mathbf{B}^{\mathsf{F}}}{\begin{array}{c}\mathbf{A}^{\mathsf{T}} \\ \mathbf{B}^{\mathsf{F}}\end{array}}$$

However, if you use more complex inference rules (i.e. more than one connective involved) then you have to prove your derived inference rule.

Solution:

$$\begin{array}{c}
((\neg P \Rightarrow Q) \Rightarrow ((P \Rightarrow Q) \Rightarrow Q))^{\mathsf{F}} \\
(\neg P \Rightarrow Q)^{\mathsf{T}}((P \Rightarrow Q) \Rightarrow Q)^{\mathsf{F}} \\
P^{\mathsf{T}} \vee Q^{\mathsf{T}}; (P \Rightarrow Q)^{\mathsf{F}} \vee Q^{\mathsf{T}} \\
P^{\mathsf{T}} \vee Q^{\mathsf{T}}; (P \Rightarrow Q)^{\mathsf{T}}; Q^{\mathsf{T}} \\
P^{\mathsf{T}} \vee Q^{\mathsf{T}}; P^{\mathsf{F}} \vee Q^{\mathsf{T}}; Q^{\mathsf{T}}
\end{array}$$

for the conversion to clause normal form, so we have

resolution derivation

$$\begin{array}{c|c}
Q^{\mathsf{T}} & \textit{initial} \\
P^{\mathsf{T}} \vee Q^{\mathsf{F}} & \textit{initial} \\
P^{\mathsf{F}} \vee Q^{\mathsf{F}} & \textit{initial} \\
P^{\mathsf{T}} & \textit{resolved} \\
P^{\mathsf{F}} & \textit{resolved} \\
\Box &
\end{array}$$

For the second part we proceed similarly

$$\begin{array}{c}
((P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow \neg(\neg R \wedge P))^{\mathsf{F}} \\
(P \Rightarrow Q) \wedge (Q \Rightarrow R)^{\mathsf{T}}; \neg(\neg R \wedge P)^{\mathsf{F}} \\
(P \Rightarrow Q)^{\mathsf{T}}; (Q \Rightarrow R)^{\mathsf{T}}; \neg(\neg R \wedge P)^{\mathsf{F}} \\
P^{\mathsf{F}} \vee Q^{\mathsf{T}}; Q^{\mathsf{F}} \vee R^{\mathsf{T}}; \neg R^{\mathsf{T}}; P^{\mathsf{T}} \\
P^{\mathsf{F}} \vee Q^{\mathsf{T}}; Q^{\mathsf{F}} \vee R^{\mathsf{T}}; R^{\mathsf{F}}; P^{\mathsf{T}}
\end{array}$$

and then the resolution proof

$$\begin{array}{cc}
P^{\mathsf{F}} \vee Q^{\mathsf{T}} & \textit{initial} \\
Q^{\mathsf{F}} \vee R^{\mathsf{T}} & \textit{initial} \\
R^{\mathsf{F}} & \textit{initial} \\
P^{\mathsf{T}} & \textit{initial} \\
Q^{\mathsf{T}} & \textit{resolved} \\
Q^{\mathsf{F}} & \textit{resolved} \\
\Box & \textit{resolved}
\end{array}$$

Problem 1.136: Consider the following two formulae where the first one is in conjunctive normal form and the second in disjunctive normal form 25pt

1. $(P \vee \neg P) \wedge (Q \vee \neg Q)$
2. $P \wedge Q \vee (\neg P \vee \neg Q)$

Try to find the shortest proofs of both formulae using the resolution method as well as the tableau method. Describe your observations concerning the proof length in dependency on the normal form and proof method.

Solution: For the first forumla we have the tableau

$$\begin{array}{c|c}
\multicolumn{2}{c}{(P \vee \neg P) \wedge (Q \vee \neg Q)^{\mathsf{F}}} \\
P \vee \neg P^{\mathsf{F}} & Q \vee \neg Q^{\mathsf{F}} \\
P^{\mathsf{F}} & Q^{\mathsf{F}} \\
\neg P^{\mathsf{F}} & \neg Q^{\mathsf{F}} \\
P^{\mathsf{T}} \bot & Q^{\mathsf{T}} \bot
\end{array}$$

For the resolution proof we first have to convert into clause normal form.

$$\begin{array}{c}
((P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow \neg(\neg R \wedge P))^{\mathsf{F}} \\
(P \Rightarrow Q) \wedge (Q \Rightarrow R)^{\mathsf{T}}; \neg(\neg R \wedge P)^{\mathsf{F}} \\
(P \Rightarrow Q)^{\mathsf{T}}; (Q \Rightarrow R)^{\mathsf{T}}; \neg(\neg R \wedge P)^{\mathsf{F}} \\
P^{\mathsf{F}} \vee Q^{\mathsf{T}}; Q^{\mathsf{F}} \vee R^{\mathsf{T}}; \neg R^{\mathsf{T}} P^{\mathsf{T}} \\
P^{\mathsf{F}} \vee Q^{\mathsf{T}}; Q^{\mathsf{F}} \vee R^{\mathsf{T}}; R^{\mathsf{F}}; P^{\mathsf{T}}
\end{array}$$

then we have the resolution derivation

$$\begin{array}{c|c}
P^{\mathsf{F}} \vee Q^{\mathsf{T}} & \textit{initial} \\
Q^{\mathsf{F}} \vee R^{\mathsf{T}} & \textit{initial} \\
R^{\mathsf{F}} & \textit{initial} \\
P^{\mathsf{T}} & \textit{initial} \\
Q^{\mathsf{T}} & \textit{resolved} \\
Q^{\mathsf{F}} & \textit{resolved} \\
\Box & \textit{resolved}
\end{array}$$

Now to the next formula; here we have the tableau

$$\begin{array}{c|c}
\multicolumn{2}{c}{P \wedge Q \vee (\neg P \vee \neg Q)^{\mathsf{F}}} \\
\multicolumn{2}{c}{P \wedge Q^{\mathsf{F}}} \\
\multicolumn{2}{c}{\neg P \vee \neg Q^{\mathsf{F}}} \\
\multicolumn{2}{c}{\neg P^{\mathsf{F}}} \\
\multicolumn{2}{c}{\neg Q^{\mathsf{F}}} \\
\multicolumn{2}{c}{P^{\mathsf{T}}} \\
\multicolumn{2}{c}{Q^{\mathsf{T}}} \\
p^{\mathsf{F}} & Q^{\mathsf{F}} \\
\bot & \bot
\end{array}$$

For the resolution proof we convert to clause normal form:

$$\begin{array}{c}
(P \vee \neg P) \wedge (Q \vee \neg Q)^{\mathsf{F}} \\
(P \vee \neg P)^{\mathsf{F}} \vee (Q \vee \neg Q)^{\mathsf{F}} \\
P^{\mathsf{F}} \vee (Q \vee \neg Q); \neg P (Q \vee \neg Q)^{\mathsf{F}} \\
P^{\mathsf{F}} \vee Q^{\mathsf{F}}; P^{\mathsf{F}} \vee Q^{\mathsf{T}}; P^{\mathsf{T}} \vee Q^{\mathsf{F}}; P^{\mathsf{T}} \vee Q^{\mathsf{T}}
\end{array}$$

So we have the resolution derivation

$$\begin{array}{c|c}
P^{\mathsf{F}} \vee Q^{\mathsf{F}} & \textit{initial} \\
P^{\mathsf{F}} \vee Q^{\mathsf{T}} & \textit{initial} \\
P^{\mathsf{T}} \vee Q^{\mathsf{F}} & \textit{initial} \\
P^{\mathsf{T}} \vee Q^{\mathsf{T}} & \textit{initial} \\
Q^{\mathsf{T}} & \textit{resolved} \\
Q^{\mathsf{F}} & \textit{resolved} \\
\Box & \textit{resolved}
\end{array}$$

We note that for the formula in DNF the shortest method is the tableaux and for the one in CNF it is the resolution method. This is not particularly surprising, since the Resolution medthod is CNF-based (we construct the CNF in for clause normal form first), whereas Tableau is DNF-based.

2 How to build Computers and the Internet (in principle)

2.1 Circuits

2.1.1 Graphs and Trees

Conjecture 6 *Let G be a graph with a cycle and $n \in \mathbb{N}$, then there is a path p in G with $length(p) > n$.*

20pt

Problem 2.1 (Infinite Paths)
Prove or refute ?? using the formal definitions (no, it is not sufficient to just draw a picture).

Problem 2.2 (Node Connectivity Relation is an Equivalence Relation) 25pt

Let $G = \langle V, E\rangle$ be an undirected graph and the relation C be defined as

$$C := \{\langle u, v\rangle \mid \text{there is a path from } u \text{ to } v\}$$

Prove or refute that C is an equivalence relation.

Hint: Recall the properties of an equivalence relation!

Solution:

Proof:

P.1.1 reflexivity:From every node, there is a zero-length path to itself. □

P.1.2 symmetry:If there is a path from a node u to a node v, just reverse it. □

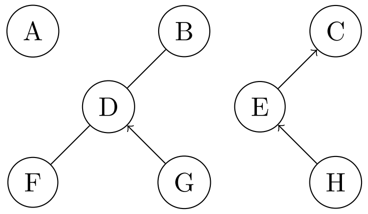
P.1.3 transitivity:If there is a path from u to v and a path from v to w, we can reach w from u via v. □

□

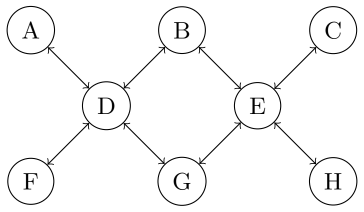
Problem 2.3 (Directed Graph)
We call a graph connected, iff for any two nodes n_1 and n_2 there is a path starting at n_1 and ending at n_2.

Complete the partially directed graph below by adding directed edges or directing undirected edges such that it becomes a connected, (fully) directed graph where each $indeg(n) = outdeg(n)$ for all nodes n.

How many initial, terminal nodes and how many paths does your graph have?



Solution: This graph has neither a initial nor a terminal node and infinite many paths since it is cyclic.



Problem 2.4: Draw examples of 4pt

1. a directed graph with 4 nodes and 6 edges
2. a undirected graph with 7 nodes and 8 edges.

Present a mathematical representation of these graphs.

Solution: Any graph will do, for instance the following ones (we only give the mathematical formulation here, go draw them yourselves)

- $\langle\{a,b,c,d\},\{\langle a,b\rangle,\langle b,c\rangle,\langle c,d\rangle,\langle d,a\rangle,\langle a,c\rangle,\langle b,d\rangle\}\rangle$
- $\langle\{a,b,c,d,e,f,g\},\{\{a,b\},\{b,c\},\{c,d\},\{d,e\},\{e,f\}\{f,g\},\{g,a\},\{a,c\},\{c,e\}\}\rangle$

Problem 2.5 (Planar Graphs)
A graph G is called planar if G can be drawn in the plane in such a manner that edges do not cross elsewhere than vertices. The geometric realization of a planar graph gives rise to regions in the plane called faces; if G is a finite planar graph, there will be one unbounded (i.e. infinite) face, and all other faces (if there are any) will be bounded. Given a planar realization of the graph G, let $v = \#(V)$, $e = \#(E)$, and let f be the number of faces (including the unbounded face) of G's realization.

Prove or refute the Euler formula, i.e. that $v - e + f = 2$, must hold for a connected planar graph.

Solution:

Proof: Proof by induction on the number of faces

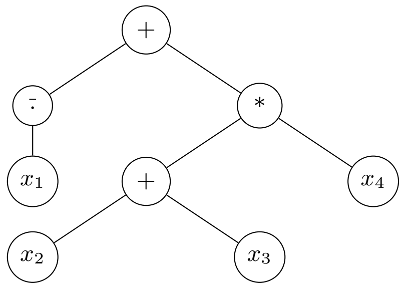
P.1 If G has only one face, it is acyclic and connected, so it is a tree and $e = v - 1$. Thus $v - e + f = 2$.

P.2 Otherwise, choose an edge e connecting two different faces of G, and remove it; e can then only appear once in the boundary of each face, so the graph remains connected – any path involving e can be replaced by a path around the other side of one of the two faces. This removal decreases both the number of faces and edges by one, and the result then holds by induction. □

Problem 2.6 (Parse trees and isomorphism)
Let P_e be the parse-tree of $e := \overline{x_1} + (x_2 + x_3) * x_4$

1. Draw the graphic representation of P_e.
2. Write the mathematical representation of a graph G that is different but equivalent to P_e.

Solution:

1. 

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

Problem 2.7 (Size and Depth of a Binary Tree) 3pt

Given the following data type for binary trees, define functions `size` and `depth` that compute the depth and the size of a given tree.

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

Write a function `fbbtree` that given a natural number n returns a fully balanced binary tree of depth n

Solution:

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

Problem 2.8 (Graph basics)
For each of the five directed graphs below do the following:

- State whether the graph is also a tree and explain why.
- Determine the depth of the graph.
- Write out in math notation a path from A to E if one exists and determine the path's length.

1. $G_1 := \langle\{A,B,C,D,E\},\{\langle A,B\rangle,\langle A,C\rangle,\langle A,D\rangle,\langle D,E\rangle\}\rangle$
2. $G_2 := \langle\{A,B,C,D,E\},\{\langle A,B\rangle,\langle B,C\rangle,\langle C,A\rangle,\langle C,D\rangle,\langle C,E\rangle\}\rangle$
3. $G_3 := \langle\{A,B,C,D,E\},\{\langle A,B\rangle,\langle B,C\rangle,\langle B,D\rangle,\langle C,E\rangle\}\rangle$
4. $G_4 := \langle\{A,B,C,D,E\},\{\langle A,B\rangle,\langle A,C\rangle,\langle B,D\rangle,\langle D,C\rangle,\langle C,B\rangle,\langle A,D\rangle\}\rangle$
5. $G_5 := \langle\{A,B,C,D,E\},\{\langle D,A\rangle,\langle D,B\rangle,\langle D,E\rangle,\langle D,C\rangle\}\rangle$

Solution:

1. - Yes, because it has no cycles and the graph is connected.
   - 2
   - $\langle A,D,E\rangle$ - length 2
2. - No, because it has a cycle - $\langle A,B,C,A\rangle$.
   - infinite
   - $\langle A,B,C,E\rangle$ - length 3
3. - Yes, because it has no cycles and the graph is connected.
   - 3
   - $\langle A,B,C,E\rangle$ - length 3
4. - No, because it has a cycle - $\langle B,D,C,B\rangle$.
   - infinite
   - E is not reachable from A, since $indeg(E) = 0$ i.e. E is a source.
5. - Yes, because it has no cycles and the graph is connected.
   - 1
   - E is not reachable from A, since $outdeg(A) = 0$ i.e. A is a sink.

Conjecture 7 *1. Let $G = \langle V, E\rangle$ be a directed graph. Then,*

$$\sum_{i=1}^{\#(V)} indeg(v_i) = \sum_{i=1}^{\#(V)} outdeg(v_i) = \#(E)$$

2. If G is undirected, we have

$$\sum_{i=1}^{\#(V)} deg(v_i) = 2 \cdot \#(E)$$

25pt

Problem 2.9 (Degrees in an Undirected Graph)

Prove or refute the conjecture above

Note: For undirected graphs, we introduce the notation deg with $\deg(v) = \text{indeg}(v) = \text{outdeg}(v)$ for each node.

Hint: Use induction over the number of edges. Derive the second assertion from the first one.

Solution:

Proof: by induction over $m = \#(E)$

P.1.1 $m = 0$ **(base case)**:For graphs that only consist of isolated nodes, both assertions hold trivially. □

P.1.2 $m \to m + 1$ **(induction step)**:

P.1.2.1 If we remove an arbitrary edge $e \in E$ from G, we obtain $G\backslash\{e\}$

P.1.2.2 $G\backslash\{e\}$ is a directed (or undirected, resp.) graph with m edges.

P.1.2.3.1 directed graph:By removing one edge, we have decreased the sum of in-degrees as well as the sum of out-degrees by one. □

P.1.2.3.2 undirected graph:By removing one edge $e = \langle u, v\rangle$, we have decreased the degree of u as well as the degree of v by one and thus the sum of degrees by two. □

□

□

Problem 2.10 (Graph representation in memory)
How would you represent a graph in memory if you write a program which processes it in some way? Give 2-3 variants and explain the advantages and disadvantages of each method.

Solution:

1. **The adjacency matrix**. You represent a graph in a matrix (two-dimensional array). Each column and row corresponds to a vertex. In a matrix cell you put a number which denotes the presence (or absence) of an edge between two vertices. E.g. you can use 0 and 1 for undirected graphs, or -1, 0 and 1 for directed ones. Also such a number may represent a weight of an edge if needed.
   - **Advantages**: convenient to work with if our program frequently checks if there is an edge between two vertices.
   - **Disadvantages**: takes a lot of memory, especially when a graph contains a few edges and a lot of nodes.
2. **The incidence matrix**. You represent a graph also in a matrix, the rows correspond to vertices, the columns to edges. In the $m[i][j]$ you put 1 if $\langle a_i, v\rangle$ belongs to V, where v is an arbitrary vertex, -1 if $\langle v, a_i\rangle$belongs to V, and 0 if there is no v that $\langle a_i, v\rangle$ or $\langle v, a_i\rangle$ belongs to V.
   - **Advantages**: simplifies the finding cycles in a graph.
   - **Disadvantages**: when there is more edges than vertices then this method consumes even more memory than the previous one.
3. **List of pairs of numbers representing edges**.
   - **Advantages**: convenient when we mostly work with edges, but not with separate vertices. Requires less memory than the previous two.
   - **Disadvantages**: working with vertices is not so easy.
4. For each vertex v we have a list which contains the numbers of vertices the v vertex is connected to. but
   - **Advantages**: consumes less memory than 1.
   - **Disadvantages**: to find out whether vertex v is connected to vertex u we have to traverse the entire list for vertex v.

Problem 2.11: How many edges can a directed graph of size n (i.e. with n vertices) have maximally. How many can it have if it is acyclic? Justify your answers (prove them). 4pt

Solution:

Theorem 8 *A directed graph with n vertices has at most n^2 edges.*

Proof:

P.1 Let $G = \langle V, E \rangle$.

P.2 By definition $E \subseteq V^2$, so maximally $\#(V) = n^2$. □

Theorem 9 *A DAG with n vertices has at most $(n(n-1))/2$ edges.*

Proof: by induction on n

P.1.1 If $n = 1$:

P.1.1.1 then $G_1 = \langle \{c\}, E \rangle$, since the only possible edge $\langle c, c \rangle$ is a cycle. □

P.1.2 If $n > 1$:

P.1.2.1 Let $G_n = \langle V_n, E_n \rangle$ be a maximal DAG,

P.1.2.2 then the graph $G_{n-1} = \langle commiV_{n-1}, E_{n-1} \rangle$ which we obtain from G_n by deleting an arbitrary vertex c and all the edges c in must be a maximal DAG with $n-1$ nodes (otherwise we could add an edge to G_n).

P.1.2.3 Thus $\#(E_{n-1}) = ((n-1)(n-2))/2$. Now, there can be at most $n-1$ edges in V_n, which c occurs in without cycles (G_{n-1} has $n-1$ vertices).

P.1.2.4 Therefore, $\#(V_n) = ((n-1)(n-2))/2 + (n-1) = (n-1)(n-2) + (2(n-1))/2 = ((n-1)n)/2$. □

□

Problem 2.12 (Undirected tree properties) 4pt

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:

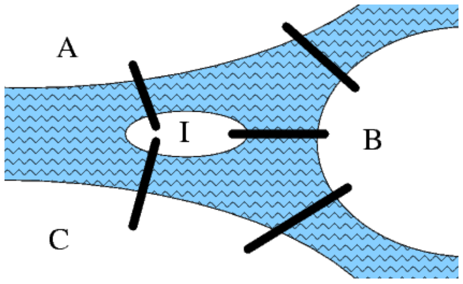
1. G is an undirected tree
2. For any two nodes $n_1 \neq n_2$ there is a *single* path starting at n_1 and ending at n_2
3. G is a connected graph, but it becomes disconnected after deleting any edge
4. G is connected and $\#(E) = \#(V) - 1$
5. G is acyclic and $\#(E) = \#(V) - 1$
6. G is acyclic, but 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 2.13 ((Modified) Königsberg Bridge Problem) 3pt 10min

Consider a river fork with three banks (A,B,C) and one island (I) connected with bridges as shown in the figure.



Is it possible to walk accross each of the bridges exactly once in an uninterrupted tour and return to the starting point?

In order to prove your answer first translate the question into a graph problem where the banks and the island are modeled as nodes and the bridges as undirected edges.

Hint: Consider the degree of each node (i.e.the number for edges connected to it). Relate the degrees of the nodes to the constraint of an uninterrupted tour.

Solution: If there is a round trip path which passes all edges of the graph once then each node must have an even degree, because whenever the path enters a node it must leave it again.

Since in the given bridge problem the nodes I and B have odd degree there is no such round trip.

$$\begin{tikzcd}
 & A \arrow[dl, no head] \arrow[dr, no head] & \\
I \arrow[rr, no head] \arrow[dr, no head] & & B \arrow[dl, no head] \\
 & C &
\end{tikzcd}$$

Problem 2.14 (Parse Tree) 35pt

Given the data type prop for formulae

```
datatype prop = tru | fals (* true and false *)
              | por of prop * prop (* disjunction *)
              | pand of prop * prop (* conjunction *)
              | pimpl of prop * prop (* implication *)
              | piff of prop * prop (* biconditional *)
              | pnot of prop (* negation *)
              | var of int (* variables *)
```

Write an SML function that computes the parse tree for a formula. The output format should be

- a list of integers for the set of vertices,
- a list of pairs of integers for the set of edges,
- and for the labeling function a list of pairs where the first component is an integer and the second a string (the label).

Solution:

```
datatype prop = tru | fals (* true and false *)
| por of prop * prop (* disjunction *)
| pand of prop * prop (* conjunction *)
| pimpl of prop * prop (* implication *)
| piff of prop * prop (* biconditional *)
| pnot of prop (* negation *)
| var of int (* variables *)
```

The output is a triple (`vertices`, `edges`, `labeling_list`) where

- `vertices` is simply an integer (so the vertices are represented as integers from 1 to that number)
- `edges` is a list of pairs of integers that are the vertices between which there are edges
- `labeling_list`: a list of (`vertex:integer`, `label:string`) pairs that will be used to make a labeling function which takes the index of a vertex and returns its label, e.g. a string `"impl"` Input: A pair (`root`, `p`)
- `root` is a name for the root node (an integer), see the function `maketree`
- p a variable of datatype `prop` in which a boolean expression is stored

```
fun totree(mroot, pnot(be)) = let val (verout, edges, lblpairs) = totree(mroot+1, be)
                                                    in (verout, [mroot, mroot+1] :: edges, (mroot, "-") :: lblp
                                                    end |
       totree(mroot, tru) = (mroot, nil, [(mroot, "T")])|
       totree(mroot, fals) = (mroot, nil, [(mroot, "F")])|
       totree(mroot, var v) = (mroot, nil, [(mroot, Int.toString(v))])|
       totree(mroot, two_var) = let val ax = fn (por(be1, be2)) => ("OR", be1, be2) |
                                                                          (pand(be1, be2)) => ("AND", be1
                                                                          (pimpl(be1, be2)) => ("=>", be1
                                                                          (piff(be1, be2)) => ("<=>", be1
                                                          val (lbl, be1, be2) = ax two_var
                                                          val (verout1, edges1, lblpairs1) = totree(mroot+1, be
                                                          val (verout2, edges2, lblpairs2) = totree(verout1+1,
                                                   in (verout2, [[mroot, mroot+1],[mroot, verout1+1]] @ edges1
                                                   end
```

Now we have an optional wrapper function that

- eliminates the need for the index of the root (default value is 1)
- converts the `labeling_list` into labeling function that takes a vertex and returns its label

```
local
       fun findString(num, hd::tl) = let
              val (a,b) = hd
       in
              if a = num then b
              else findString(num, tl)
       end

in
       fun maketree(t) = let
              val (lastnode, edges, lblpairs) = totree(1, t)
       in (lastnode, edges, fn num => findString(num, lblpairs))
       end
end
```

Finallly: an example of how the program can be tested:

```
totree(1, por(por(piff(pnot(var 4), fals), pimpl(tru, pand(var 2, var 3))), var 9));
val it =
(12,[[1,2],[1,12],[2,3],[2,7],[3,4],[3,6],[4,5],[7,8],[7,9],[9,10],[9,11]],
[(1,"OR"),(2,"OR"),(3,"<=>"),(4,"-"),(5,"4"),(6,"F"),(7,"=>"),(8,"T"),
(9,"AND"),(10,"2"),(11,"3"),(12,"9")])
: int * int list list * (int * string) list *)
```

2.1.2 Introduction to Combinatorial Circuits

Problem 2.15 (DNF Circuit with Quine McClusky) 25pt

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

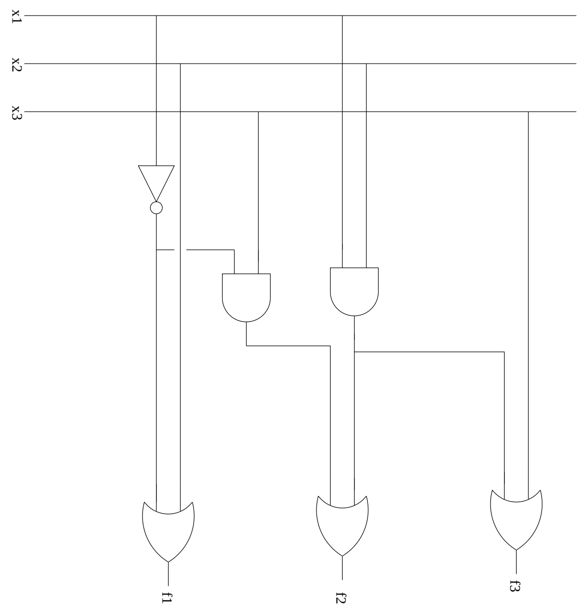
X_1	X_2	X_3	$f_1(X)$	$f_2(X)$	$f_3(X)$
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	1	1	1

Hint: Use Quine-McCluskey to compute minimal polynomials for the three component functions, look for shared monomials, and build the DNF circuit.

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

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

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



Problem 2.16 (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

Hint: Use Quine-McCluskey to compute minimal polynomials for the three component functions, look for shared monomials, and build the DNF circuit.

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 2.17 (Combinational Circuit) 12pt 10min

Consider the following Boolean function

$$f\colon \{0,1\}^3 \to \{0,1\}^2; \langle i_1, i_2, i_3\rangle \mapsto \langle \overline{i_1} * i_2 + i_2 * \overline{i_3}, \overline{i_1 + i_2} * i_3\rangle$$

Draw the corresponding combinational circuit and write down its labeled graph $G = \langle V, E, f_g\rangle$ in explicit math notation.

Solution: The solution is obvious. All we need is to just draw graph and write some line about math representation.

Problem 2.18 (Combinational Circuit for Shift) 5pt 10min

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

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

Hint: Think of a variant of multiplexer.

Solution:

Problem 2.19 (Is XOR universal?)
Imagine a logical gate XOR that computes the logical exclusive disjunction. Prove or refute whether the set $S = \{\text{XOR}\}$ 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 $G \in \{\text{AND}, \text{OR}, \text{NOT}\}$ to S, and prove that the set $S' = \{\text{XOR}, G\}$ 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. $\{\text{NAND}\}$ is an example from the lecture.

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

Problem 2.20 (Alarm System)

You have to devise an alarm system that signals if the image recorded by a camera changes. The camera is preprogrammed with a static image, divided into 8 regions. Whenever an observed region is different from the preprogrammed one, the corresponding input bit $\langle r_0, ..., r_7 \rangle$ is set to 1. The image is sampled at discrete time periods. The value of an input (clk) changes between 0 and 1 on every time interval.

Design a circuit with one output which is set to 1 if two or more regions (the inputs $\langle r_0, ..., r_7 \rangle$) are different from the preprogrammed image for two consecutive intervals. We do not care if different sets of regions are marked as different between the consecutive intervals. We also don't care what happens once the output is set to one.

You may use all elementary gates and all circuit blocks studied in class.

Hint:

- First make a circuit that determines how many of the regions are different.
- Make a circuit that outputs 1 if two or more regions are different in 2 consecutive intervals.

2.1.3 Realizing Complex Gates Efficiently

999pt

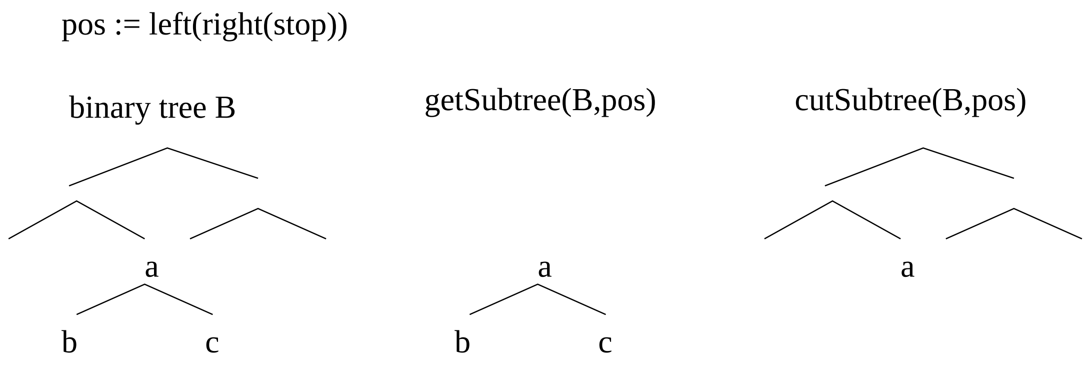
Balanced Binary Trees Problem 2.21 (Operations on Binary Trees)
Given the SML datatype `btree` for binary trees and `position` for a position pointer into a binary tree:

```
datatype btree = leaf | parent of btree * btree;
datatype position = stop | right of position | left of position;
```

The interpretation of a position `right(left(stop))` is like a reversed path: start from root follow the right branch then the left and then stop.

Write two SML functions:

- `getSubtree` that takes a binary tree and a position and returns the subtree of the that binary at the corresponding position.
- `cutSubtree` that takes a binary tree and a position and returns the binary tree where the subtree at the corresponding position is cut off; i.e replaced by a leaf.



In both cases an exception should be raised if the position exceeds the observed binary tree.

Solution:

```
datatype btree = leaf | parent of btree * btree;
datatype position = stop | right of position | left of position;

exception TreeExceeded;

fun getSubtree (parent(l, r), right(pos)) = getSubtree(r, pos)
  | getSubtree (parent(l, r), left(pos)) = getSubtree(l, pos)
  | getSubtree (tree, stop) = tree
  | getSubtree (_, _) = raise TreeExceeded;

fun cutSubtree (parent(l, r), right(pos)) = parent(l, cutSubtree(r, pos))
  | cutSubtree (parent(l, r), left(pos)) = parent(cutSubtree(l, pos), r)
  | cutSubtree (tree, stop) = leaf
  | cutSubtree (_, _) = raise TreeExceeded;

val testTree = parent(parent(leaf, parent(leaf, leaf)),parent(leaf, leaf));

val testPos = left(right(stop));

(* test get Subtree:
- getSubtree(testTree, testPos);
val it = parent (leaf,leaf) : btree
- cutSubtree(testTree, testPos);
val it = parent (parent (leaf,leaf),parent (leaf,leaf)) : btree
*)
```

Problem 2.22 (Number of Paths in Balanced Binary Tree) 4pt

Let $p(n)$ be the number of different paths in a fully balanced binary tree of depth n. Find a recursive equation for $p(n)$.

Hint: Do not forget the base case(s) for small n.

Solution: Base case: $p(0) = 0$ and $p(1) = 2$. Recursive rest for $n > 1$:

$$p(n) := n * 2^n + p(n - 1)$$

Problem 2.23 (Length of the inner path in balanced trees) 4pt

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 2.24 (Depth of a Fully Balanced Binary Tree) 10pt

Prove or refute that in a fully balanced binary tree with $n \geq 1$ nodes, the depth is $\log_2(n)$.

Solution:

Proof: by induction over n

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

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

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

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

□

Realizing n-ary Gates No problems supplied yet.

2.2 Arithmetic Circuits

2.2.1 Basic Arithmetics with Combinational Circuits

Problem 2.25 (Number System Conversion)

6pt **Positional Number Systems** 6min

Convert the following 12-bit twos complement numbers into hexadecimal and decimal numbers.

1. 1000 0101 0100
2. 0010 1000 1010
3. 1101 0101 1001

Problem 2.26 (Mapping between Positional Number Systems) 25pt

Show that the mapping $\psi\colon D^+ \to \{/\}^*$ from the definition of a positional number system is indeed a bijection.

Solution:

Proof:

P.1 $\varphi\colon D \to \{\varepsilon, /, //, \ldots, /^{[b-1]}\}$ is bijective by definition.

P.2 Then we have $\psi\colon D^+ \to \{/\}^*$ such that $\psi(\langle n_k, \ldots, n_1\rangle) := \oplus_{i=1}^{k}\left(\varphi(n_i) \odot (/^{[b]})^{i-1}\right)$ where D is an alphabet of digits.

P.3 To prove that ψ is an isomorphism we will show that is both $1-1$ and onto.

P.4 "$1-1$": Indeed, if we have $u = \langle n_k, \ldots, n_1\rangle$, $v = \langle m_r, \ldots, n_1\rangle$, $u, v \in D^+$ and $u \neq v$, i.e. there is a j such that $n_j \neq m_j$ or $k \neq r$ then $\psi(u) = \oplus_{i=1}^{j-1}\left(\varphi(n_i) \odot (/^{[b]})^{i-1}\right) + \varphi(n_j) \odot (/^{[b]})^{j-1} + \oplus_{i=j+1}^{k}\left(\varphi(n_i) \odot (/^{[b]})^{i-1}\right) \neq \oplus_{i=1}^{j-1}\left(\varphi(m_i) \odot (/^{[b]})^{i-1}\right) + \varphi(m_j) \odot (/^{[b]})^{j-1} + \oplus_{i=j+1}^{r}\left(\varphi(m_i) \odot (/^{[b]})^{i-1}\right) = \psi(v)$ since φ is bijective (i.e. different inputs to it give different outputs).

P.5 "onto": We define its inverse $\psi^{-1}\colon \{/\}^* \to D^+$ in the following way: Consider $U_1 = /^{[c]}$ be inputted and $U_i = U_{i-1} \oslash /^{[b]}$, $2 \leq i \leq k$, where $k = \lfloor \frac{c}{b} \rfloor + 1$, $U_x \in \{/\}^*$ for $1 \leq x \leq k$, where b is the base, and $\oslash\colon \{/\}^* \times \{/\}^* \to \{/\}^*$ integer division in the unary system defined the usual way. Then we will take our output to be $\langle n_k, \ldots, n_1\rangle \in D^+$ such that $n_i = \varphi^{-1}(U_i \div /^{[b]})$ for $1 \leq i \leq k$ where $\div$ is (similary defined to $\oslash$) defined as division by module in our unary system and φ^{-1} is the inverse to the above mentioned φ.

P.6 This concludes our proof. □

Problem 2.27 (Binary Number Conversion) 20pt

Write an SML function `binary` that converts decimal numbers into binary strings and an inverse `decimal` that converts binary strings into decimal numbers. Use the positive integers (of built-in type `int`) as a representation for decimal numbers. `binary` should raise an exception, if applied to a negative integer.

Solution:

```
exception NegInteger
fun binary 0 = "0" |
    binary 1 = "1" |
    binary n = if (n<0) raise NegInteger
                        else binary(n div 2)^(Int.toString(n mod 2))

fun decimal s = let
    fun bintodec nil = 0
      | bintodec sa = foldl (fn (x, y) => 2*y+(if x = #"0" then 0 else 1)) 0 sa
in bintodec(explode(s))
end
```

Problem 2.28 (Playing with bases)
Convert 2748 from decimal to hexadecimal, binary and octal representation.

Solution: Divide repeatedly 2748 by 16 and take the remainders - get ABC. $A = 1010$, $B = 1011$ and $C = 1100$, thus $ABC = 101010111100$. Starting from right to left convert every 3 bits to their corresponding octal value: 5274.

Problem 2.29 (Converting to decimal in SML)
Write an SML function

`to_int = fn : string -> int`

that takes a string in binary, octal or hexadecimal notation and converts it to a decimal integer. If the string represents a binary number, it begins with 'b' (e.g. "b1011"), if it is an octal number - with '0' (e.g. "075") and if it is a hexadecimal number it begins with '0x' (e.g. "0x3A").

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

For example we have

```
to_int("b101010") -> 42
```

Solution:

```
exception InvalidInput;

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

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

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

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

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

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

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

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

Adders Problem 2.30 (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 2.31 (Carry Chain and Conditional Sum Adder) 35pt

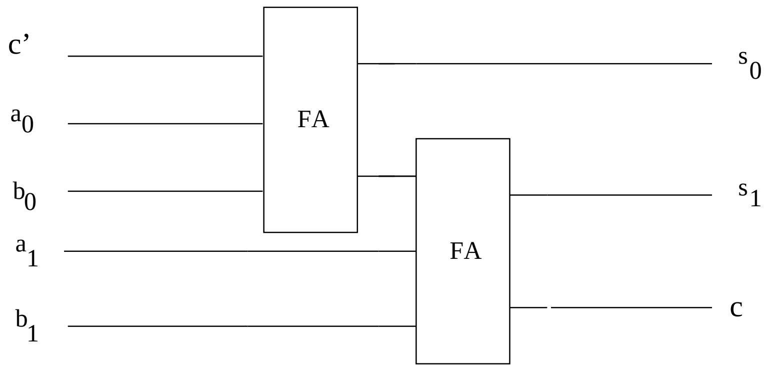
Draw an explicit combinational circuit of a 4-bit Carry Chain Adder and a 4-bit Conditional Sum Adder. Do not use abbreviations, but only NOT, AND, OR, XOR gates. Demonstrate the addition of the two binary numbers $\langle 1,0,1,1\rangle$ and $\langle 0,0,1,1\rangle$ on both adders; i.e. annotate the 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 2.32 (Carry Chain Adder and Subtractor for TCN) 4pt 10min

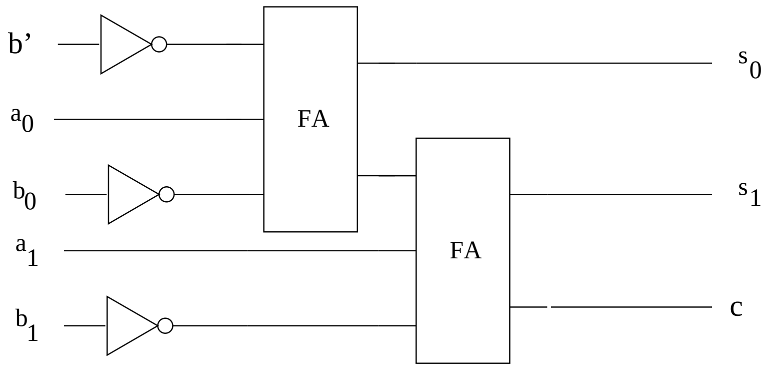
- Draw a 2-bit carry chain adder only using (1-bit) full adders.
- Draw a subtractor for two's complement numbers using (1-bit) full-adders and Boolean gates of your choice.

Hint: Remember: An n-bit subtractor $f^n_{\mathrm{SUB}}(a, b, b')$ can be implemented as n-bit full-adder $(\mathrm{FA}^n(a, \overline{b}, b'))$

Solution: 2 bit carry chain adder:



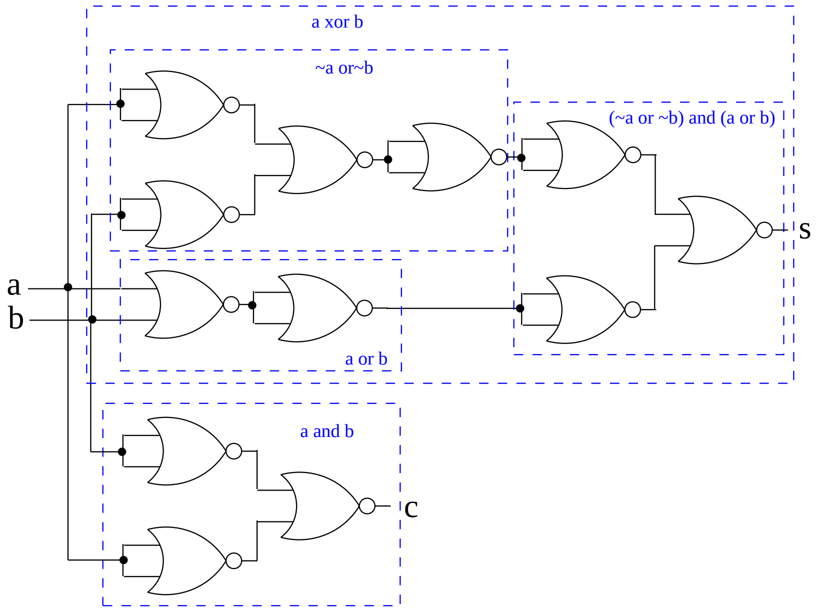
2 bit TCN subtractor:



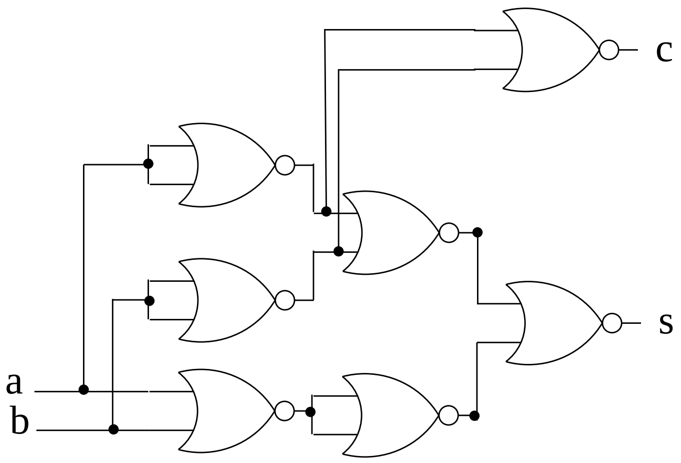
Problem 2.33 (Half Adder)
Design an explicit combinational circuit for the half-adder using only NOR gates. What is its cost and depth? Looking at the first, straightforward solution, can cost and depth be improved?

Hint: First express the XOR gate by AND, OR and NOT gates then express each of these gates by NOR gates. Then think of further improvements.

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 NOR gates. Solution steps: First express the XOR gate by AND, OR and NOT gates then express each of these gates by NOR gates. This process is sketched by the blue boxes. The result circuit has depth 5 and cost 12.



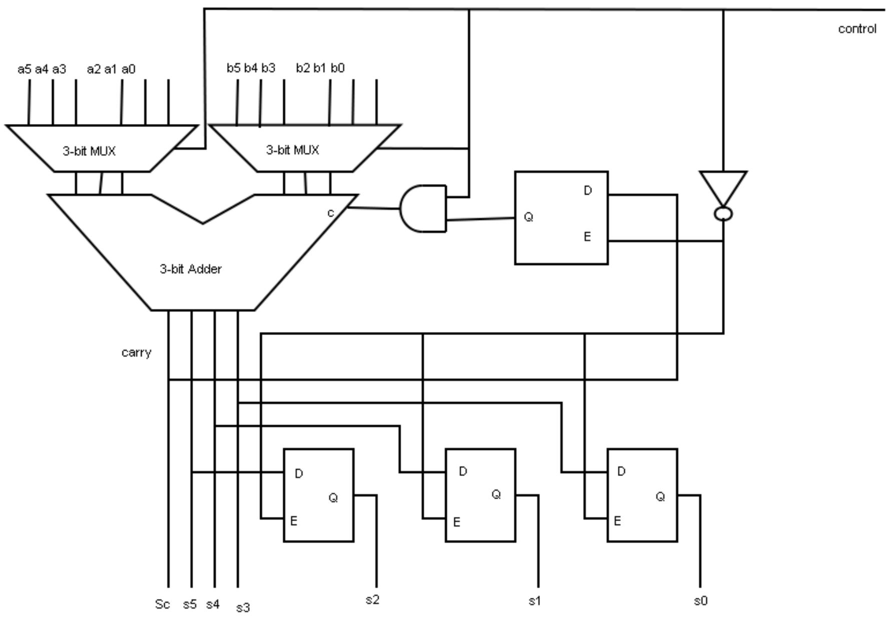
Removing double negations and a smarter placing of the circuit for the carrier output yields a circuit with depth 3 and cost 7.



Problem 2.34 (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.

Hint: Think about using the D Flip-Flop with an enable input to store intermediate data.

Solution:



2.2.2 Arithmetics for Two's Complement Numbers

Problem 2.35 (Binary Number Systems)

- Write down the definition of $\langle\!\langle\cdot\rangle\!\rangle$, $(\langle\!\langle\cdot\rangle\!\rangle^{-})$, and $\langle\!\langle n\rangle\!\rangle_{\cdot}^{2s}$.
- Given the binary number $a = 10110$ compute $\langle\!\langle a\rangle\!\rangle$, $(\langle\!\langle a\rangle\!\rangle^{-})$, and $\langle\!\langle a\rangle\!\rangle_{n}^{2s}$.

Problem 2.36 (Sign-and-Magnitude Adder)
Recall the naïve *sign and magnitude* representation for n-bit integers: If the sign bit is 0, the number is positive, else negative. The other $n-1$ bits represent the absolute value of the number.

1. Describe how to add two equally-signed n-bit numbers (simple).
2. Describe how to add two n-bit numbers numbers with different sign bits (a bit more tricky).
3. Draw a combinational circuit of a 4-bit sign and magnitude adder (one sign bit, three data bits). You may use the 1-bit full adder/subtractor (with one input that selects whether to add or to subtract) known from the lecture, an n-bit multiplexer that selects one of two n-bit numbers, as well as an n-bit comparator that computes the function $f\colon \{0,1\}^2 \to \{0,1\}$ defined as follows:
$$f(a,b) := \begin{cases} 1 & \text{if } a \leq b \\ ,0 & \text{else} \end{cases}$$
Be sure to explain the layout of your circuit.
4. How can an over-/underflow be detected at the outputs? In which cases can an over-/underflow occur?

Solution:

1. Add the absolute values and keep the sign
2. Choose the greater of the absolute values of the two numbers ($|a|$), subtract the absolute value of the other number ($|b|$) from it and take the sign of a.
3. (Wire the above.)
4. An over-/underflow can occur when two equally-signed numbers are added. It can be detected by checking the last carry-out.

Problem 2.37: Given following integer numbers in base ten. Convert them to 32-bit Two's Complement numbers. 12pt

1. 3643
2. 5731923
3. -128
4. -24689

Solution: First we need the numbers' normal binary representation, therefore[4]

1. $3643 = \varphi_{111000111011}(2)$
2. $5731923 = \varphi_{10101110111011001010011}(2)$
3. $-128 = -\varphi_{10000000}(2)$
4. $-24689 = -\varphi_{110000001110001}(2)$

To align a positive number to 32 bits we just fill in the space to the left with its sign bit (0), therefore

1.
2. $3643 = \langle\!\langle 2 \rangle\!\rangle^{2s}_{00000000000000000000111000111011}$
3. $5731923 = \langle\!\langle 2 \rangle\!\rangle^{2s}_{00000000010101110111011001010011}$

To convert a negative number we need to complement it and add 1 to the resulted number. Afterwards we will fill in the space on its left with its sign bit (1).

1. $-128 = \langle\!\langle 2 \rangle\!\rangle^{2s}_{11111111111111111111111110000000}$
2. $-24689 = \langle\!\langle 2 \rangle\!\rangle^{2s}_{1111111111111111001111110001111111}$

---

[4] EdNote: Check all of these.

Problem 2.38: Given the following integer numbers as 16-bit Two's Complement numbers. 9pt

1. 1010 0001 0100 0000
2. 0010 1110 1110 1110
3. 1101 0011 1111 0010

Convert them into decimal numbers.

Solution: We just have to sum up powers of 2, adding $-$ in front of the sign bit's power.

1. $1010000101000000 = 2^{-15} + 2^{13} + 2^{8} + 2^{6} = -24256$
2. $0010111011101110 = 2^{13} + 2^{11} + 2^{10} + 2^{9} + 2^{7} + 2^{6} + 2^{5} + 2^{3} + 2^{2} + 2^{1} = 12014$
3. $1101001111110010 = -2^{15} + 2^{14} + 2^{12} + 2^{9} + 2^{8} + 2^{7} + 2^{6} + 2^{5} + 2^{4} + 2^{1} = -11278$

Problem 2.39 (The Structure Theorem for TCN) 4pt 7min

Write down the structure theorem for two's complement numbers (TCN) and make use of it to convert

- the integer -53 into a 8-bit TCN.
- the 8-bit TCN 10110101 into an integer.

Furthermore convert

- the integer -53 into a 10-bit TCN.
- the 10-bit TCN 1110110101 into an integer.

The 10-bit version of the conversion task shouldn't be any effort after solving the 8-bit version. You just have to remember the appropriate lemma to transfer an n-bit TCN to an $n+1$-bit TCN. How is the lemma called and what does it state?

Solution:
The 8-bit version:

- $B(()53) = 00110101$, $\overline{B(53)} = 11001010$, $B(\langle\!\langle\overline{B(53)}\rangle\!\rangle + 1) = 11001011$
- $\overline{B(\langle\!\langle 10110101\rangle\!\rangle - 1)} = 01001011$, $-\langle\!\langle\overline{B(\langle\!\langle b\rangle\!\rangle - 1)}\rangle\!\rangle = -75$

The 10-bit version: We make use of the *sign bit duplication lemma*

- We just have prepend two Ïïn front to get from the 8-bit to the 10-bit version: $B(\langle\!\langle\overline{B(53)}\rangle\!\rangle + 1) =$ 1111001011
- With the argument reverted we see that 1110110101 is the 10-bit variant of 10110101, hence it represents the same integer as above namely -75.

Problem 2.40 (2s Complement Conversion) 15pt

Write an SML function `tcn` that takes an integer i and a natural number n as arguments and converts i into an n-bit two's complement number if it is in range and raises an exception otherwise.

Write an SML function that converts a 2s complement number into a decimal integer.

Solution:

```
exception OverFlow
exception UnderFlow

fun align(n, s, sb) = if String.size(s) < n
                    then Int.toString(sb)^align(n-1, s, sb)
                    else s

fun flipadd (nil, _) = ""
| flipadd (#"0"::t, 0) = flipadd(t, 0)^"0"
| flipadd (#"1"::t, 0) = flipadd(t, 1)^"1"
| flipadd (h::t, 1) = flipadd(t, 1)^(if h = #"0" then "1" else "0")


(* converts a decimal number into a TCN's binary string *)
fun tcn (i, n) =
if i>=0 then
   if String.size(binary(i)) < n then align(n, binary(i), 0)
   else raise OverFlow
else
   if String.size(binary(~i-1)) < n then
      align(n, flipadd(foldl op:: nil (explode(binary(~i))), 0), 1)
   else raise UnderFlow

(* converts a TCN's binary string into a decimal number *)
fun todec(s) = let
   fun exp 0 = 1 |exp n = 2*exp(n-1)
   fun dec(#"0"::t) = decimal(s) |
   dec(#"1"::t) = decimal(implode(t))-exp(String.size(s)-1)
in
   dec(explode(s))
end
```

Problem 2.41 (Shift and Duplication on PNS) 8pt 10min

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 2.42: Compute the intermediate carry ($\mathrm{ic}_k(9235, 26234, 1)$) for $k = 3$ and $k = 5$. 15pt

Hint: You have to convert the first two arguments to binary numbers of the same range beforehand.

Solution: The two binary representations of the two numbers are:

1. 9235 = 0010010000010011
2. 26234 = 0110011001111010

Now we can perform addition to get that $(\mathrm{ic}_3(())9235, 26234, 1) = 0$ and $(\mathrm{ic}_5(())9235, 26234, 1) = 1$.

2.2.3 Algorithmic/Logic Units

Problem 2.43 (TCN Substraction) 3pt 6min

Let $A = 576$ and $B = 9$.

1. convert the numbers into an n-bit TCN system. What is the minimal n in order to encode A as well as B?
2. perform a binary subtraction $A - B$ and check the result by converting back to the decimal system.

Solution: n should be 11. The 11-bit TCN representations are:

$$\langle\langle B(576)\rangle\rangle = 01001000000$$

$$\langle\langle B(9)\rangle\rangle = 00000001001$$

The subtraction $A - B$ in TCN representation is

$$\langle\langle B(576)\rangle\rangle - \langle\langle B(9)\rangle\rangle = \langle\langle B(576)\rangle\rangle + \langle\langle \overline{B(9)}\rangle\rangle + 1$$

This corresponds to

$$01001000000 - 00000001001 = 01001000000 + 11111110110 + 1 = 01000110111$$

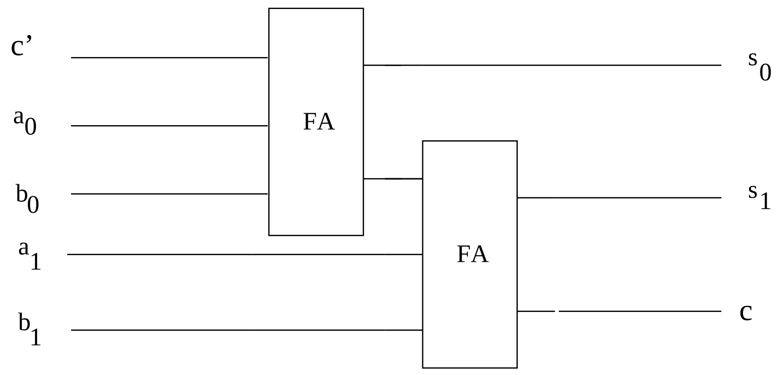
In fact: $\langle\langle B(576-9)\rangle\rangle = \langle\langle B(565)\rangle\rangle = 01000110111$

Problem 2.44 (Carry Chain Adder and Subtractor for TCN) 2pt 6min

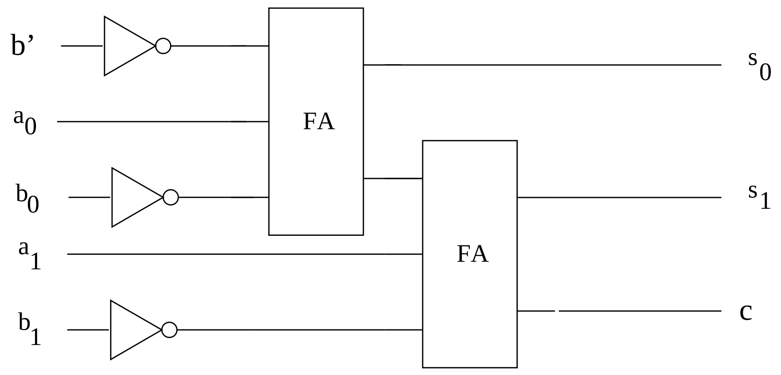
- Draw a 2-bit carry chain adder only using (1-bit) full adders as primitives.
- Draw a 2-bit subtractor for two's complement numbers using (1-bit) full-adders and Boolean gates of your choice.

Hint: Remember: An n-bit subtractor $f^n_{\text{SUB}}(a, b, b')$ can be implemented as n-bit full-adder $(\text{FA}^n(a, \bar{b}, \overline{b'}))$

Solution: 2 bit carry chain adder:



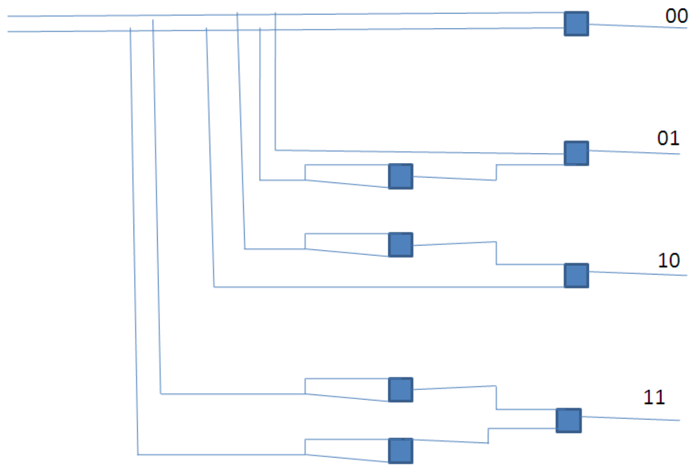
2 bit TCN subtractor:



2.3 Sequential Logic Circuits and Memory Elements

Problem 2.45 (2bit Address Decoder)
Design a 2 bit address decoder using only NOR gates.

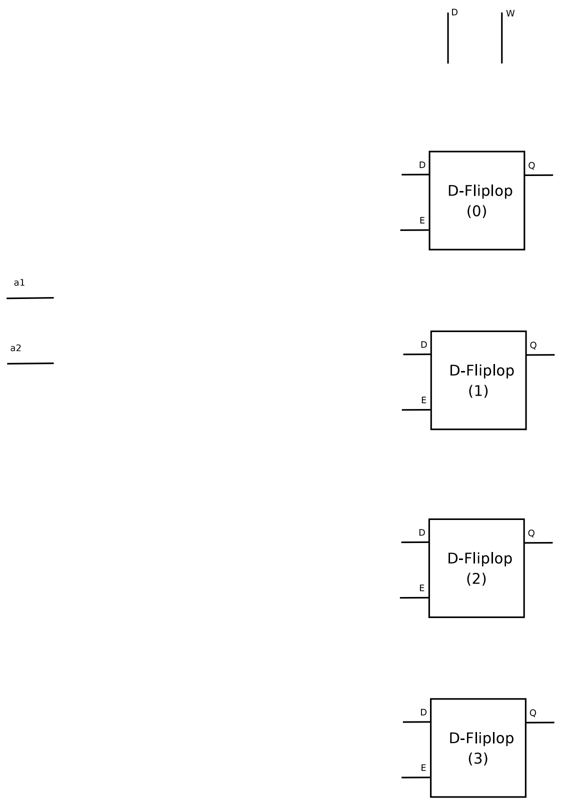
Solution:



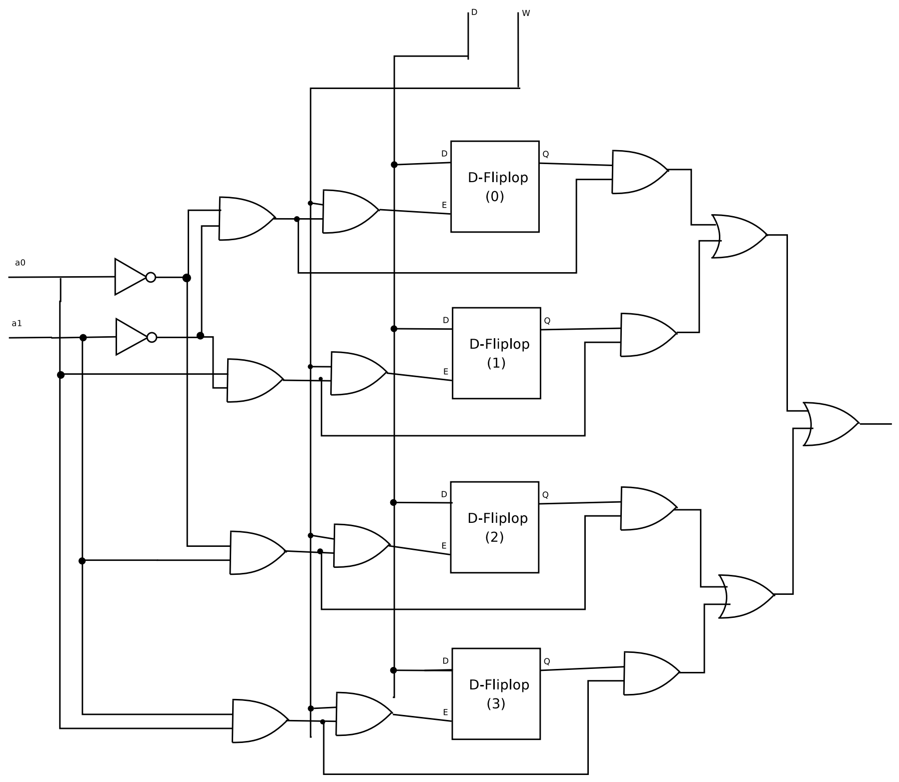
Problem 2.46 (Reading from and writing to memory)

Suppose you have a 2-bit addressed memory of 4 bits managed by 4 D-Flipflops aligned as shown in the figure. The input of the circuit consists of a total of 4 bits. 2 of the bits (a_0 and a_1) provide a 2-bit address. In addition there is a data bit D and a write bit W.

Design a circuit which output should be the data memorized in the D-Flipflop addressed by $\langle a_1, a_0\rangle$. In addition if the write bit W is 1, your circuit should write the data from the data bit D to the same D-Flipflop addressed by $\langle a_1, a_0\rangle$.



Solution:

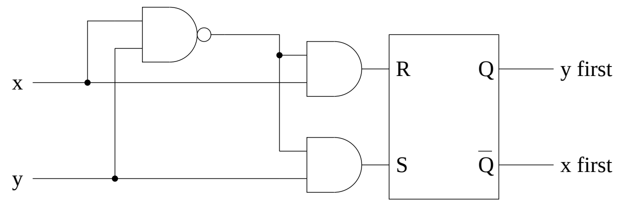

D
W
a0
a1
D
Q
D-Fliplop
(0)
E
D
Q
D-Fliplop
(1)
E
D
Q
D-Fliplop
(2)
E
D
Q
D-Fliplop
(3)
E

Problem 2.47 (Event Detection with RS Flipflops)
Using RS flipflops, you can detect events.

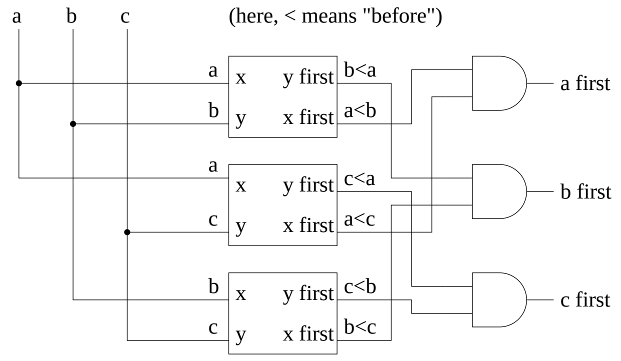
1. Design a sequential logic circuit (draw a graph) with two inputs and two outputs that detects, which out of *two* events occurred first. Use the RS flipflop and elementary gates (AND, OR, NOT, . . .). Assume that, initially, all inputs are 0 and the RS flipflop(s) are holding a 0. If input I_i, where $i \in \{1, 2\}$, changes its value to 1, output O_i should change its value to 1, and all other outputs should yield 0. The outputs must not change any more when the second input changes to 1.
2. Combine several (how many?) of the circuits from step 1 to a similar event detector for three events.

Note: You need not handle the case of two inputs simultaneously changing to 1.

Solution: Circuit that checks which out of two events (x or y) occurred first:



Three of those combined to a circuit that checks which out of three events (a, b, or c) occurred first:



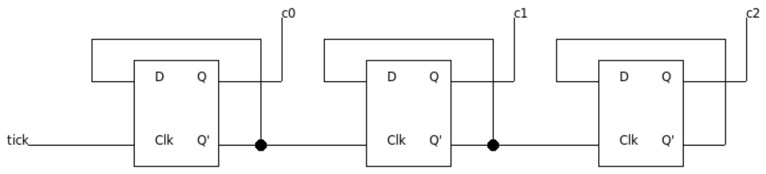
Problem 2.48 (Binary counters)

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: For those of you who are curious here is how an edge-trigerred D-flipflop is built from NAND gates: `http://en.wikipedia.org/wiki/File:Edge_triggered_D_flip-flop.png`. If you're trying to understand this it will help to note that a real physical gate has a certain delay. When the input changes it takes some time (nanoseconds) for the output to react.

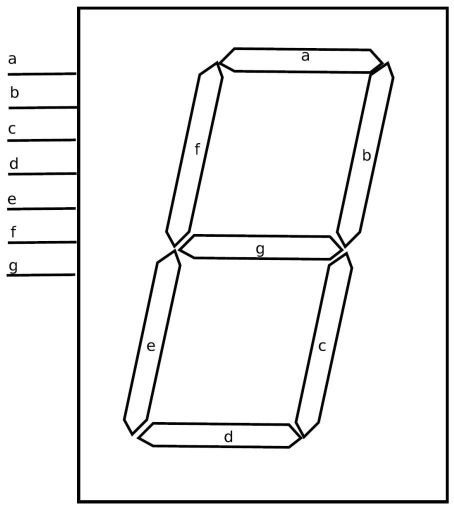
Solution:



Problem 2.49 (Displaying a two-bit number)
Your task for this problem is to create a 2-bit synchronous counter and display the output in a decimal form with the help of 8 light emitting diodes.

You need to assemble this circuit only with the help of the following items:

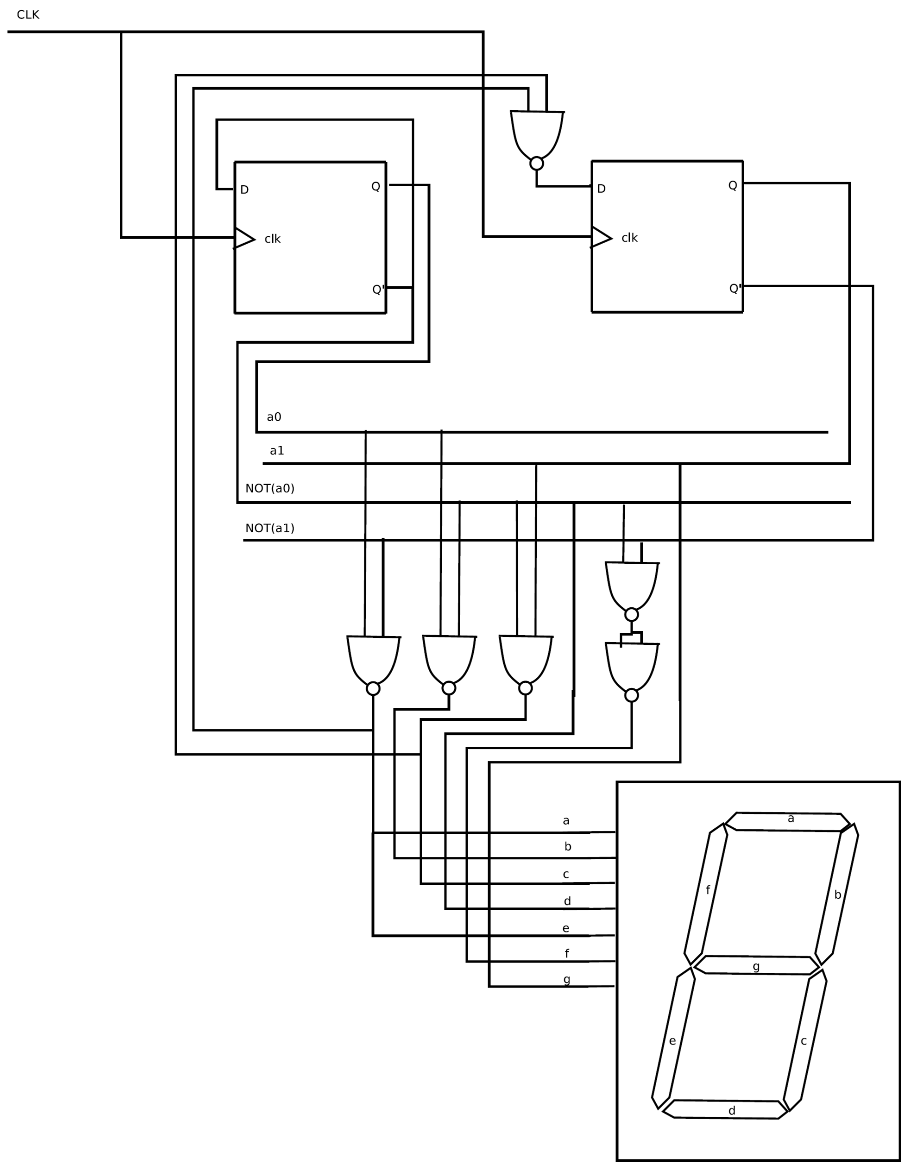
- 2 positive edge triggered D-flipflops
- 6 NAND gates
- 1 digit display circuit with 8 inputs ($a - g$) corresponding to 8 diodes arranged in the figure below
- 1 signal generator that provides you with a clock signal that you should use to trigger the D-flipflops
- set of wires



Note:

- Basically your task is to create a 2-bit counter and decode the 2-bit output of the counter into 8-bits so that the display shows proper numbers from 0 to 3.
- Positive edge triggered D-Flipflop is just like a normal D-flipflop with the exception that it writes the data when the enable signal (clock) transits from 0 to 1, and in all other cases (constant 0, constant 1, transition $1 \to 0$) nothing happens.
- You cannot use constant signals.
- For all of the inputs to the 1-digit display logical true (1) means ON and logical false (0) means OFF for the corresponding diode.
- You dont need to worry about the power supplies of the diodes, ICs and the flipflops.

Solution:

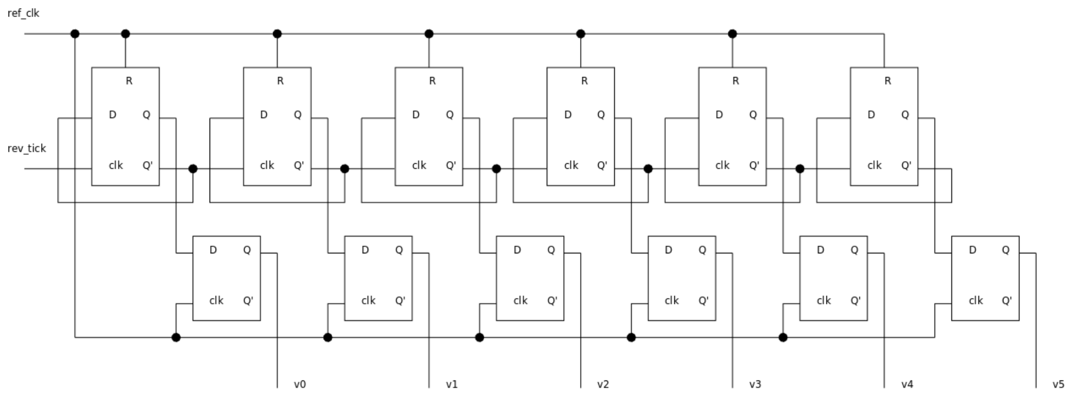

CLK
D
Q
clk
Q'
D
Q
clk
Q'
a0
a1
NOT(a0)
NOT(a1)
a
b
c
d
e
f
g
a
f
b
g
e
c
d

Problem 2.50 (Making a speedometer)
You are working for a car manufacturer and are given the task to make a digital speedometer for a future model. The electrical engineers tell you that they can provide you with two inputs: *rev_tick* very briefly goes from 0 to 1 and then back to 0, whenever the wheels of the car complete one revolution and *ref_clk* that every second very briefly goes form 0 to 1 and then back to 0. You know that the wheels of the car have a circumference of 1 meter. For the initial design you need to provide an electronic circuit that measures the speed in meters per second. You have to provide a number of outputs $a_0, \ldots, a_n$ that represent the current speed. You also know that the car has a maximum speed of 220 km/h.

Imagine that you wanted to display the speed in km/h. What is the maximum resolution your speedometer could achieve? What improvements to the car design can you propose to make this better?

For this problem you should use the edge-trigerred flip-flop together with an extended version that has one additional input R. Whenever R is one, the internal state of the flip-flop is reset to 0 ($Q = 0$) regardless of the state of the D and *clk* inputs. Reseting the internal state when R becomes 1 also happens after a short delay.

Solution:



With the current design we can only achieve a resolution of 3.6 km/h (1m/s). To improve this we need to have a greater number of impulses per 1 revolution of the wheels. For example 100 ticks per revolution will be enough to provide a resolution of 0.1 km/h.

2.4 Machines

2.4.1 How to build a Computer (in Principle)

20pt

Problem 2.51 (Hyperpower)
Write an assembler program that reads an integer $n \geq 1$ stored in $P(0)$, and writes n^n in $P(1)$.

Solution:

P	instruction
0	`LOADI` 1
1	`STORE` 1
2	`LOAD` 0
3	`STORE` 2
4	`JUMP`$_=$ 15
5	`LOADI` 0
6	`STORE` 3
7	`LOAD` 3
8	`ADD` 1
9	`STORE` 3
10	`LOAD` 2
11	`SUBI` 1
12	`STORE` 2
13	`JUMP`$_{\neq}$ -6
14	`LOAD` 3
15	`STORE` 1
16	`LOAD` 0
17	`SUBI` 1
18	`JUMP` -14
19	`STOP` 0

Problem 2.52 (Multiplication) 10pt

Write an assembler program (for the assembler language we defined in class) that multiplies the values of data cells 1 and 2 and stores the result in data cell 0.

Problem 2.53 (Poking zeros) 25pt

Given are $n \geq 1$ (n is stored in $P(0)$) integers stored in $P(10) \ldots P(9+n)$, such that no two zeros are next to each other and $P(10) \neq 0 \neq P(9+n)$. Write an assembler program that overwrites all zeros in that array with the sum of the numbers in the neighboring cells of its position.

Solution:

P	instruction
0	`LOAD` 0
1	`SUBI` 1
2	`STORE` 0
3	`LOADI` 11
4	`MOVE` ACC $IN1$
5	`LOAD` 0
6	`SUBI` 1
7	`STORE` 0
8	`JUMP`$_{\leq}$ 12
9	`LOADIN` 1 0
10	`JUMP`$_{\neq}$ 6
11	`LOADIN` 1 -1
12	`STORE` 1
13	`LOADIN` 1 1
14	`ADD` 1
15	`STOREIN` 1 0
16	`MOVE` $IN1$ ACC
17	`ADDI` 1
18	`MOVE` ACC $IN1$
19	`JUMP` -14
20	`STOP` 0

Problem 2.54 (Simulating a Register Machine) 70pt

Write an SML function `regma` (register machine) that simulates the simple register machine we discussed in class. To represent the program and data store, you should use SML vectors as described in `http://www.standardml.org/Basis/vector.html`. In a nutshell, `Vector.sub(arr,i)` returns the i^{th} element of the vector `arr` and `Vector.update(arr,i,x)` returns the vector `arr`, except that the i^{th} element is replaced by `x`. Finally (useful for testing) `Vector.fromList` makes a vector from a list.

So the the data store should be of type `int vector` and the program store is of type `(instruction * int) vector`, where `instruction` is defined by the following type

```
datatype instruction =
         load | store | add | sub | loadi | addi | subi |
         loadin1 | loadin2 | storein1 | storein2 |
         moveaccin1 | moveaccin2 | movein1acc | movein2acc | movein1in2 | movein2in1 |
         jump | jumpeq | jumpne | jumpless | jumpleq | jumpgeq | jumpmore |
         nop | stop
```

`regma` should take as input a data store `data` and a program store `prog`, and `regma(prog,data)` should return the value of the accumulator register, when the program encounters a `stop` instruction.

Solution: We first write an auxiliary function that takes care of the current instruction by a large case statement. Let us start out with the load/store instructions:

```
    \scriptsize
fun arg(n,p) = let val (_,a) = Vector.sub(n,p) in a end
fun doinst ((load,i),prog,data,acc,pc,in1,in2) =
       doinst(Vector.sub(pc+1,prog),prog,data,arg(pc,prog),pc+1,in1,in2)
  | doinst((store,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,Vector.update(data,acc,i),pc+1,in1,in2)
  | doinst ((loadi,i),prog,data,acc,pc,in1,in2) =
       doinst(Vector.sub(pc+1,prog),prog,data,i,pc+1,in1,in2)
  | doinst((loadin1,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,Vector.sub(data,in1+i),pc+1,in1,in2)
  | doinst((loadin2,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,Vector.sub(data,in2+i),pc+1,in1,in2)
  | doinst((storein1,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,Vector.update(data,acc,in1+i),pc+1,in1,in2)
  | doinst((storein2,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,Vector.update(data,acc,in2+i),pc+1,in1,in2)
```

Then come the cases for the computation instructions, where we just make use of the SML computation facilities.

```
    \scriptsize
  | doinst((add,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc + Vector.sub(data,i),pc+1,in1,in2)
  | doinst((Vector.sub,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc - Vector.sub(data,i),pc+1,in1,in2)
  | doinst((add,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc + i,pc+1,in1,in2)
  | doinst((Vector.sub,i),prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc - i,pc+1,in1,in2)
```

The register move instructions are rather boring:

```
    \scriptsize
  | doinst(moveaccin1,prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,acc,in2)
  | doinst(moveaccin2,prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,in1,acc)
  | doinst(movein1acc,prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,in1,pc+1,in1,in2)
  | doinst(movein2acc,prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,in2,pc+1,in1,in2)
  | doinst(movein1in2,prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,in1,in1)
  | doinst(movein2in1,prog,data,acc,pc,in1,in2) =
      doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,in2,in2)
```

The jump instructions can be mapped to conditional expressions in SML using the SML comparisons

```
    \scriptsize
  | doinst((jump,i),prog,data,acc,pc,in1,in2) =
     doinst(Vector.sub(pc+i,prog),prog,data,acc,pc+i,in1,in2)
  | doinst((jumpeq,i),prog,data,acc,pc,in1,in2) =
       if (acc = 0)
       then doinst(Vector.sub(pc+i,prog),prog,data,acc,pc+i,in1,in2)
       else doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,in1,in2)
  ...
  | doinst((jumpmore,i),prog,data,acc,pc,in1,in2) =
       if (acc > 0)
       then doinst(Vector.sub(pc+i,prog),prog,data,acc,pc+i,in1,in2)
       else doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,in1,in2)
```

For the skip instruction we do nothing, except increment the program counter and supply the next instruction:

```
    \scriptsize
      | doinst(nop,prog,data,acc,pc,in1,in2) =
     doinst(Vector.sub(pc+1,prog),prog,data,acc,pc+1,in1,in2)
```

Finally, the stop instruction just returns the value of the accumulator.

```
    \scriptsize
      | doinst(stop,prog,data,acc,pc,in1,in2) = acc
```

With this giant case distinction, the function `regma` is very simple, we just have to use `doinst` with suitable initial values.

```
    \scriptsize
fun regma (prog,data) = doinst(Vector.sub(0,prog),prog,data,0,1,0,0)
```

Problem 2.55 (sorting-by-selection) 20pt 20min

Let $n \geq 1$ be stored in $P(0)$ and n numbers stored in $P(2) \dots P(n+1)$. Write an assembler program that performs a sorting by selection and outputs the result in $P(n+2) \dots P(2n+1)$. Write comments to each line of your code (like in the example codes from the slides). Uncommented code will not be considered.

Solution:

P	instruction	comment
0	`LOAD` 0	$ACC\colon = P(0) = n$
1	`ADD` 0	$ACC\colon = ACC + n$
2	`MOVE` ACC $IN2$	The recursion index. Initializing $IN2 = 2n$ (will come in handy when printing the results)
3	`LOAD` 0	$ACC\colon = n$
4	`MOVE` ACC $IN1$	$IN1\colon = n$ Initialising the sequence index. Outer loop starts.
5	`LOADIN` 1 1	
6	`STORE` 1	Initialize $MAX = P(1) = P(n+1)$
7	`MOVE` $IN1$ ACC	$ACC\colon = IN1 = n$ Starting inner loop
8	`SUBI` 1	
9	`MOVE` ACC $IN1$	$IN1 - -$
10	`JUMP`$_=$ 7	if $IN1$ becomes 0 we have a MAX
11	`LOADIN` 1 1	$ACC\colon = P(IN1 + 1)$
12	`SUB` 1	$ACC\colon = ACC - MAX$
13	`JUMP`$_<$ 2	if $P(IN1 + 1) < MAX$ jump
14	`LOADIN` 1 1	else $MAX = P(IN1 + 1)$
15	`STORE` 1	
16	`JUMP` -9	End inner loop (for one max)
17	`LOAD` 1	$ACC = MAX$
18	`STOREIN` 2 1	$P(IN2 + 1)\colon = ACC = MAX$
19	`LOAD` 0	now we have to make the max we chose to be equal to 0, so we wouldn't find it again
20	`MOVE` ACC $IN1$	we test if the current number equals $P(1)$
21	`LOADIN` 1 1	
22	`SUB` 1	
23	`JUMP`$_=$ 5	if we find the number we make it 0
24	`MOVE` $IN1$ ACC	
25	`SUBI` 1	
26	`MOVE` ACC $IN1$	
27	`JUMP` -6	
28	`LOADI` 0	
29	`STOREIN` 1 1	
30	`MOVE` $IN2$ ACC	
31	`SUBI` 1	
32	`MOVE` ACC $IN2$	$IN2 - -$
33	`SUB` 0	$IN2\colon = IN2 - n$
34	`JUMP`$_>$ -20	if $IN2 - n > 0$ go back again. End big loop.
35	`STOP` 0	

Problem 2.56 (Binary to decimal) 20pt 20min

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

Solution:

label	instruction	comment
	`LOAD` 0	
	`MOVE` ACC $IN1$	The recursion index. Initializing $IN1 := n$
	`LOADI` 0	
	`STORE` 1	Initialize $P(1) := 0$
$\langle loop \rangle$	`MOVE` $IN1$ ACC	
	`JUMP`$_=$ $\langle end \rangle$	if $IN1$ becomes 0 we are done.
	`LOAD` 1	
	`ADD` 1	
	`STORE` 1	$P(1) := 2 \cdot P(1)$
	`LOADIN` 1 1	
	`ADD` 1	
	`STORE` 1	$P(1) := P(1) + P(IN1+1)$
	`MOVE` $IN1$ ACC	
	`SUBI` 1	
	`MOVE` ACC $IN1$	$IN1--$
	`JUMP` $\langle loop \rangle$	go to next iteration.
$\langle end \rangle$	`STOP` 0	

2.4.2 A Stack-based Virtual Machine

Problem 2.57 (Reasons for Virtual Machines)
Thinking back to the lectures about $\mathcal{L}(\texttt{VM})$ and `SW`, sum up the benefits of compiling programs in high-level languages to the language of a virtual machine instead of directly compiling them to an assembler language `ASM`.

Problem 2.58 (Binary Conversion in $\mathcal{L}$(VM)) 12pt 15min

Write a $\mathcal{L}$(VM) program that converts a binary natural number into a decimal natural number. Suppose that n, the number of digits, is stored in `stack[2]` and n numbers 0 or 1 above it follow, where the top of stack is the least significant bit. `stack[0]` and `stack[1]` are available for your use. Your program should leave only the converted number on the stack (in `stack[0]`). You are allowed to use labels for (conditional) jumps.

For instance an initial stack $\begin{array}{|c|}\hline 1\\\hline 0\\\hline 1\\\hline 3\\\hline ?\\\hline ?\\\hline\end{array}$ should give the result stack $\boxed{5}$.

Solution: `con` (0)

```
poke (0) ; init. result to 0
con (1)
poke (1) ; init. 2^i to 1
peek (1)
mul ; multiply with 2^i
peek (0)
add ; add to result
poke (0)
peek (1) ; update multiplier
con (2)
mul
poke (1) con (1) 1 ; update digit counter
peek (2)
sub
poke (2)
peek (2) ; if counter = 0 go out
cjp 4 jp − 26 ; else again
poke (1) 1 ; clean stack and stop
add
halt
```

Problem 2.59 (Fibonacci Numbers) 12pt

Assume the data stack initialized with `con` n for some natural number n. Write a $\mathcal{L}$(`VM`) program that computes the n^{th} Fibonacci number and returns it on the top of the stack.

Hint: Remember that the n^{th} Fibonacci number is given by the following recursive equations:

$$fib(n+1) = f(n) + fib(n-1) \qquad fib(0) = 0 \qquad fib(1) = 1$$

Solution:

```
con n The requested fibonacci number
con 0 con 1 The 0th and the 1st fibonacci number
con 0 peek 0 leq cjp 5 If $\RMdatastore{0}$ <= 0
peek 1 halt return the current fibonacci number
peek 2 else save the next fibonacci number ...
peek 1 peek 2 add poke 2 ... compute the number after next and save at $\RMdatastore{2}$
poke 1 ... make the next number current ...
con 1 peek 0 sub poke 0 ... decrease n by 1 ...
jp -28 ... and jump back the the beginning
```

2.4.3 A Simple Imperative Language

20pt

Problem 2.60 (Convert Highlevel Code to VM Code)
Given is an array `A[0..10]` and the following piece of imperative code:

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

Suppose the array is loaded on stack (top value being `A[10]`). Convert the code into VM code.

Problem 2.61 (Static procedure for logarithm)
Write down a static procedure in $\mathcal{L}(\texttt{VM})$ that computes $f(x) = \lfloor \log_2(x) \rfloor$. This procedure should not be recursive. Use the new `lpeek` and `lpoke` instructions from the previous exercise. Is there something you do at the end of your procedure that is not part of your algorithm. If yes, then describe a more elegant way of doing that by modifying the behavior of an existing `VM` instruction.

Hint: Remember that at the end of a static procedure call exactly one value - the result - should be left on the stack.

Solution:

`proc` 1 34	$f(x)$
`con` 0 `con` 2	local variables n, y
`arg` 1 `lpeek` 1 `leq` `cjp` 16	if $y \leq x$
`con` 1 `lpeek` 0 `add` `lpoke` 0	$n := n + 1$
`con` 2 `lpeek` 1 `mul` `lpoke` 1	$y := y * 2$
`con` 0 `lpoke` 0 `add`	else: eliminate all excess elements on the stack
`return`	return n

At the end we have two variables left on the stack n and y. We need to eliminate y. Therefore we make it 0 and then add this to n. In this way only the final result is left on the stack.

A better way of implementing this for example is to make the return instruction take one argument which represents the number of excess variables on the stack. The implementation of this instruction can then take care of adjusting the stack pointer to the right position.

Problem 2.62 (While Loop in $\mathcal{L}$(VM)) 6pt 10min

Write a program in the Simple While language that takes two numbers A and B, given at the memory addresses 1 and 2, and returns $(A+B)^{42}$. Show how the compiled version of it looks like in the Virtual Machine Language $\mathcal{L}$(VM) (concrete, not abstract syntax).

Solution:

`var n := 1; var a := A; var b := B;`	con 1 peek 1 peek 2
`var c := a+b;`	add con 1
`while n >= 42 do (`	peek 0 con 43 leq cjp *halt*
`  p := c * p;`	peek 1 peek 2 mul poke 2
`  n := n + 1;`	peek 0 con 1 add poke 0
`)`	jp back
`return p;`	halt

Problem 2.63 (Simple While program on Fibonacci)

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

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

Hint: Remember that the n^{th} Fibonacci number is given by the following recursive equations:

$$fib(n+1) = f(n) + fib(n-1) \qquad fib(0) = 0 \qquad fib(1) = 1$$

Solution:

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

VM code:

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

2.4.4 Compiling Basic Functional Programs

Problem 2.64 (Cross identifiers?)
Now suppose you want to compile a μML program containing a few function declarations such that they use the local identifiers from the functions defined above. For example,

```
([("F1", ["n","a"], Sub(Id "n",Id "a")),
  ("F2", ["m","x"], Mul(Add(Id "m",Id "x"),Id "n"))],
 App("F2", [Con 1, Con 2]) );
```

Will such a program compile? If yes, will it execute correctly? Explain your answer.

Hint: You may want to track down the compilation process on a given example.

Solution: Since the compiler doesn't check for such cross-references, it will get confused and ultimately stop compilation raising an uncaught error.

Problem 2.65 (Duplicate identifiers?)
Suppose you want to compile a μML program containing a few function declarations such that some of them contain the same identifier names such as

```
([("F1", ["n","a"], Sub(Id "n",Id "a")),
 ("F2", ["m","n","a"], Mul(Add(Id "m",Id "a"),Id "n"))],
 App("F2", [Con 1, Con 2, Con 3]) );
```

Will such a program compile? If yes, will it execute correctly? Explain your answer.

Hint: You may want to track down the compilation process on a given example.

Solution: Such programs will compile successfully anyway, since the compilation of every list of identifiers is followed by the function that uses it. Although the identifiers from the list from the function that follows will be poured in the same environment, the identical names will be overwritten and the usage of the new identifiers will cause no problems.

Problem 2.66 (Prime numbers)
Write a program in μML that takes an integer $n > 1$ and returns 1 if the number is prime and 0 otherwise. Your program should be a pair of a well defined list of function declarations, and a single `App` call of the main function. Obviously, that function will call the helping function(s) in its body and helping functions may call themselves. Can you solve the problem using only two helping functions?

Solution: A long, standard solution

```
(
[

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

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

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

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

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

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

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

],
App("IsPrime", [Con 119]) );
```

Another ingenious test for prime numbers taken from ”Goedel, Escher, Bach”

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

("DF", ["x", "n"],
 If(Leq(Id"n", Con 2), App("DND", [Con 2, Id"x"]),
  If(App("DND", [Id"n", Id"x"]),
     If(App("DF", [Id"x", Sub(Id"n", Con 1)]), Con 1, Con 0),
     Con 0)
  )
),

("Prime", ["x"],
  App("DF", [Id"x", Sub(Id"x", Con 1)])
)
],

App("Prime", [Con 9])
);
```

2.4.5 A theoretical View on Computation

10pt
10min

Problem 2.67: Explain the concept of a Turing machine, what is it used for? What is a universal Turing machine?

Problem 2.68 (Turing Machine) 12pt

Given the alphabet $\{0, 1\}$ and a initial tape that starts with 0,1,0.

1. Define a transition table that converts the three entries of this tape to 1,0,1 and terminates afterwards independently of the tape's tail.
2. Give an example initial tape where your transition table wouldn't terminate or argue why such an initial tape can't exist.

Hint: The Turing machine terminates when there is no action in the transition table applicable.

Problem 2.69 (Boolean And) 5pt 8mm

Suppose a tape with only two cells arbitrarily filled with 0 or 1 and the head of the Turing machine over the left cell. Define a transition table such that the machine always terminates with a final state where the left cell has value 1 if and only if both cells contained 1 in the initial state; i.e. the machine should evaluate the a boolean "and".

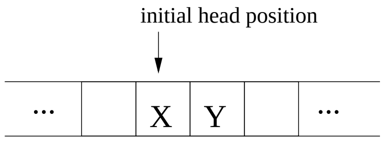
Hint: Admissible moves are *left*, *right*, and *stop* with the obvious meaning.

Solution: Missing entries in the transition table can be filled arbitrarily; i.e. they are irrelevant for the solution of the problem.

Old	Read	Write	Move	New
s_0	0		stop	
s_0	1	0	right	s_1
s_1	0		stop	
s_1	1		left	s_2
s_2	1		stop	

Problem 2.70 (Boolean Equivalence) 11pt 20min

Consider a tape arbitrarily filled with ones and zeros and the head initially positioned over some cell "X" as depicted below



Define a transition table for an always terminating Turing machine TM that computes the boolean equivalence of "X" and "Y": Upon halting, your TM should return the value 1 in cell "X" if the values of the cells "X" and "Y" were initially equal and otherwise 0.

Try to use as few states as possible. The number of points you can obtain for this exercise is $\max(0, 14 - x)$, where x is the number of states of your working TM.

Hint: You only need to consider the two cells "X" and "Y". It does not matter where the head stays when the TM terminates.

Note:

1. Admissible moves are *left*, *right*, and *none* with the obvious meaning.
2. You are free to overwrite the initial value of "Y" and to introduce additional symbols in the alphabet, if you need it for your solution.

Solution: There are lots of possible solutions.

The following four-state solution (including the final state) by Dmakreshanski Pesikan does not use any additional alphabet symbols:

Old	Read	Write	New	Move
s_1	0	0	s_2	right
s_1	1	1	s_2	right
s_2	0	0	s_3	left
s_3	1	0	s_4	left
s_3	0	1	s_4	none

Tanmay Pradhan presented a solution that uses additional symbols and only needs *two states.* This is supposed to be optimal.

Old	Read	Write	New	Move
s_1	0	W	s_2	right
s_1	1	Y	s_2	right
s_2	0	L	s_1	left
s_2	1	L	s_2	left
s_1	W	1	s_1	right
s_1	Y	0	s_1	right
s_2	W	0	s_1	right
s_2	Y	1	s_1	right

If we require that the head must stop on "X" – but we don't , as Darko pointed out! – , it gets more complicated. The following solution by Christoph Lange is quite straight-forward, but not optimal:

Old	Read	Write	New	Move
a	0	0	b	right
a	1	1	b	right
b	0	0	c_0	left
b	1	1	c_1	left
c_0	0	1	$\perp$	none
c_0	1	0	$\perp$	none
c_1	0	0	$\perp$	none
c_1	1	1	$\perp$	none

Andrei Aiordachioaie supposed this four-state solution:

Old	Read	Write	New	Move
s_0	1	X	right	s_x
s_0	0	Y	right	s_y
s_x	1	1	left	s_x
s_x	X	1	none	$\perp$
s_x	0	(anything)	left	s_y
s_x	Y	0	none	$\perp$
s_y	0	0	L	s_y
s_y	Y	1	none	$\perp$
s_y	1	(anything)	left	s_x
s_y	X	0	none	$\perp$

Problem 2.71 (Halting Reductions)
The fact that a TM cannot decide if another TM halts on a given input is not the only limit of computation. There are a lot of other things TM's cannot do, and the halting problem can be used to prove this. This process is called "reduction to the halting problem": for proving that a TM cannot decide a certain a property P, assume that it could and then use it to construct another TM that can decide the halting problem (i.e. to decide if some TM halts on some given input).

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

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

Hint: Here is an example of how to solve such a task. All you need to do is to figure out how to adapt this to the points above.

- Prove or refute that no TM can decide in general if another TM halts on the empty input.
- Assume we have a machine M that can decide if another TM halts on the empty input. We want to decide if a given TM N halts on input x. We can construct a machine K that started on the empty input, writes x on the tape and then simulates $N(x)$. If $M(K)$ (M run on a coded version of K as input) outputs yes, then it means that K halted on the empty input, thus N halted on x, no means the opposite. Thus, we can decide the halting problem, which is false.

Solution:

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

Problem 2.72 (Number of Steps of a Turing Machine)
Let $s_{\max}(n)$ be the maximum number of steps that an n-state Turing machine with the alphabet $\{0,1\}$ can take on an empty tape, halting in the end. Is the function $s_{\max}$ computable? Give a proof or a refutation.

Hint: If we had an implementation of $s_{\max}$, how could we implement the `will_halt` function from the lecture using $s_{\max}$?

Note: From the lecture, we know that it is impossible to implement a function `will_halt(program, input)`. Assume the following corollary, known as the "halting problem on the empty tape", as given: It is even impossible to write a Turing machine (or an equivalent function `will_halt_empty(program)`, resp.) that tells whether an arbitrary Turing machine halts on an *empty* tape.

Solution:
Proof: by contradiction

P.1 $s_{\max}(n)$ is the maximum number of steps a halting n-state Turing machine can take on an empty tape.

P.2 Any n-state machine that runs for more than $s_{\max}(n)$ steps must be non-halting.

P.3 Using $s_{\max}$, one could now implement `will_halt_empty(TM)` as follows:

- Let n be the number of states of `TM`.
- Compute $m := s_{\max}(n)$.
- Simulate at most m steps of `TM`.
- If `TM` has not halted so far, return "yes"; otherwise, return "no".

P.4 This contradicts the non-computability of `will_halt_empty`. □

Problem 2.73 (TM and languages)

Design a Turing Machine which accepts the language $\{101100...1^{n-1}0^{n-1}1^n0^n \mid n > 0\}$ (halts with "yes" if such input is given and halts with "no" otherwise). First describe in plain English the core idea of how your algorithm works. Think of possible wrong inputs, and show how your TM handles them.

Note:

- The point of this exercise is to help you think of how to approach and solve a problem. Imagine you are given 0 points for a TM which only partially works (some wrong inputs can pass as accepted or the other way around).
- For exercises about TM construction, please format the transition table according to the TM simulator at `http://ironphoenix.org/tril/tm/` (here you will also find some example programs). This way you will be able to check your "code" and your TAs will have an easier time grading.

Solution:

- A good starting point is to think of how to handle something of the form $X^k1^{k+1}0^{k+1}$, where X is a special symbol we will use to mark processed 0s. For processed 1s we will use Y.
- This is a relatively easier problem, and once we solve this, we can just repeat the process n times over our input.
- **Idea:** for each X, mark the first '1' (going from left to right) with a Y, and mark the last '0' with an X. You also have to mark the initial 'X's to keep their track (Well mark them with a 'Z').
- **Note:** if you would start marking the 0s from the left too, you would get tricked by inputs like XXX10101010.
- Keep doing this step until there are no more 'X's in front. The word will look something like $ZZZYYY10XXX$ (for a correct input). Now you just have to check that there is exactly one 1 followed by exactly one 0. You dont have to check for the number of Ys and Xs since you only wrote one for each Z.
- *Possible wrong inputs:*
  - $XXX111110000 \rightarrow ZZZYYY110XXX$ (will fail at the last check)
  - $XXX111100000 \rightarrow ZZZYYY100XXX$ (will fail at the last check)
  - $XXX10101010 \rightarrow ZXXYX10101010$ (will fail when processing the second X)
  - $XXX110000 \rightarrow ZZXYY00XX$ (will fail when processing the third Z)
- There are also a few things to be taken care of in the beginning and in the end:
  - In the beginning we dont have the preceding 'X's, but we only want a '1' followed by a '0' (which happens to be the same thing as our final check → have this as a starting state).
  - In the end, on a correct input we will have 'X's followed by blanks. So let our checking procedure halt with "yes" if it immediately encounters a blank.

Solution: Here is the complete code:

```
1,_ H,1,>
1,1 2,Y,>
1,0 H,0,>
1,X H,0,>
1,Y H,0,>

2,0 5,Z,>
2,1 H,0,>
2,X H,0,>
2,Y H,0,>
2,_ H,0,>

3,_ H,1,>
3,X 4,X,>
3,0 H,0,>
3,1 H,0,>
3,Y H,0,>
```

```
4,X 5,Z,>
4,Y 10,Y,>
4,1 H,0,>
4,0, H,0,>
4,_ H,0,>

5,X 5,X,>
5,Y 5,Y,>
5,1 6,Y,>
5,_ H,1,>
5,0 H,0,>

6,1 6,1,>
6,0 7,0,>
6,X H,0,>
6,Y H,0,>
6,_ H,0,>

7,0 7,0,>
7,1 8,1,<
7,X 8,X,<
7,Y H,0,>
7,_ 8,_,<

8,0 9,X,<
8,Y H,0,>
8,X H,0,>
8,1 H,0,>
8,_ H,0,>
```

Solution:

```
9,0 9,0,<
9,1 9,1,<
9,Y 9,Y,<
9,X 9,X,<
9,Z 4,Z,>

10,Y 10,Y,>
10,1 2,Y,>
10,0 H,0,>
10,X H,0,>
10,_ H,0,>
```

Problem 2.74 (TM and TCN numbers)
Given a tape with an n-bit binary number written after symbol + or − (denoting if the number is positive or negative), design a Turing Machine which will convert it to a TCN. Initially, the head is over the sign symbol. There is no restriction where would the head be after halting. If the number of states exceeds 4, you will lose 2 points per extra state. **Uncommented code will not be graded.**

For example we would have

```
Input: -101
Output: 1011
```

Solution:

Old	Read	Wr.	Mv.	New
q_1	$+$	0	_	H
q_1	-	1	R	q_{flip}
q_{flip}	0	1	R	q_{flip}
q_{flip}	1	0	R	q_{flip}
q_{flip}	_	_	L	q_{addone}
q_{addone}	0	1	_	H
q_{addone}	1	1	L	q_{addone}
q_{addone}	_	1	_	H

Problem 2.75 (Turing Machine Simulating a Half Adder)
Given the alphabet $\{0,1\}$ and a finite set of states of your choice. Define upon these sets a transition table that behaves like a half adder, i.e. it reads two bits from the tape and writes a sum and carry bit on the tape again (at any arbitrary but fixed position).

Solution: According to the table for the Half bit Adder

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

by considering the head on A and the state of the system $s1$ at the beginning the head will be moved to C and the system will be in state $s9$ when halting. The complete transition table is the following:

oldstate	*read*	*write*	*move*	*newstate*
$s1$	0	0	R	$s2$
$s1$	1	1	R	$s3$
$s2$	0	0	R	$s4$
$s2$	1	1	R	$s5$
$s3$	0	0	R	$s5$
$s3$	1	1	R	$s6$
$s4$	0	0	R	$s7$
$s4$	1	0	R	$s7$
$s7$	0	0	L	$s9$
$s7$	1	0	L	$s9$
$s5$	0	0	R	$s8$
$s5$	1	0	R	$s8$
$s8$	0	1	L	$s9$
$s8$	1	1	L	$s9$
$s6$	0	1	R	$s7$
$s6$	1	1	R	$s7$

2.5 The Information and Software Architecture of the Internet and WWW

2.5.1 Overview

nothing here yet

2.5.2 Internet Basics

nothing here yet

2.5.3 Basics Concepts of the World Wide Web

Problem 2.76 (Quiz for the TAs)

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

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

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

You can provide a fictive action attribute.

Hint: HTML is useful and easy to learn. Start by finding a nice tutorial online.

Problem 2.77 (HTML basics)
Answer the following questions about HTML:

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

Solution:

1. Hyper Text Markup Language
2. The World Wide Web Consortium
3. `<h1>`
4. `<br />`
5. `<body style="background-color:yellow">`
6. `<b>`
7. `<i>`
8. `<a href="http://www.link.com">WordToBeLinked</a>`
9. `<a href="mailto:xxx@yyy">`

10. `<a href="url"target="_blank">`
11. `<table><tr><td>`
12. `<td align="left">`
13. `<ol>`
14. `<ul>`
15. `<input type="checkbox"/>`
16. `<input type="text"/>`
17. `<select>`
18. `<textarea>`
19. `<img src="image.gif"/>`
20. `<body background="background.gif">`

Problem 2.78 (For Future Generations)

As one of the last assignments, we would like you to look a bit into the future. Imagine yourselves one year from now. Some of you will definitely be TAs at that time, so it's time to show your creativity and teaching skills. Your task is to basically create an HTML form representing the examination you would give to the freshmen in 2012. It can be any midterm or final for GenCS I or II. There are only a few specifications you must look out for. The rest is fully up to you.

The web form must:

1. Include multiple choice and 'fill in the blanks' questions, enough for an actual exam time of 75 or 120 minutes.
2. Include all of the following: button, radio button, check box, drop down box, text input.
3. The exam must contain figures and sections of code from any of the studied programming languages that you ask questions on.
4. Link your exam to some useful pages. Make it like an 'open book' exam and offer some actual existing resources.
5. The overall style should be professional. Put a bit of effort into appearance and aesthetics.
6. In the end, the scoring system should work. Nothing too fancy, but it should be an operational exam from start to finish.

Hint: HTML is useful and easy to learn. Start by finding a nice tutorial online. You might wish to consider JavaScript for your scoring mechanism. Also, CSS is recommended for brushing up your design!

Problem 2.79 (Web browsers)

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

Solution:

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

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

2.5.4 Web Applications

Nothing here yet

2.5.5 Introduction to Web Search

Problem 2.80 (SML Web Crawler)
A web crawler is a program that will store a copy (mirror) of a web site. Generally, crawlers access a given web page and, after retrieving the HTML source, they extract the links and also download those pages (or images or scripts). This will provide the user the possibility to access these pages even when they are not connected to the internet or to perform different measurements on the pages.

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

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

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

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

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

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

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

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

Hint: Try to do the request on telnet first, by connecting to the *host* on port 80. Check resources online (i.e. Wikipedia) on how to make a valid HTTP request.

---

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

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

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

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

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

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

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

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

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

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

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

Hint: You might want to extend this function to also handle folders, such that you can store the pages or images relative to the root page you start your crawl on. However, you are not requested to do so.

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

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

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

(a) use `getPage` to retrieve the HTTP response

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

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

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

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

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

Solution:

Problem 2.81 (Ranking pages)
In this task you will gain some practical experience with a real-world web crawler and you will come up with your own page ranking procedure!

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

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

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

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

Solution:

2.5.6 Security by Encryption

Nothing here yet

2.5.7 An Overview over XML Technologies

Nothing here yet.

2.5.8 The Semantic Web

nothing here yet

2.6 Legal Foundations of Information Technology

2.6.1 Intellectual Property, Copyright, and Licensing

nothing here yet

2.6.2 Information Privacy

nothing here yet

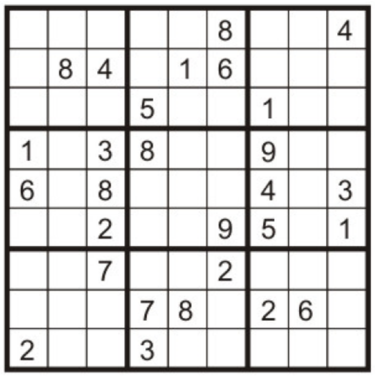
3 Search and Declarative Computation

3.1 Problem Solving and Search

3.1.1 Problem Solving

[5] EdN:5

Problem 3.1 (Sudoku)



					8			4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8				4		3
		2			9	5		1
		7			2			
			7	8		2	6	
2			3					

This question will give you an excuse to play Sudoku (see www.websudoku.com for explanation) while doing homework. Consider using search to solve Sudoku puzzles: You are given a partially filled grid to start, and already know there is an answer.

- Define a state representation for Sudoku answer search. A state is a partially filled, valid grid in which no rows, column, or 3x3 square contains duplicated digits. Also specify what transitions would be.
- If the puzzle begins with 28 digits filled, what is L, the length of the shortest path to goal using your representation?
- On a typical PC, which search algorithm would you choose: BFS, DFS or IDS? Why?

Solution:

- A 9x9 matrix whose elements are 1 to 9 or 0 as empty. Transitions are any valid filling of an empty cell. Only valid matrices (no duplication in rows, column, or 3x3 squares) are allowed.
- $L = 81 - 28 = 53$
- DFS is the most suitable, and actually almost all Sudoku search program use DFS. B, the branching factor, can be in the range of 9×53 so $B >> L$. Hence BFS and IDS will have serious memory problems on a typical PC. Since L is fixed so $L = L_max = L_min$, DFS is the only choice here.

---

[5] EdNote: we should extract some problem formulation sub-problems from e.g. moving-knight

Problem 3.2 (Define Problem Formulation)
Define the concept of *Problem Formulation*.

Solution: See definition 13.3 from slide 158.[6]

[6] EdNote: need to replace this by an sref

Problem 3.3: Does a finite state space always lead to a finite search tree? How about a finite space state that is a tree? Justify your answers.

Problem 3.4 (Problem formulation)
You and your roommate just bought an 8 liter jug full of beer. In addition you have two smaller empty jugs that can hold 5 and 3 liters respectively. Being good friends you want to share the beer equally. For this you need to split the amount in two separate jugs and each should contain exactly 4 liters. Write a formal description of this problem. What is one possible solution? What is the cost of your solution?

Solution:
We encode the states as three digits where the first digit is the amount of beer in the 8 liter jug, the second digit is the amount in the 5 liter jug and the last digit is the amount in the 3 liter jug.

- Initial state: 800
- Actions:
    1. $pour8in5$
    2. $pour8in3$
    3. $pour5in3$
    4. $pour5in8$
    5. $pour3in5$
    6. $pour3in8$

  Each of these actions pours beer from one jug to anonther until either the first jug is empty or the second jug is full. The corresponding successor function S can be derived from these actions.
- Goal test: $x = \mathbf{440}$
- Path cost: The amount of actions we perform to reach the solution.

A sample solution is:

$$[pour8in5, pour5in3, pour3in8, pour5in3, pour8in5, pour5in3, pour3in8]$$

$$800 \rightarrow 350 \rightarrow 323 \rightarrow 620 \rightarrow 602 \rightarrow 152 \rightarrow 143 \rightarrow 440$$

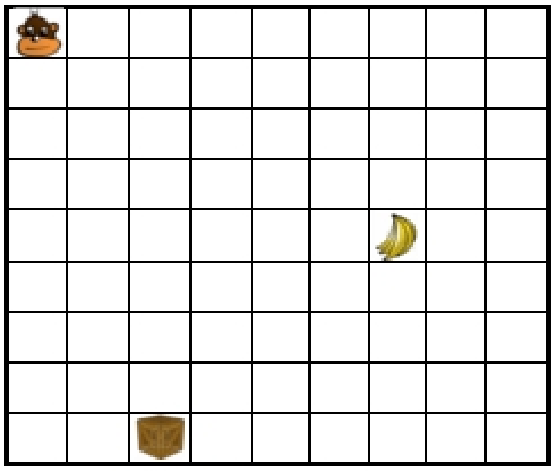
It has a cost of 7.

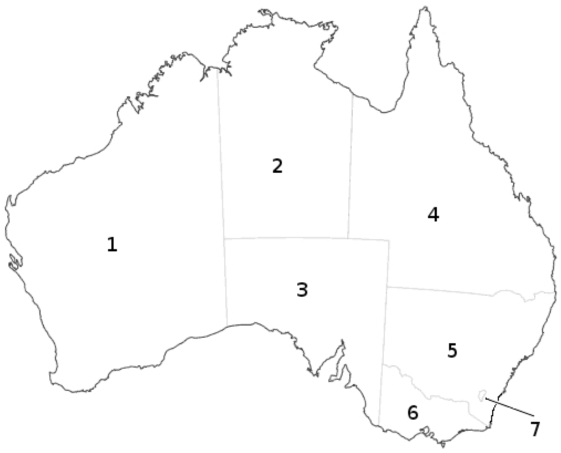
Problem 3.5 (Problem formulation and solution) 20pt 20min

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.





Hint: Refer to the slides for specifications regarding problem formulation and solution. Path cost is a function that assigns cost to every operator.

Solution:

a) 1.

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

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 3.6 (Search of the max element) 8pt 10min

Formalize the task of finding the maximum element in a set of the integer numbers. What are the properties of your search? Justify your answers.

Solution: The properties to be considered are completesness, time, space, optimality.

3.1.2 Search

[7]

Problem 3.7 (The Dog/Chicken/Grain Problem)

EdNote: 20min 20pt

A farmer wants to cross a river with a dog, a chicken, and a sack of grain. He has a boat which can hold himself and either of these three items. He must avoid that either dog and chicken or chicken and grain are together alone on one river bank, since otherwise something gets eaten.

1. Represent the farmer's problem of crossing the river without losing his goods as a search problem.
2. Draw a sufficiently large portion of the search tree induced by this problem to exhibit a solution.
3. Discuss three search strategies and their advantages and disadvantages in this scenario.

Hint: The farmer can also take something back over the river.

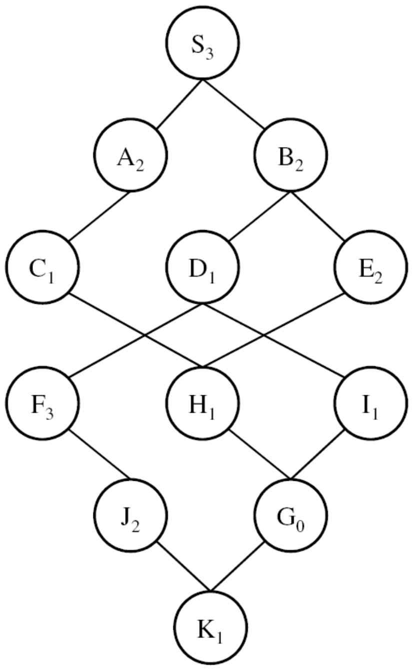
Solution: We present the states as a pair $\langle S, T\rangle$ of sets, where S is the set of items of the on the start bank and T that of items on the target bank. The initial state is $\langle\{f,d,c,g\},\emptyset\rangle$ and the goal state is $\langle\emptyset,\{f,d,c,g\}\rangle$. The actions are represented as $cross(A)$, where A is the item taken over $back(A)$ the action of taking A back.

$$\begin{array}{l}
\langle\{f,d,c,g\},\emptyset\rangle \longrightarrow^{c(c)} \langle\{d,g\},\{f,c\}\rangle \longrightarrow^{b} \langle\{f,d,g\},\{c\}\rangle \longrightarrow^{c(d)} \\
\langle\{g\},\{f,d,c\}\rangle \longrightarrow^{b(c)} \langle\{f,c,g\},\{d\}\rangle \longrightarrow^{c(g)} \langle\{c\},\{f,d,g\}\rangle \longrightarrow^{b} \\
\langle\{f,c\},\{d,g\}\rangle \longrightarrow^{c(c)} \langle\emptyset,\{f,d,c,g\}\rangle
\end{array}$$

[7] EdNote: need to take these problems apart, so that they do not mention specific search strategies

Problem 3.8 (Moving a Knight)
Consider the problem of moving a knight on a 3x4 board, with start and goal states labeled as S and G in the figure below. The search space can be translated into the following graph. The letter in each node is its name and you do not need to worry about its subscript for now.

S_3	H_1	D_1	K_1
I_1	J_2	A_2	E_2
C_1	B_2	G_0	F_3



Make the following assumptions:

- The algorithms do not go into infinite loops (i.e. once a node appears on a path, it will not be considered again on this path)
- Nodes are selected in alphabetical order when the algorithm finds a tie.

Write the sequence of nodes in the order visited by the specified methods (until the goal is reached). Note: You may find it useful to draw the search tree corresponding to the graph above.

- DFS
- BFS

Solution:

- DFS : S A C H E B D F J K G
- BFS : S A B C D E H F I G

3.1.3 Uninformed Search Strategies

Problem 3.9 (Uninformed Search)
Explain all uninformed search strategies introduced in class and compare their advantages and disadvantages with respect to completeness, time, space, and optimality.

Solution: Breadth-first search Uniform-cost search Depth-first search Depth-limited search Iterative deepening search

[8] EdN:8

Problem 3.10 (Sudoku)

					8			4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8				4		3
		2			9	5		1
		7			2			
			7	8		2	6	
2			3					

This question will give you an excuse to play Sudoku (see www.websudoku.com for explanation) while doing homework. Consider using search to solve Sudoku puzzles: You are given a partially filled grid to start, and already know there is an answer.

- Define a state representation for Sudoku answer search. A state is a partially filled, valid grid in which no rows, column, or 3x3 square contains duplicated digits. Also specify what transitions would be.
- If the puzzle begins with 28 digits filled, what is L, the length of the shortest path to goal using your representation?
- On a typical PC, which search algorithm would you choose: BFS, DFS or IDS? Why?

Solution:

- A 9x9 matrix whose elements are 1 to 9 or 0 as empty. Transitions are any valid filling of an empty cell. Only valid matrices (no duplication in rows, column, or 3x3 squares) are allowed.
- $L = 81 - 28 = 53$
- DFS is the most suitable, and actually almost all Sudoku search program use DFS. B, the branching factor, can be in the range of 9×53 so $B >> L$. Hence BFS and IDS will have serious memory problems on a typical PC. Since L is fixed so $L = L_max = L_min$, DFS is the only choice here.

---

[8] EdNote: we need to take the sudoku problem apart and only have the third bullet point here

Problem 3.11: Describe a state space in which iterative deepening search performs much worse than depth-first search (for example $O(n^2)$ vs. $O(n)$).

Solution: Depth-First search strategy performs great if the solutions are dense. Consider an abstract situation where we have many possible solutions and they are roughly at the same depth. Furthermore let the solution with minimal depth be at a high depth level.

To make the situation more concrete consider a search problem with the following parameters:

- $b = 100$
- $d = 33$
- $m = 36$

Assume furthermore that the solutions are very dense (most leaves represent a possible solution). Here depth-first will find a solution extremely fast while iterative-deepening will take much more time to reach the optimal solution at depth 33.

Problem 3.12 (Actions with Negative Costs)
Suppose that actions can have arbitrary large negative costs.

1. Explain why this possibility would force any optimal algorithm to explore the entire state space.
2. Does it help if we insist that step costs must be greater than or equal than to some negative constant c? Justify your answer.

Problem 3.13 (Implementing Search) 50pt

Implement the depth-first and breadth-first search algorithms in SML. The functions `depthFirst` and `breadthFirst` 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 the goal state together with a list of actions that reaches the goal state from the initial state.

Hint:

1. Write an auxiliary function that takes the fringe (i.e. a list of unexpanded states together with the plans to reach them) as an accumulator argument.
2. It is always good to treat the failure case with an exception.
3. The problem may become simpler to think about, if you first write a function that does not care about actions, which makes `next` simpler and also the return actions of the auxiliary function.

Solution: We will follow the hint and write a simple function first and later extend it to the full case.

```
exception search_exhausted
fun depthFirst next goal x =
    let fun dfs [] = raise search_exhausted
          | dfs (state::rest) = if goal(state) then state
                                else dfs (next state @ rest)
     in dfs [x] end;
\smlout{val depthFirst = fn : ('a -> 'a list) -> ('a -> bool) -> 'a -> 'a}

fun breadthFirst next goal x =
    let fun bfs [] = raise search_exhausted
          | bfs (state::rest) = if goal(state) then state
          else bfs (rest @ next state)
     in bfs [x] end;
\smlout{val breadthFirst = fn : ('a -> 'a list) -> ('a -> bool) -> 'a -> 'a}
```

Note that the programs only differ in the order of the arguments in the recursive call of the local function. Now, we extend the functions to deal with actions. Here we add the plans how to get to the fringe node to the states in the argument of the local function. Thus we need to add the current plan to the actions in the recursive call.

```
fun depthFirst next goal x =
    let fun dfs [] = raise search_exhausted
          | dfs ((plan,state)::rest) =
            if goal(state) then plan
            else dfs ((map (fn ((act,st)) => (act::plan,st)) (next state)) @ rest)
     in rev(dfs [(nil,x)]) end;
\smlout{val depthFirst = fn : ('a -> ('b * 'a) list) -> ('a -> bool) -> 'a -> 'b list}
fun breadthFirst next goal x =
    let fun bfs [] = raise search_exhausted
          | bfs ((plan,state)::rest) =
            if goal(state) then plan
            else bfs (rest @ (map (fn ((act,st)) => (act::plan,st)) (next state)))
     in rev(bfs [(nil,x)]) end;
\smlout{val breadthFirst = fn : ('a -> ('b * 'a) list) -> ('a -> bool) -> 'a -> 'b list}
```

Problem 3.14 (Implementing Search)
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.

Hint:

1. Write an auxiliary function that takes the fringe (i.e. a list of unexpanded states together with the plans to reach them) as an accumulator argument.
2. It is always good to treat the failure case with an exception.
3. The problem may become simpler to think about, if you first write a function that does not care about actions, which makes `next` simpler and also the return actions of the auxiliary function.

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*)
```

[9]

Problem 3.15 (A Trip Through Romania) EdN:9 30pt

Represent the Romanian map we talked about in class in a concrete `next` function. Search with the procedures from Problem 1.13 a trip from Arad to Bucharest. Compare the solution paths and run times.

Solution:

```
datatype State = Arad | Zerind | Oradea | Timisoara | Sibiu | Lugoj | Mehadia |
                             RimnicuVilcea | Fagaras | Pitesti | Craiova | Dobreta | Giurgiu |
                             Bucharest | Urziceni | Hirsova | Eforie | Vaslui | Iasi | Neamt;

datatype Actions = goAra | goZer | goOra | goTim | goSib | goLug | goMeh | goRim | goFag | goPit |
                               goCra | goDob | goGiu | goBuc | goUrz | goHir | goEfo | goVas | goIas | goNea;

fun next Arad = [(goZer, Zerind), (goSib, Sibiu), (goTim, Timisoara)] |
       next Timisoara = [(goAra, Arad), (goLug, Lugoj)] |
       next Zerind = [(goAra, Arad), (goOra, Oradea)] |
       next Oradea = [(goZer, Zerind), (goSib, Sibiu)] |
       next Sibiu = [(goAra, Arad), (goOra, Oradea), (goRim, RimnicuVilcea), (goFag, Fagaras)] |
       next Lugoj = [(goTim, Timisoara), (goMeh, Mehadia)] |
       next Mehadia = [(goLug, Lugoj), (goDob, Dobreta)] |
       next Dobreta = [(goMeh, Mehadia), (goCra, Craiova)] |
       next Craiova = [(goDob, Dobreta), (goPit, Pitesti), (goRim, RimnicuVilcea)] |
       next RimnicuVilcea = [(goCra, Craiova), (goPit, Pitesti), (goSib, Sibiu)] |
       next Pitesti = [(goRim, RimnicuVilcea), (goCra, Craiova), (goBuc, Bucharest)] |
       next Fagaras = [(goSib, Sibiu), (goBuc, Bucharest)] |
       next Bucharest = [(goPit, Pitesti), (goFag, Fagaras), (goGiu, Giurgiu), (goUrz, Urziceni)] |
       next Giurgiu = [(goBuc, Bucharest)] |
       next Urziceni = [(goBuc, Bucharest), (goHir, Hirsova), (goVas, Vaslui)] |
       next Hirsova = [(goEfo, Eforie), (goUrz, Urziceni)] |
       next Eforie = [(goHir, Hirsova)] |
       next Vaslui = [(goUrz, Urziceni), (goIas, Iasi)] |
       next Iasi = [(goVas, Vaslui), (goNea, Neamt)] |
       next Neamt = [(goIas, Iasi)];

fun goal Sibiu = true |
       goal _ = false;

val RESdepth = depthFirst(Giurgiu, next, goal);
val RESbreadth = breadthFirst(Giurgiu, next, goal);

(* The result is:
val RESdepth = (Bucharest,[goZer,goOra,goSib,goRim,goCra,goPit,goBuc])
: State * Actions list
val RESbreadth = (Bucharest,[goSib,goFag,goBuc]) : State * Actions list

We know that BFS always finds the optimal solution, and in our case
this is actually so. We go to Bucharest in 3 actions, while DFS has found a path with 7 actions.
Depending of where a solution it, BFS may prove to be faster than DFS. However, DFS will use less
memory in literally all cases. Our test case is too simple to notice any difference *)
```

[9]EdNote: make a separate problem in formaulation from the problem representation in SML and reference this here.

Problem 3.16 (Relations between search strategies) 15pt

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:

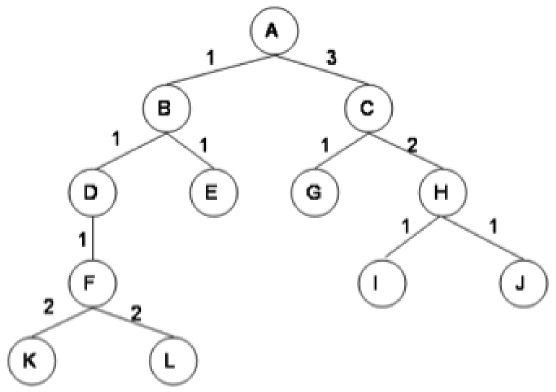
1. Let's consider a UCS on a search tree where the cost of each action is the same. Then the cost of a node will be proportional to its depth (distance from initial node). Therefore, the smallest-cost nodes in that way will actually be the shallowest nodes. So our UCS will expand first the shallowest 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 3.17 (Search Strategy Comparison on Tree Search)
Consider the tree shown below. The numbers on the arcs are the arc lengths.



Assume that the nodes are expanded in alphabetical order when no other order is specified by the search, and that the goal is state G. No visited or expanded lists are used. What order would the states be expanded by each type of search? Stop when you expand G. Write only the sequence of states expanded by each search.

Search Type	Sequence of States
Breadth First	
Depth First	
Iterative Deepening (step size 1)	
Uniform Cost	

Problem 3.18 (Missionaries and cannibals)
Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. The final goal is to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place.

1. Formulate the problem precisely. When defining the operators, it is not necessary that you write every possible *state* $\rightarrow$ *state* combination, but you should make it clear how one would derive the next state from the current one.
2. Suppose the next-function for depth first search (DFS) and breadth first search (BFS) expands a state to its successor states using the operators you have defined in 1. in the order you have defined them. Operators that leave more cannibals than missionaries on one side will not be considered. Likewise, operators that lead to the immediate previous state will not be considered (e.g., after moving a cannibal from left to right, the next-function for this state will not include a state where a cannibal moves from right to left). Draw the search tree till depth 3. What are the first 5 nodes explored by DFS? What are the first 5 nodes explored by BFS?
3. If you would implement this problem, would you rather use BFS or DFS to find the solution? Briefly explain why?

Solution:

1. Here is one possible representation:

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

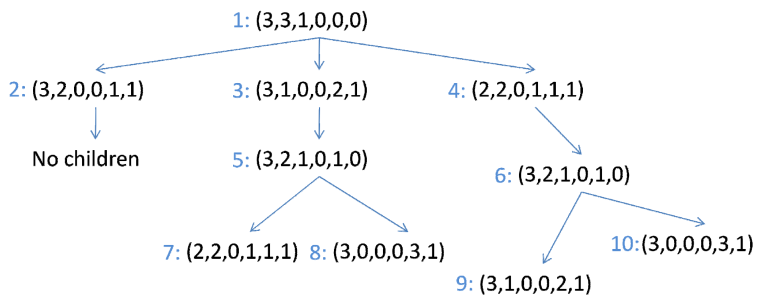
   The **Initial State** is $(3, 3, 1, 0, 0, 0)$

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

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

   (a) Move a missionary to the other side.
   (b) Move a cannibal to the other side.
   (c) Move 2 missionaries to the other side.
   (d) Move 2 cannibals to the other side.
   (e) Move a missionary and a cannibal to the other side.

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



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

Problem 3.19: 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. 50pt

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*)
```

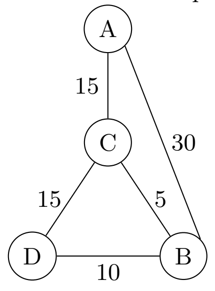
Problem 3.20 (Interpreting Search Results) 15pt

The state of Ingushetia has only four cities (A, B, C, and D) and a few two-way roads between them, so that it can be modeled as an undirected graph with four nodes. The task is to go from city A to city D. The UCS algorithm finds a solution to this task that is 10km shorter than the one BFS finds. The solution of BFS in turn is 10km shorter than the one of the DFS algorithm.

Draw a map of Ingushetia with roads and their distances that satisfies both conditions. What paths between A and D in your map will be found as solutions by each of those algorithms?

Note: All algorithms had repetition checking implemented, so that when a node is expanded, all its children that belong to a list of previously expanded nodes during the execution of that algorihtm are ignored. In addition, when no order of choosing a node for expansion is specified by an algorithm, expansion in alphabetical order takes place.

Solution: One possible solution:



The paths found are:

UCS ACBD (30km)

BFS ABD (40km)

DFS ABCD (50km)

A second solution: All possible edges should be present in the graph, except AD. The weigths are given as : AB = 10, AC = 10, AD is not in the graph, BC = 20, BD = 20, CD = 10. The solutions are then: DFS: ABCD , with cost = 40; BFS: ABD, with cost = 30; UCS: ACD, with cost = 20.

(Thanks to Darko Pesikan for suggesting this problem.)

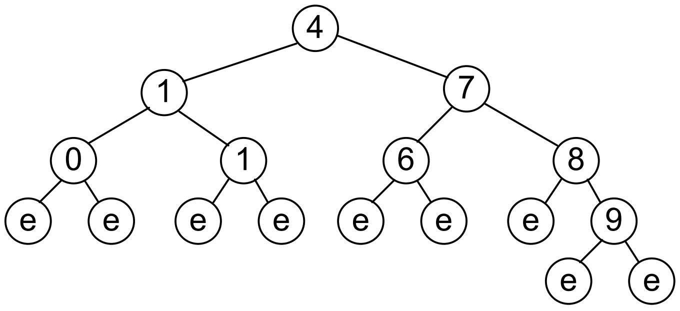
Problem 3.21 (Treesort Function) 14pt

Your task is to write a treesort function in SML that sorts a list of integers by first creating a binary search tree from the list and then loading the tree (in a sorted order) back into a list.

Use the following definition of a binary search tree:

- All leaves are empty nodes.
- All internal nodes carry a value and a left and a right subtree.
- The values of all nodes in a node's left subtree are smaller than the node's value and all nodes in its right subtree are greater or equal to the node's value.

The following tree is an example of a binary search tree:



Given the following datatype:

```
datatype searchtree = empty | node of searchtree*searchtree*int;
```

The tree above would be represented as follows:

```
node(node(node(empty,empty,0),node(empty,empty,1),1),
node(node(empty,empty,6), node(empty,node(empty,empty,9),8),7) , 4);
```

Write the functions using the searchtree datatype. The function sort should be of the following type:

```
fn treesort: int list -> int list
```

Solution:

```
datatype searchtree = empty | node of searchtree*searchtree*int;

(* insert a new value into a binary search tree *)
fun insert empty new = node(empty,empty,new)
 | insert (node(left,right,n)) new = if (n > new)
   then node((insert left new),right,n)
   else node(left,(insert right new),n);

(* create a binary search tree from a list *)
fun maketree list = foldl (fn (a,b) => insert b a) empty list;

(* loads a binary search tree into a list in a sorted manner *)
fun load empty = []
 | load (node(left,right,n)) = (load left)@[n]@(load right);

(* first makes a tree from the original values and them loads them
  back in a sorted order *)
fun treesort list = load (maketree list);
```

Problem 3.22 (Power Source Search)
A robot is on the 5x5 map shown below. It wants to reach a power source, but its sensors only allow it to detect the source once it is in the same cell with it. Find a problem formulation in the quadruple format presented in the lecture such that depth first search will find a solution after expanding exactly 6 nodes.

Assume that the `next` function of the DFS algorithm used returns the `(action, state)` tuples in the order in which the corresponding operators are defined. For example, if your operators are `jump` and `sing`, then the next function called on state i would return a list `[(jump, state j), (sing, state k)]` and not the other way around. (this is just an example, these operators will not do a very good job ... :))

Define a path cost for this problem. What is the cost of this solution? Is the solution optimal?

How many node expansions would BFS make considering the same `next` function?

			R	
P				P

R represents the robot and P a power source.

Solution:

```
S = {1,2,...25}
I = 9 //initial state
G = 20 //goal state
O = {down, right, up, left} // this is why the
//order in which they are specified matters, like this you get 5 expansions
```

```
DFS will go
9 -> 14 (down) -> 19 (down) -> 24 (down) now there is no down available so ->
25 (right) -> no down, no right, so up takes u to 19, goal state
```

So it expands 6 nodes.

With cost 1 per move, the cost of the solution is 5 and it is not optimal

BFS would find the solution at depth 3, so for this it will make all the expansions until depth 2, so $1 + 4 + 16$, then on the last level, the solution would be the second node expanded because the correct path is down down right. The first node will correspond to down down down, but the next will be the right solution. So $1 + 4 + 16 + 2 = 23$.

[10] EdN:10

Problem 3.23 (Maximum independent set)

An independent set of vertices in a graph G is a set where no two vertices are adjacent. The maximum independent set of vertices in a graph is an independent set with the greatest number of vertices. This number is denoted as $\alpha(G)$.

- Using what we have learned about search, how can you construct a representation that can be used to find a maximum independent set in a graph?
- What search algorithm is most appropriate?
- Estimate the number of maximum independent sets in a graph

Solution:

- Let G be a graph in which we want to find a maximum independent set. Pick the arbitrary vertex a. Let G_1 be a graph which is obtained by deleting the vertex a from the graph G, i.e. $G_1 = G - a$. And let G_2 be a graph which is obtained from G by deleting all vertices which are adjacent to a.

  Now let X be an arbitrary independent set of G. If $a \notin X$, then X is an independent set of G_1. If $a \in X$, then X is an independent set of G_2. Note that $\#(G) = \#(G_1) + 1$ and if a is not isolated then $\#(G) > \#(G_2)$. Thus we reduce the problem to the same two problems for smaller graphs and $\alpha(G) = \max \alpha(G_1), \alpha(G_2)$, i.e. we find maximum independent sets of graphs G_1 and G_2 and choose the maximum.

  We can build the following tree for our search algorithms. The root of a tree is G. An inner vertex of a tree is associated with a subgraph X and arbitrary vertex x to be removed. Left and right children of X are graph X_1 and X_2. Leaves of a tree are graphs without any edges and at the same time are independent sets of G.
- The DFS could be applied. Using that it is not necessary to store information about the whole tree in memory. We can store only those parts which are enough for finding the leaves.
- The number of maximum independent sets are less or equal to the number of leaves.

---

[10]EdNote: take the next problem apart as well.

3.1.4 Informed Search Strategies

Problem 3.24 (A looping greedy search)
Draw a graph and give a heuristic so that a greedy search for a path from a node A to a node B gets stuck in a loop. Draw the development of the search tree, starting from A, until one node is visited for the second time.

Indicate, in one or two sentences, how the search algorithm could be modified or changed in order to solve the problem without getting stuck in a loop.

Solution:

1. The example from the lecture, i. e. traveling through Romania.
2. Use A^*, or remember which nodes have been visited before and don't visit them again.

Problem 3.25 **(A^* Theory)**
What is the condition on the heuristic function that makes A^* optimal? Does a heuristic with this condition always exist?

Solution: Admissible heuristic - always underestimates the real cost to the goal. This always exists: $h(x) = 0$.

Problem 3.26 (A variant of A^*) 10pt

Imagine an algorithm B^* that uses the evaluation function $f(n) = g(n) \cdot h(n)$, where $g(n)$ is the path cost to the current node n, and $h(n)$ is a heuristic function. Is this algorithm better or worse than A^*? Explain your findings. What does $h(n)$ represent?

Solution: Comment by Andrei Aiordachioaie, to be formatted...

It's not that complicated and it leaves room for creativity to students. It would be nice to see how exactly they think :) As I see it, it's worse than A^* because heuristics h(n) needs extreme values for different nodes. When n is close to the root, h(n) needs to be very big to estimate a realistic distance to the goal, while if n is near the goal, the heuristic has to be very small. We will therefore need to work with floating-point numbers, which are a bit slower :)

Problem 3.27 (True or False on A^*)
True or False? Explain why.

1. A^* search always expands fewer nodes than DFS does.
2. For any search space, there is always and admissible and monotone A^* heuristic.

Solution:

1. A^* search always expands fewer nodes than DFS does.

   False. No optimal search algorithm can be more effcient than A^*, but DFS is not optimal and it can be lucky in searching goals. For example on the Sudoku search problem above, DFS almost always expands fewer nodes than A^*.
2. For any search space, there is always and admissible and monotone A^* heuristic

   True. For example $h(s) = 0$.

Problem 3.28 (Sudoku Revisited)

					8			4
	8	4		1	6			
			5			1		
1		3	8			9		
6		8				4		3
		2			9	5		1
		7			2			
			7	8		2	6	
2			3					

Remember the Sudoku problem from the last homework. You were asked which search algorithm you would choose on a typical PC: BFS, DFS or IDS. Is A^* better than your first choice? What is an admissible heuristic for A^*?

Solution: Same argument applies: the branching factor is huge, so A^* will run into memory troubles. An admissible heuristics for A^* is the number of cells left, but this does not differentiate in terms of choice (i.e. at any depth in the tree all nodes have the same heuristic value).

Problem 3.29 (Monotone heuristics) 12pt 15min

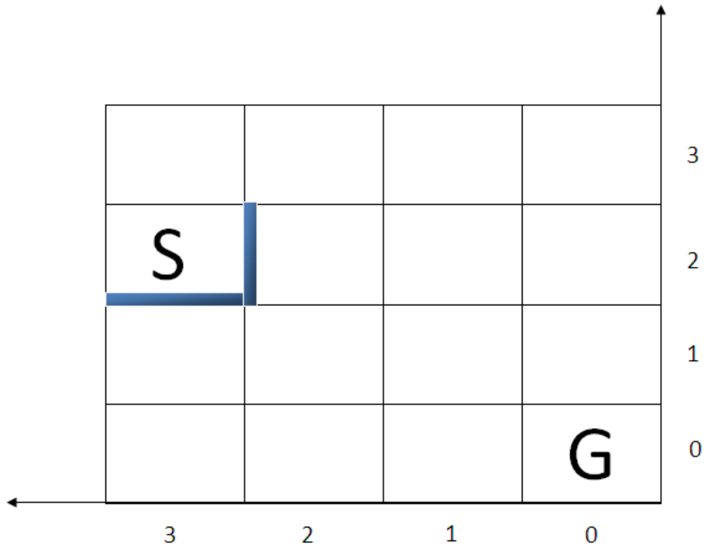
Let $c(n, a, n')$ be the cost for a step from node n to a successor node n' for an action a. A heuristic h is called *monotone* if $h(n) \leq h(n') + c(n, a, n')$. Prove or refute that if a heuristic is monotone, it must be admissible. Construct a search problem and a heuristic that is admissible but 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 3.30 (**A Good Old Friend, the Maze**)



Given a maze like the one above, consider using search to find the way from start to goal. The shaded areas are walls. You start from S and can only go left, right, up or down (unless there is a wall). All movements cost the same. The heuristic function is the Manhattan distance, $h = |x_1 - x_2| + |y_1 - y_2|$. For the following questions, explanations are required (simple answer is not enough).

1. Is this an admissible heuristic for A^* for the maze problem?
2. Is it an admissible heuristic if you can move in 8 directions instead of 4 (so also diagonally), if any movement still costs the same?
3. Which performs better with this heuristic, A^* or simple Greedy?
4. For the case of moving in all 8 directions, is the Euclidean distance, $h_e = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, admissible?
5. For the case of moving in all 8 directions, provide an admissible heuristic that is different from h and h_e, call it h_1, such that h_1 is non-trivial (non-constant and not the hardcoded actual cost).
6. Getting back to the 4 direction movement, is h_e more efficient for A^* than h?

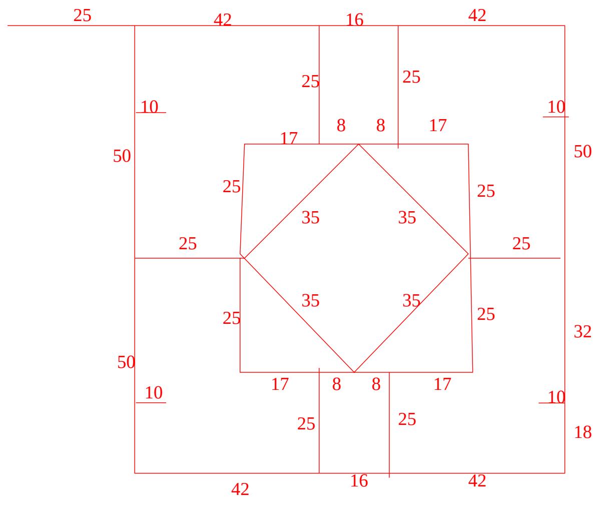
Solution:

1. Yes, the Manhattan distance always underestimates the cost - in the case of walls constricting the path it will be strictly smaller than the actual cost.
2. No, it overestimates. In the example above, S has Manhattan distance 5 but can reach the goal in 4 steps.
3. A^* doesn't get stuck like Greedy.
4. No, because the position diagonally next to the goal has a cost of 1 and a Euclidean distance of $\sqrt{2} > 1$
5. Divide the Euclidean distance by $\sqrt{2}$ to fix the problem.
6. In 4 moves, Manhattan is a closer estimate of the real cost and will thus perform better.

Problem 3.31 (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 3.32 (Relaxed Problem)
The relaxed version of a search problem P is a problem P' with the same states as P, such that any solution of P is also a solution of P'. More precisely, if s' is a successor of s in P, it is also a successor in P' with the same cost. Prove or refute that for any state s, the cost $c'(s)$ of the optimal path between s and the goal in P' is an admissible A^* heuristic for P.

Hint: Think about the graphical representation of the problems.

Solution: Graphically, P' has all the arcs from P, plus maybe more. This means that the optimal path in P' is the same as the optimal path in P, or better through the possibility of using the additional arcs. Therefore $c'(s) \leq c(s)$, which is the admissibility criterion for a heuristic.

Problem 3.33 (Relations between search strategies) 15pt

Uniform-cost search is a special case of A^* search.

Solution: Consider an A^* search such that the estimated cost to the goal is given by $h(n) = 0$ for all nodes n. Now $f(n) = g(n) + h(n) = g(n) + 0 = g(n) =$"the path cost for n". Since $f(n)$ is our evaluation function, desirability will be actually the path cost from the initial node to n, therefore cheapest path-cost nodes will be exapanded first, which is the behaviour of UCS.

Problem 3.34 (Global Solutions)
For each of the following algorithms, briefly state why or why not they are guaranteed to converge to a global optimum on a problem P:

1. A^* search with the heuristic from the problem above
2. Greedy search with the same heuristic
3. Hill Climbing
4. Genetic Algorithms

Solution:

1. search with the heuristic from the problem above: will converge to a global solution because the heuristic is admissible
2. Greedy search with the same heuristic: no, it actually might not converge at all because it can get stuck in infinite loops, as it doesn't take into account the distance so far, like A^*
3. Hill Climbing: no, it will just climb the current "hill", which does not necessarily have the biggest "hight"
4. Genetic Algorithms: no, there is no guarantee that GA's find the optimal solution. The quality of a solution of course depends on how cross-overs, mutations and other settings are chosen, and it can happen that the population simply cannot reach the region in the solution space in which the global optimum is.

3.1.5 Local Search

Problem 3.35 (Local Search)
What is a *local* search algorithm?

1. What does the “fringe” known from generic search algorithms look like in a local search algorithm?
2. What is the space complexity of local search?
3. Name two practical applications for local search.
4. Name a simple algorithm for local search. Give a brief overview of its advantages and disadvantages.

Problem 3.36 (Greedy vs. Hill Climbing)
What is the fundamental difference between Greedy Search and Hill Climbing? Explain.

Solution: HC is local search, i.e. the path is not kept because we are only interested in finding a solution and not how we got there.

Problem 3.37 (Local Beam Search)
What known algorithm does Local Beam Search become if $k = 1$?

Solution: The beam is 1, so there is only one starting point, so Hill Climbing.

Problem 3.38 (Hill Climbing) 40pt

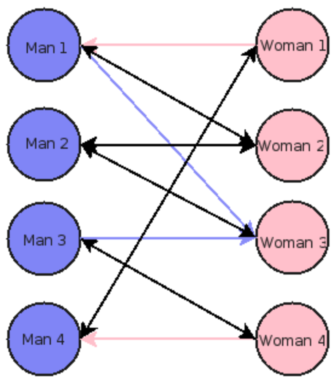
Consider a world with equal number of women and men. Every man is interested in a nonnegative number of women and vice versa. You are given a matrix that specifies a directed graph of *interest* between the people. Write an SML function that uses local search to find a pairing `{<man,woman>,<man,woman>,...}` 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
[(false,false),(false,false),(true,false),(true,true)],[(true,true),(false,false),(false,false),(fal
```

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
[(false,false),(false,false),(true,false),(true,true)],[(true,true),(false,false),(false,false),(fal
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>,<` and `{<man 1,woman 2>,<man 2,woman 1>,<man 3,woman 3>}` 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 <

        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
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,woman 1)
val test3 = compare_res (f init3 matrix2) [(man 2,woman 3),(man 1,woman 2),(man 3,woman 4),(man 4,woman 1)
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 3.39 (Easter Bunnies in Boxes)
Imagine there are n Easter bunnies and n different coloured boxes, and each bunny has specific color preferences and will like their box on a scale of 1 to 10. We want to makes as many bunnies as happy as we can, so the overall fitness of an assignment of bunnies in boxes will be the sum of how much each bunny likes its box. An assignment is admissible if each bunny has exactly 1 box. 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. Don't take the term crossover too literally though - it is not a must that you split the chromosomes and cross over their parts, you can think about the concept of reproduction in general. Similarly for mutation.

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 3.40 (Implementing simulated annealing)
Write an SML function that implements the simulated annealing algorithm to find the x value where a function $f(x)$ has a maximum. Your function should take the following arguments:

- `f : real->real` the SML implementation of $f(x)$
- `(a,b) : real*real` an interval $[a;b]$ in which to search for the maximum
- `schedule : int->real` a function that maps time steps to temperature values

For example the maximum of $f(x) = -(x-2)^2$ in $[0.0;5.0]$ is at $x = 2.0$. Given a good temperature schedule your implementation should be able to compute the maximum of $sin(x)$ with an accuracy of 0.0001. Show this at the end of your program by computing the maximum of $sin(x)$ in the interval $[0.0;5.0]$.

The complete signature of the function should look like this:
`find_max : (real -> real) -> real * real -> (int -> real) -> real`

Solution:

```
val min_temp = 0.000001;

(* First set random seed. *
val rand_state =
  let
              val now = Time.toMicroseconds ( Time.now () );
        val x = IntInf.div(now,1000);
        val y = IntInf.toInt (IntInf.mod(x,10000));
        in
              Random.rand (y,y)
        end;

(* picks a random point in the interval [current-eps;current+eps] with a lower limit of a and
   an upper limit of b*)
fun get_random_successor current a b eps =
  let
    val random_num = Random.randReal rand_state;
    val upper = if ( current + eps) > b then b else current + eps;
    val lower = if (current - eps) < a then a else current - eps;
  in
    lower + ( random_num * (upper-lower) )
  end;

(* Choses the next state based on the current state the temperature and the energy difference. *)
fun pick_with_probability current next deltaE temp =
  if deltaE > 0.0
  then next
  else
    if (Random.randReal rand_state) < Math.exp(deltaE/temp)
    then next
    else current;

(* Uses the simulated annealing algorithm to find the maximum of f in the interval [a,b] given
   the specified schedule. Furthermore this function must know the current solution, time and an
   epsilon value. The epsilon value is used to limit the neighborhood in which successor states
   can be chosen.*)
fun sim_ann f a b schedule time current eps =
  let
    val temp = schedule time;
    val next = get_random_successor current a b eps;
    val deltaE = (f next) - (f current);
  in
    if temp < min_temp
      then current
      else sim_ann f a b schedule (time+1) (pick_with_probability current next deltaE temp) eps
   end;

(* Uses the simulated annealing algorithm to find the maximum of f in the interval [a,b] given
   the specified schedule. *)
fun find_max f (a,b) schedule = sim_ann f a b schedule 0 ((a+b)/2.0) ((b-a)/10.0);
```

```
(* TESTING *)
fun schedule_lin time = 2.0 - 0.0002 * real(time);
fun schedule_exp time = Math.pow(0.95, real(time) ) * 50.0;

fun compute_num_steps schedule time =
  if schedule time < min_temp then time else compute_num_steps schedule (time+1);

val test_lin = Math.sin (find_max Math.sin (0.0,5.0) schedule_lin);
val test_lin_ok = (1.0 - test_lin) < 0.0001;
val test_lin_steps = compute_num_steps schedule_lin 0;

val test_exp = Math.sin (find_max Math.sin (0.0,5.0) schedule_exp);
val test_exp_ok = (1.0 - test_exp) < 0.0001;
val test_exp_steps = compute_num_steps schedule_exp 0;
```

Problem 3.41 (Simulated annealing schedules)
In the simulated annealing algorithm one has to choose a temperature schedule. Two possible schedules are:

- **The linear cooling scheme:** $T_{k+1} = T_k - \alpha = T_0 - (k+1) * \alpha$
- **The exponential cooling scheme:** $T_{k+1} = \alpha T_k = \alpha^{k+1} T_0$ where $\alpha < 1.0$ (the typical value is 0.95, but this really depends on the problem - and the smaller this is, the less iterations you will have).

The exponential cooling scheme typically performs better. Explain why this might be the case. To help you with this you should do an experiment where you try to achieve the desired accuracy in the pevious question by using both a linear and an exponential schedule.

Solution: It is important that near the end of our search we focus on only good solutions. By this time ideally we should be in the region of the global maximum and we should use the last iterations in order to find an optimal solution within the current "hill". The exponential cooling schedule allows just that. In the beginning it is still possible to make big jumps therefore escaping local maxima but at the end it leaves quite a lot of time where the algorithm tries to fine tune the current solution (because with low temperatures, the probability of accepting a worse state is very small, which means the state improves or stays the same at each iteration, thus allowing for a (randomized) hill-climbing like search for the top of the hill). The exponential schedule allows for more time at smaller temperatures.

To achive similar performance with the linear schedule we typically need a lot more iterations.

Problem 3.42 (Simulated Annealing)
Assume that you are using Simulated Annealing to solve the 8Queens problem. The SA is at a point where $T = 3$, the energy (fitness) of the current state is $E_{current} = 7$ and the energy of the neighboring state is $E_{neighbor} = 4$. With what probability will the neighbor be accepted as the new state and why?

Solution: $e^{\frac{4-7}{3}} = \frac{1}{e}$

3.2 Logic Programming

3.2.1 Introduction to Logic Programming and PROLOG

nothing here

3.2.2 Programming as Search

These exercises should be tried by everybody. They will confront you with the main (conceptual) problems of programming PROLOG, like relational programming, recursion, and a term language.
Problem 3.43: Build a database of facts about flight connections from Bremen Airport and write some query predicates for connections. Consider it is furthermore plausible to assume that whenever it is possible to take a flight from A to B, it is also possible to take a flight from B to A.

3.2.3 Logic Programming as Resolution Theorem Proving

No problems supplied yet.

References

[Koh11a] Michael Kohlhase. General Computer Science; 320101: GenCS I Lecture Notes. Online course notes at `http://kwarc.info/teaching/GenCS1/notes.pdf`, 2011.

[Koh11b] Michael Kohlhase. General Computer Science: 320201 GenCS II Lecture Notes. Online course notes at `http://kwarc.info/teaching/GenCS2/notes.pdf`, 2011.

[Koh11c] Michael Kohlhase. General Computer Science; Problems for 320101 GenCS I. Online practice problems at `http://kwarc.info/teaching/GenCS1/problems.pdf`, 2011.

[Koh11d] Michael Kohlhase. General Computer Science: Problems for 320201 GenCS II. Online practice problems at `http://kwarc.info/teaching/GenCS2/problems.pdf`, 2011.

[MM00] Meinel and Mundhenk. *Mathematische Grundlagen der Informatik*. Teubner, 2000.