1 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 [Gen11a, Gen11c]. 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 [Gen11b, Gen11d], 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

1 Preface **1**

2 Getting Started with "General Computer Science" **3**
2.1 Overview over the Course . . . 3
2.2 Administrativa . . . 3
2.3 Motivation and Introduction . . . 3

3 Elementary Discrete Math **4**
3.1 Mathematical Foundations: Natural Numbers . . . 4
3.2 Naive Set Theory . . . 7
3.3 Naive Set Theory . . . 9
3.4 Relations and Functions . . . 10

4 Computing with Functions over Inductively Defined Sets **11**
4.1 Standard ML: Functions as First-Class Objects . . . 11
4.2 Inductively Defined Sets and Computation . . . 23
4.3 Inductively Defined Sets in SML . . . 26
4.4 A Theory of SML: Abstract Data Types and Term Languages . . . 32
4.4.1 Abstract Data Types and Ground Constructor Terms . . . 32
4.4.2 A First Abstract Interpreter . . . 35
4.4.3 Substitutions . . . 38
4.4.4 A Second Abstract Interpreter . . . 42
4.4.5 Evaluation Order and Termination . . . 43
4.5 More SML: Recursion in the Real World . . . 46
4.6 Even more SML: Exceptions and State in SML . . . 47

5 Encoding Programs as Strings **60**
5.1 Formal Languages . . . 60
5.2 Elementary Codes . . . 64
5.3 Character Codes in the Real World . . . 70
5.4 Formal Languages and Meaning . . . 71

6 Boolean Algebra **72**
6.1 Boolean Expressions and their Meaning . . . 72
6.2 Boolean Functions . . . 83
6.3 Complexity Analysis for Boolean Expressions . . . 85
6.4 The Quine-McCluskey Algorithm . . . 91
6.5 A simpler Method for finding Minimal Polynomials . . . 96

7 Propositional Logic **101**
7.1 Boolean Expressions and Propositional Logic . . . 101
7.2 Logical Systems and Calculi . . . 102
7.3 Proof Theory for the Hilbert Calculus . . . 103
7.4 The Calculus of Natural Deduction . . . 110

8 Machine-Oriented Calculi **110**
8.1 Calculi for Automated Theorem Proving: Analytical Tableaux . . . 110
8.2 Resolution for Propositional Logic . . . 119

2 Getting Started with “General Computer Science”

2.1 Overview over the Course

This should pose no problems

2.2 Administrativa

Neither should the administrativa

2.3 Motivation and Introduction

Problem 2.1 (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 2.2 (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 2.3 (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

---

3 Elementary Discrete Math

3.1 Mathematical Foundations: Natural Numbers

25pt

Problem 3.1 (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 3.2 (Natural numbers)

Prove 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 3.3 (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).

3.2 Naive Set Theory

25pt

Problem 3.4: 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 3.5: 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 3.6: 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.

3.3 Naive Set Theory

3pt

3min

Problem 3.7: Fill in the blanks in the table of Greek letters. Note that capitalized names denote capital Greek letters.

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

3.4 Relations and Functions

Problem 3.8 (Associativity of Relation Composition)

Let R, S, and T be relations on a set M. Prove 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 $\langle x, y\rangle \in (T \circ (S \circ R))$. $\square$

---

4 Computing with Functions over Inductively Defined Sets

4.1 Standard ML: Functions as First-Class Objects

Problem 4.1: 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 4.2: 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 Problem 4.1

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 4.3: 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]` ⇝ `[[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]]` ⇝ `[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 4.4 (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 4.5 (Decompressing binary lists)

Write an inverse function `UNZIPit` of the one written in Problem 4.4.

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 4.6: 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 4.7 (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)` ⇝ `"~317":string`
- `string2int` converts a suitable string to an integer, i.e. `string2int("444")` ⇝ `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 4.8: 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 4.9: Write a non-recursive variant of the `member` function from Problem 4.1 using the `foldl` function.

Solution:

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

Problem 4.10 (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 4.11 (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 4.12 (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;
```

4.2 Inductively Defined Sets and Computation

Problem 4.13: 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 4.14 (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 4.15: 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 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. □

□

4.3 Inductively Defined Sets in SML

8pt

8min

Problem 4.16: 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`.

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 4.17: 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 4.18: 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 (* division *)
                  | aemod of arithexp (* modulo *)
                  | aev of int (* variable *)
```

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

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 4.19 (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 4.20 (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 4.21 (*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 Problem 4.20. 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));
```

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

4.4.1 Abstract Data Types and Ground Constructor Terms

Problem 4.22: 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 4.23: 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 4.24 (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.

4.4.2 A First Abstract Interpreter

30pt

Problem 4.25: 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 4.26: 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 4.27: 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	/	/

4.4.3 Substitutions

4pt

5min

Problem 4.28 (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 $\tau(t)$).

Solution:

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

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

Problem 4.30 (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;
```

4.4.4 A Second Abstract Interpreter

20pt

Problem 4.31: 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 $2n-1 \mapsto n$.

4.4.5 Evaluation Order and Termination

4pt
10min

Problem 4.32: 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 4.33: 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 $\langle f{::}\mathbb{N} \to \mathbb{N}\,;\, \{f(n_{\mathbb{N}}) \rightsquigarrow s(f(n_{\mathbb{N}}))\}\rangle$ 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 4.34: Give the recursion relation of the abstract procedures in Problem 4.6, ??, ??, and Problem 4.25 and discuss termination. 15pt

---

Solution:

Problem 4.6: $\{\langle s(n), n\rangle \mid n \in \mathbb{N}\}$

??: all recursion relations are empty

??: the recursion relation is empty

Problem 4.25: the recursion relation for g is $\{\langle s(n), n\rangle \mid n \in \mathbb{N}\}$, the one for max is empty

---

4.5 More SML: Recursion in the Real World

No problems supplied yet.

4.6 Even more SML: Exceptions and State in SML

Problem 4.35 (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 4.36 (Operations with Exceptions)
Add to the functions from Problem 4.20 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 4.37 (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 4.38 (Transformations with Errors) 10pt

Extend the function from Problem 4.7 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 4.39 (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 4.40 (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 4.41 (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 4.42 (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 previosly 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 4.43 (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 4.44 (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;
```

---

5 Encoding Programs as Strings

5.1 Formal Languages

3pt
5min

Problem 5.1: 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. $s_1, s_4, s_6 \in L$[2]

---

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

Problem 5.2: 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:

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

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

Problem 5.3: 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 5.4 (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";
```

5.2 Elementary Codes

2pt

Problem 5.5: Given the alphabets $A = \{\mathtt{a}, 2\}$ 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 5.6 (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 5.7: 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 5.8 (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 5.9 (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 5.10 (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 Problem 5.6) 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
```

5.3 Character Codes in the Real World

No problems supplied yet.

5.4 Formal Languages and Meaning

No problems supplied yet.

6 Boolean Algebra

6.1 Boolean Expressions and their Meaning

15pt

Problem 6.1 (Boolean complements)
Prove 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 6.2: 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 6.3 (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 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 6.4: Given the following SML data types for Boolean formulae and truth values 20pt

```
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 *)
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 eval(_,bez) = myfalse
  | eval(_,beo) = mytrue
  | eval(f,bep(x,y)) =
    if (eval(f,x) = mytrue orelse eval(f,y) = mytrue)
    then mytrue else myfalse
  | eval(f,bet(x,y)) =
    if (eval(f,x) = mytrue andalso eval(f,y) = mytrue)
    then mytrue else myfalse
  | eval(f,bec(x)) = if eval(f,x) = myfalse
                     then mytrue else myfalse
  | eval(f,bev(n)) = f(n)
```

Problem 6.5: Given the SML data types from Problem 6.4, write a simplified version of the function using the built-in truth values in SML, i.e. an evaluation function `evalbib : (int -> bool) -> boolexp -> b` 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 6.6 (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 6.7: Write a function `beprint : boolexp -> string` that converts `boolexp` formulae from Problem 6.4 to E_{bool} strings. This should be the inverse function to the function `beparse` from Problem 6.6. 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 6.8: 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 6.9 (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:

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

Problem 6.10 (Boolean Equivalence)
Prove 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.

6.2 Boolean Functions

10pt

Problem 6.11 (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*
⟨F, F, F⟩	T	⟨T, F, F⟩	T
⟨F, F, T⟩	T	⟨T, F, T⟩	T
⟨F, T, F⟩	T	⟨T, T, F⟩	F
⟨F, T, T⟩	T	⟨T, T, T⟩	T

Problem 6.12 (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: $(x_1 + x_2 + \overline{x_3})\,(x_1 + \overline{x_2} + \overline{x_3})\,(\overline{x_1} + \overline{x_2} + \overline{x_3})$

6.3 Complexity Analysis for Boolean Expressions

Problem 6.13 (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 6.14 (Relations among polynomials) 5pt 6min

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

Problem 6.15: 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 $n^3 + 3 * n \leq n^3$.
$(logn^3) \in \Theta(log(n))$	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 6.16 (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 6.17: 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 6.18 **(Proof of Membership in Landau Set)** 3pt 10min

Prove by induction that 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 $(n+1)!^2 = n+1^2 * (n)!^2 \geq n+1^2 n^n$ by (IH). Hence, we have to show that $n+1^2 n^n \geq n+1^{n+1}$ which we do by the equivalence transformations $n+1^2 n^n \geq n+1^{n+1}$ and $n^n \geq n+1^{n-1}$ and now??? TODO !!!!!!

6.4 The Quine-McCluskey Algorithm

14pt

Problem 6.19 (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 6.20: 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 6.21 (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 6.22 (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:

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

6.5 A simpler Method for finding Minimal Polynomials

10pt

Problem 6.23 (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:

	$\overline{A}\,\overline{B}$	$\overline{A}B$	AB	$A\overline{B}$
$\overline{C}\,\overline{D}$	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 6.24 (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:

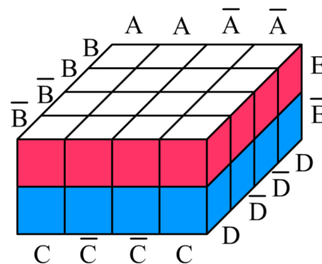
	$\overline{A}\,\overline{B}$	$A\overline{B}$	AB	$\overline{A}B$
$\overline{C}\,\overline{D}$	F	T	T	T
$C\overline{D}$	T	F	F	T
CD	F	T	T	T
$\overline{C}D$	T	T	T	F

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 6.25 (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 6.26 (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 6.27 (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.

7 Propositional Logic

7.1 Boolean Expressions and Propositional Logic

2pt

7min

Problem 7.1 (The *Nor* Connective)

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

7.2 Logical Systems and Calculi

Problem 7.2 (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).

7.3 Proof Theory for the Hilbert Calculus

5pt

Problem 7.3: 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: $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}\,\mathrm{MP}$ $\qquad \frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\,\mathrm{Subst}$

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

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

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 7.4 (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:

	Formula	Justification
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 7.5: 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: $\dfrac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}\,\mathrm{MP}$ $\qquad \dfrac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}\,\mathrm{Subst}$

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 7.6 (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:

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

Prove that the calculus is sound.

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

A	¬A	A ∨ ¬A
0	1	1
1	0	1

A	B	A ∨ B	B ∨ A	A ∧ B ⇒ B ∧ 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.

A	B	¬A	¬B	A ⇒ B	¬B ⇒ ¬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 7.7 (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:

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

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. $\vdash \mathbf{A} \Rightarrow \mathbf{A}$
$\vdash (P \Rightarrow Q) \Rightarrow (P \Rightarrow Q)$
$\vdash ((P \Rightarrow Q) \wedge P) \Rightarrow Q$

Problem 7.8 (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)

$\square$

Problem 7.9 (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)

□

7.4 The Calculus of Natural Deduction

No problems supplied yet.

8 Machine-Oriented Calculi

8.1 Calculi for Automated Theorem Proving: Analytical Tableaux

Problem 8.1: 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:

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

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

Problem 8.2: Prove the associative law for disjunction $(P \vee Q) \vee R \Leftrightarrow P \vee (Q \vee R)$[2] with the tableau method.

Solution:

$$(P \vee Q) \vee R \Leftrightarrow P \vee (Q \vee R)^{\mathsf{F}}$$

$(P \vee Q) \vee R^{\mathsf{F}}$			$(P \vee Q) \vee R^{\mathsf{T}}$		
$P \vee (Q \vee R)^{\mathsf{T}}$			$P \vee (Q \vee R)^{\mathsf{F}}$		
P^{F}			P^{F}		
Q^{F}			Q^{F}		
R^{F}			R^{F}		
P^{T}	Q^{T}	R^{T}	P^{T}	Q^{T}	R^{T}
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

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

Problem 8.3 (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:

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

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 8.4 **(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 8.5 (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:

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

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 8.6 (A *Nor* Tableau Calculus) 4pt 13min

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

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

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.

$$\frac{\mathbf{A} \downarrow \mathbf{B}^{\mathsf{T}}}{\begin{array}{c}\mathbf{A}^{\mathsf{F}} \\ \mathbf{B}^{\mathsf{F}}\end{array}} \qquad \frac{\mathbf{A} \downarrow \mathbf{B}^{\mathsf{F}}}{\mathbf{A}^{\mathsf{T}} \mid \mathbf{B}^{\mathsf{T}}} \qquad \frac{\begin{array}{c}\mathbf{A}^{\alpha} \\ \mathbf{A}^{\beta}\end{array} \quad \alpha \neq \beta}{\bot}$$

And the proof of the formula is

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

Problem 8.7 (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 8.8 (Automated Theorem Prover) 30pt

Building on the tableau procedure from Problem 8.7 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 8.9 (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

8.2 Resolution for Propositional Logic

10pt

Problem 8.10: Compute the Clause normal form of $(P \Leftrightarrow Q) \Leftrightarrow (R \Leftrightarrow P)$ with and without using the derived rules.

Problem 8.11: 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

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

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	$\Box$	with 6 and 7

Problem 8.12 (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 8.13 (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:

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

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:

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

Problem 8.14: 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 8.15: 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

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

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

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

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.

References

[Gen11a] General Computer Science; 320101: GenCS I Lecture Notes. Online course notes at `http://kwarc.info/teaching/GenCS1/notes.pdf`, 2011.

[Gen11b] General Computer Science; Problems for 320101 GenCS I. Online practice problems at `http://kwarc.info/teaching/GenCS1/problems.pdf`, 2011.

[Gen11c] General Computer Science: 320201 GenCS II Lecture Notes. Online course notes at `http://kwarc.info/teaching/GenCS2/notes.pdf`, 2011.

[Gen11d] 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.