

# 1 Preface

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

This document contains practice and homework problems for the material covered in the lecture (notes). The problems are tailored for understanding and practicing and should be attempted without consulting the solutions, which are available at [?, ?]

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 taught 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 assistants over the years: Darko Pesikan, Nikolaus Rath, Florian Rabe, Andrei Aiordachioaie, Dimitar Asenov, Alen Stojanov, Felix Schlesinger, Ștefan 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 Cflănaru, Maria Alexandra Alecu, Miroslava Georgieva Slavcheva, Corneliu-Claudiu Prodescu, Flavia Adelina Grosan, Felix Gabriel Mance, Anton Antonov, Alexandra Zayets, Ivaylo Enchev.

---

<sup>1</sup>International University Bremen until Fall 2006

# Contents

<b>1</b>	<b>Preface</b>	<b>1</b>
<b>2</b>	<b>Getting Started with “General Computer Science”</b>	<b>3</b>
2.1	Overview over the Course . . . . .	3
2.2	Administrativa . . . . .	3
2.3	Motivation and Introduction . . . . .	3
<b>3</b>	<b>Elementary Discrete Math</b>	<b>3</b>
3.1	Mathematical Foundations: Natural Numbers . . . . .	3
3.2	Naive Set Theory . . . . .	5
3.3	Naive Set Theory . . . . .	6
3.4	Relations and Functions . . . . .	7
<b>4</b>	<b>Computing with Functions over Inductively Defined Sets</b>	<b>8</b>
4.1	Standard ML: Functions as First-Class Objects . . . . .	8
4.2	Inductively Defined Sets and Computation . . . . .	10
4.3	Inductively Defined Sets in SML . . . . .	11
4.4	A Theory of SML: Abstract Data Types and Term Languages . . . . .	12
4.4.1	Abstract Data Types and Ground Constructor Terms . . . . .	12
4.4.2	A First Abstract Interpreter . . . . .	13
4.4.3	Substitutions . . . . .	14
4.4.4	A Second Abstract Interpreter . . . . .	15
4.4.5	Evaluation Order and Termination . . . . .	16
4.5	More SML: Recursion in the Real World . . . . .	17
4.6	Even more SML: Exceptions and State in SML . . . . .	18
<b>5</b>	<b>Encoding Programs as Strings</b>	<b>20</b>
5.1	Formal Languages . . . . .	20
5.2	Elementary Codes . . . . .	21
5.3	Character Codes in the Real World . . . . .	22
5.4	Formal Languages and Meaning . . . . .	23
<b>6</b>	<b>Boolean Algebra</b>	<b>24</b>
6.1	Boolean Expressions and their Meaning . . . . .	24
6.2	Boolean Functions . . . . .	25
6.3	Complexity Analysis for Boolean Expressions . . . . .	25
6.4	The Quine-McCluskey Algorithm . . . . .	26
6.5	A simpler Method for finding Minimal Polynomials . . . . .	27
<b>7</b>	<b>Propositional Logic</b>	<b>29</b>
7.1	Boolean Expressions and Propositional Logic . . . . .	29
7.2	Logical Systems and Calculi . . . . .	29
7.3	Proof Theory for the Hilbert Calculus . . . . .	29
7.4	The Calculus of Natural Deduction . . . . .	31
<b>8</b>	<b>Machine-Oriented Calculi</b>	<b>31</b>
8.1	Calculi for Automated Theorem Proving: Analytical Tableaux . . . . .	31
8.2	Resolution for Propositional Logic . . . . .	32

## 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.

#### Problem 2.2 (Keywords of General Computer Science)

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

#### 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.

## 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  $S_s = S \setminus \{s\}$  and  $S_t := S \setminus \{t\}$ . 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  $\square$

**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.

**Problem 3.3 (Peano's induction axiom)**

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

### 3.2 Naive Set Theory

**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  $\#(\mathcal{P}(A))$ )?

25pt

**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.

---

$$R_1 \subseteq A \times A, R_1 = \{\langle 23, 7 \rangle, \langle 7, 23 \rangle, \langle 5, 5 \rangle, \langle 17, 6 \rangle, \langle 6, 17 \rangle\}$$

$$R_2 \subseteq B \times B, R_2 = \{\langle 3, 3 \rangle, \langle 3, 23 \rangle, \langle 4, 4 \rangle, \langle 8, 23 \rangle, \langle 8, 8 \rangle, \langle 3, 4 \rangle, \langle 23, 23 \rangle, \langle 4, 23 \rangle\}$$

$$R_3 \subseteq B \times B, R_3 = \{\langle 3, 3 \rangle, \langle 3, 23 \rangle, \langle 8, 3 \rangle, \langle 4, 23 \rangle, \langle 8, 4 \rangle, \langle 23, 23 \rangle\}$$

**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

### 3.3 Naive Set Theory

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

3pt  
3min

Symbol					$\gamma$	$\Sigma$	$\pi$	$\Phi$
Name	alpha	eta	lambda	iota				

### 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

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

## 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`.

**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

---

**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 outputting two separate lists. `unzip [[1,4], [2,5], [3,6]] ~> [1,2,3] [4,5,6]`.

#### 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.

#### Problem 4.5 (Decompressing binary lists)

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

**Problem 4.6:** Program the function  $f$  with  $f(x) = x^2$  on unary natural numbers without using the multiplication function.

#### Problem 4.7 (Translating between Integers and Strings)

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`.

**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 =
[#" ", #" ", #"*", #" ", #" "],
[#" ", #"*", #"*", #"*", #" "],
[#"*", #"*", #"*", #"*", #"*"]]
```

**Problem 4.9:** Write a non-recursive variant of the `member` function from Problem 4.1 using the `foldl` function.

#### Problem 4.10 (Decimal representations as lists)

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
```

20pt

15pt

20pt

15pt

10min

---

**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.

---

**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 ??

10pt

**Problem 4.12 (Mapping and Appending)**

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

## 4.2 Inductively Defined Sets and Computation

**Problem 4.13:** Figure out the functions on natural numbers for the following defining equations

$$\begin{aligned}\tau(o) &= o \\ \tau(s(n)) &= s(s(s(\tau(n))))\end{aligned}$$

**Problem 4.14 (A function on natural numbers)**

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

$$\begin{aligned}\eta(o) &= o \\ \eta(s(o)) &= o \\ \eta(s(s(n))) &= s(\eta(n))\end{aligned}$$

**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  $\sigma$  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.

### 4.3 Inductively Defined Sets in SML

**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`.

8pt  
8min

Use SML syntax for the whole problem.

**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 `ms`, i.e. `nms(s(s(zero)),s(zero))` evaluates to `construct(s(zero),construct(s(zero),elist))`.

4pt  
8min

**Problem 4.18:** Given the following SML data type for an arithmetic expressions

```
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 *)
```

20pt

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

Write a (cascading) function `eval : (int -> int) -> arithexp -> int` that takes a variable assignment  $\varphi$  and an arithmetic expression  $e$  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  $\varphi$  of type `int -> int` that assigns  $\varphi(n)$  to the variable `aev(n)`.

#### 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.

#### 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)

#### Problem 4.21 (Nary 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(zero))))))))))` since  $223 = 12$ .

## 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.  $\langle \{\mathbb{S}\}, \{[c_1: \mathbb{S}], [c_2: \mathbb{S} \rightarrow \mathbb{S}], [c_3: \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{S}], [c_4: \mathbb{S} \rightarrow \mathbb{S} \rightarrow \mathbb{S}]\} \rangle$
2.  $\langle \{\mathbb{T}\}, \{[c_1: \mathbb{T}], [c_2: \mathbb{T} \times (\mathbb{T} \rightarrow \mathbb{T}) \rightarrow \mathbb{T}]\} \rangle$

5pt

**Problem 4.23:** Translate the given SML datatype

5min

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

into abstract data type in mathematical notation.

20pt

**Problem 4.24 (Nested lists)**

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  $[3, 4, [7, [8, 2], 9], 122, [2, 2]]$ .

#### 4.4.2 A First Abstract Interpreter

**Problem 4.25:** Give the defining equations for the maximum function for two numbers. This function takes two arguments and returns the larger one.

30pt

**Hint:** You may define auxiliary functions with defining equations of their own. You can use  $\iota$  from above.

**Problem 4.26:** Using the abstract data type of truth functions from ??, give the defining equations for a function  $\iota$  that takes three arguments, such that  $\iota(\varphi_{\mathbb{B}}, a_{\mathbb{N}}, b_{\mathbb{N}})$  behaves like “if  $\varphi$  then  $a$ , else  $b$ ”, where  $a$  and  $b$  are natural numbers.

15pt

**Problem 4.27:** Consider the following abstract data type:

6pt

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

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

Answer with <b>Yes</b> or <b>No</b> or <b>/</b> . and give the sort (if term)			
expression	term?	ground?	Sort
$f(g(a))$			
$f(g(\langle a, b \rangle))$			
$h(g(\langle h(x_{\mathbb{C}}), f(c) \rangle))$			
$h(g(\langle h(x_{\mathbb{B}}), f(y_{\mathbb{C}}) \rangle))$			

### 4.4.3 Substitutions

4pt

#### Problem 4.28 (Substitution)

5min

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

**Definition 1** We call a substitution  $\sigma$  **idempotent**, iff  $\sigma(\sigma(\mathbf{A})) = \sigma(\mathbf{A})$  for all terms  $\mathbf{A}$ .

**Definition 2** For a substitution  $\sigma = [\mathbf{A}_1/x_1], \dots, [\mathbf{A}_n/x_n]$ , we call the set **intro**( $\sigma$ ) :=  $\bigcup_{1 \leq i \leq n} \mathbf{free}(\mathbf{A}_i)$  the set of variables **introduced** by  $\sigma$ , and the set **supp**( $\sigma$ ) :=  $\{x_i \mid 1 \leq i \leq n\}$

30pt

**Problem 4.29:** Prove or refute that  $\sigma$  is idempotent, if **intro**( $\sigma$ )  $\cap$  **supp**( $\sigma$ ) =  $\emptyset$ .

30pt

#### Problem 4.30 (Substitution Application)

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  $\sigma$  and a term  $\mathbf{A}$  as arguments and returns the term  $\sigma(\mathbf{A})$  if  $\sigma$  is unambiguous and raises an exception otherwise.

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

#### 4.4.4 A Second Abstract Interpreter

20pt

**Problem 4.31:** Consider the following abstract procedure on the abstract data type of natural numbers:

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

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  $\mathcal{P}$ ?

#### 4.4.5 Evaluation Order and Termination

4pt

**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.

10min

---

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

---

2pt

**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.

5min

**Problem 4.34:** Give the recursion relation of the abstract procedures in Problem 4.6, ??, ??, and Problem 4.25 and discuss termination.

15pt

## 4.5 More SML: Recursion in the Real World

No problems supplied yet.

## 4.6 Even more SML: Exceptions and State in SML

5pt

### Problem 4.35 (Integer Intervals)

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.

### 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)

6pt

### Problem 4.37 (List Functions with Exceptions)

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.

10pt

### Problem 4.38 (Transformations with Errors)

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.

10pt

### Problem 4.39 (Simple SML data conversion)

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`

10pt

### Problem 4.40 (Strings and numbers)

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`

10pt

### Problem 4.41 (Recursive evaluation)

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

```

10pt

#### Problem 4.42 (List evaluation)

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

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

For example we have

```

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

```

10pt

#### Problem 4.43 (String parsing)

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]

```

10pt

#### Problem 4.44 (SML File IO)

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

```

## 5 Encoding Programs as Strings

### 5.1 Formal Languages

**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  $x \in L_i$ . 3pt  
5min

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

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

**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.

**Problem 5.3:** Let  $A := \{a, h, /, \#, x\}$  and  $\prec$  be the ordering relation on  $A$  with  $x \prec \# \prec / \prec h \prec a$ . Order the following strings in  $A^*$  in the lexical order  $<_{\text{lex}}$  induced by  $\prec$ . 3pt  
5min

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

**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.

## 5.2 Elementary Codes

2pt

**Problem 5.5:** Given the alphabets  $A = \{a, 2\}$  and  $B = \{9, \#, /\}$ .

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

30pt

**Problem 5.6 (Testing for prefix codes)**

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.

8pt

**Problem 5.7:** Let  $A := \{a, b, c, d, e, f, g, h\}$  and  $\mathbb{B} := \{0, 1\}$ , and

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

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

40pt

**Problem 5.8 (Morse Code Translator)**

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.

20pt

**Problem 5.9 (Morse Code again)**

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.

30pt

**Problem 5.10 (String Decoder without Stop Characters)**

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 an exception otherwise.

### 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

#### Problem 6.1 (Boolean complements)

15pt

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  $b = \bar{a}$ . (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.

10pt

**Problem 6.2:** Give a model for  $C_{bool}$ , where the following expressions are theorems:  $a * \bar{a}$ ,  $a + \bar{a}$ ,  $a * a$ ,  $\overline{a + a}$ .

---

**Hint:** Give the truth tables for the Boolean functions.

15pt

#### Problem 6.3 (Partial orders in a Boolean algebra)

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  $a/b = 1$ .

20pt

**Problem 6.4:** Given the following SML data types for Boolean formulae and truth values

```
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  $\varphi$  and a Boolean formula  $e$  and returns  $\mathcal{I}_\varphi(e)$  as a value.

20pt

**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 -> bool`. This function should not use any `if` constructs.

40pt

#### Problem 6.6 (Parsing boolean expressions)

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  $\neg(x)$  for the complement of  $x$  in the input syntax. So the relevant clause in the definition is now:

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

---

**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.

20pt

**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.

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.

3pt

**Problem 6.8:** Is the expression  $e := \overline{x123 * x72} + x123 * x4$  valid, satisfiable, unsatisfiable, falsifiable? Justify your answer.

5min

2pt

**Problem 6.9 (Evaluating Expressions)**

Let  $e := \overline{x_1 + x_2} + (\overline{x_2} * x_3 + x_3 * x_4)$  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.

7min

**Problem 6.10 (Boolean Equivalence)**

Prove the following equivalence:

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

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

## 6.2 Boolean Functions

10pt

**Problem 6.11 (Induced Boolean Function)**

Determine the Boolean function  $f_e$  induced by the Boolean expression  $e := (x_1 + x_2) * \overline{x_1 * x_3}$ . Moreover determine the CNF and DNF of  $f_e$ .

**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

## 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.:  $O(a) \subset O(b) \subset O(c) \equiv O(d) \subset O(e) \dots$ )

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

5pt

**Problem 6.14 (Relations among polynomials)**

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

6min

3pt

**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.

10min

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

**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.  $f(n) = \binom{n}{2}, g(n) = \binom{n}{4}$

**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

```

3pt

**Problem 6.18 (Proof of Membership in Landau Set)**

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  $n$ .

10min

---

**Hint:**

---

## 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  $(\bar{x}_1 + x_3)(x_3)$  is 3.

35pt

**Problem 6.20:** Use the algorithm of Quine-McCluskey to determine the minimal polynomial of the following functions:

$x_1$	$x_2$	$x_3$	$x_4$	$f_1$
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

$x_1$	$x_2$	$x_3$	$x_4$	$f_2$
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

15pt

**Problem 6.21 (Quine-McCluskey with Don't-Cares)**

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  $f(x_1, x_2, x_3, x_4)$  that yields T for  $x_1^0 x_2^1 x_3^0 x_4^0$ ,  $x_1^0 x_2^1 x_3^0 x_4^1$ ,  $x_1^0 x_2^1 x_3^1 x_4^0$ ,  $x_1^1 x_2^0 x_3^0 x_4^0$ ,  $x_1^1 x_2^0 x_3^0 x_4^1$ ,  $x_1^1 x_2^0 x_3^1 x_4^0$ ,  $x_1^1 x_2^1 x_3^0 x_4^1$ , "don't care" for  $x_1^0 x_2^0 x_3^0 x_4^0$ ,  $x_1^0 x_2^1 x_3^1 x_4^1$ ,  $x_1^1 x_2^1 x_3^1 x_4^1$ , and F for the other inputs.

14pt

**Problem 6.22 (CNF with Quine-McCluskey)**

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:  $a x + a \bar{x} = a$  also applies for formulas in CNF:  $(a + x)(a + \bar{x}) = a$

## 6.5 A simpler Method for finding Minimal Polynomials

10pt

**Problem 6.23 (Karnaugh-Veitch Minimization)**

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

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.

10min

**Problem 6.24 (Karnaugh-Veitch Diagrams)**

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.)

15pt

**Problem 6.25 (CNF with Karnaugh-Veitch Diagrams)**

6min

KV maps can also be used to compute a minimal CNF for a Boolean function. Using the function  $f(x_1, x_2, x_3)$  that yields T for  $x_1^0 x_2^0 x_3^0$ ,  $x_1^0 x_2^1 x_3^0$ ,  $x_1^0 x_2^1 x_3^1$ ,  $x_1^1 x_2^0 x_3^0$ , 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.

10pt

**Problem 6.26 (Karnaugh-Veitch Diagrams with Don't-Cares)**

In some cases, there is an input  $d \in \text{dom}(f)$  to a boolean function  $f: \mathbb{B}^n \rightarrow \mathbb{B}$  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.

10pt

**Problem 6.27 (Don’t-Care Minimization)**

1. Devise a concrete Boolean function  $f: \mathbb{B}^4 \rightarrow \mathbb{B}$  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

#### 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  $\neg\mathbf{A} \Leftrightarrow \mathbf{A} \downarrow \mathbf{A}$ .

---

2pt

7min

### 7.2 Logical Systems and Calculi

#### Problem 7.2 (Calculus Properties)

Explain briefly what the following properties of calculi mean:

- correctness
- completeness

### 7.3 Proof Theory for the Hilbert Calculus

**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:  $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$  MP  $\quad \quad \quad \frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}$  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  $\quad \quad \quad \frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}$  Subst

#### 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 preceding 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.  $\mathbf{B} \Rightarrow (\mathbf{A} \Rightarrow \mathbf{B})$

5pt

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.  $\mathbf{A} \Rightarrow \mathbf{A}$

**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:  $\frac{\mathbf{A} \Rightarrow \mathbf{B} \quad \mathbf{A}}{\mathbf{B}}$  MP                       $\frac{\mathbf{A}}{[\mathbf{B}/P]\mathbf{A}}$  Subst

**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:

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

Prove that the calculus is sound.

10pt

**Problem 7.7 (A calculus for propositional logic)**

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

10min

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

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

**Problem 7.8 (Hilbert Calculus)**

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

20pt

**Problem 7.9 (A Hilbert Calculus)**

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:

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

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

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

**Problem 8.2:** Prove the associative law for disjunction  $(P \vee Q) \vee R \Leftrightarrow P \vee (Q \vee R)$ <sup>2</sup> with the tableau method.

**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  $A \Leftrightarrow B^\top$ . Show the derivation.
3. Generate all models of the following expression:  $\neg Q \wedge P \Leftrightarrow Q \wedge \neg P$

11pt

**Problem 8.4 (Refutation and model generation in Tableau Calculus)**

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.

**Problem 8.5 (Tableau Calculus)**

14pt

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

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

**Problem 8.6 (A *Nor* Tableau Calculus)**

4pt

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.

13min

---

<sup>2</sup>Proving this in the Hilbert calculus from ?? takes about 300 steps.

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

$$\frac{\mathbf{A} \downarrow \mathbf{B}^T}{?} \quad \frac{\mathbf{A} \downarrow \mathbf{B}^F}{?} \quad \frac{\mathbf{A}^\alpha \quad \mathbf{A}^\beta \quad \alpha \neq \beta}{\perp}$$

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

35pt

**Problem 8.7 (Tableau Construction)**

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.

30pt

**Problem 8.8 (Automated Theorem Prover)**

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  $F^T$ ).

30pt

**Problem 8.9 (Testing the ATP)**

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:

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

4pt

**Problem 8.12 (Basics of Resolution)**

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?

5pt

**Problem 8.13 (Resolution Calculus with Nand Connective)**

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:

$$\frac{\mathbf{C} \vee \mathbf{A} \uparrow \mathbf{B}^{\top}}{?} \quad \frac{\mathbf{C} \vee \mathbf{A} \uparrow \mathbf{B}^{\text{F}}}{?}$$

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

25pt

**Problem 8.14:** Use the resolution method to prove the formulae from ??:

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:

$$\frac{\mathbf{A} \Rightarrow \mathbf{B}^{\text{F}}}{\begin{array}{c} \mathbf{A}^{\top} \\ \mathbf{B}^{\text{F}} \end{array}}$$

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

25pt

**Problem 8.15:** Consider the following two formulae where the first one is in conjunctive normal form and the second in disjunctive normal form

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.

## References