General Computer Science
320201 GenCS I & II Problems & Solutions

Michael Kohlhase

Computer Science
Jacobs University, Bremen Germany
`m.kohlhase@jacobs-university.de`

September 21, 2016

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 2003-2016. It is meant as a supplement to the course notes [Koh11a, Koh11b]. We try to keep the numbering consistent between the documents.

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

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

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

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

[1] International University Bremen until Fall 2006

Contents

Chapter 1

Getting Started with "General Computer Science"

1.1 Overview over the Course

This should pose no problems

1.2 Administrativa

Neither should the administrativa

1.3 Motivation and Introduction

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

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

Solution:

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

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

Solution:

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

---

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

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

Solution:

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

Chapter 2

Motivation and Introduction

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

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

Solution:

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

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

Solution:

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

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

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

Solution:

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

Part I

Representation and Computation

Chapter 3

Elementary Discrete Math

3.1 Mathematical Foundations: Natural Numbers

Problem 0.7 (A wrong induction proof)
What is wrong with the following "proof by induction"? 25pt

> **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 □

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

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

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

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

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

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

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

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

□

Problem 0.9 (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 Reasoning about Natural Numbers

Problem 0.10 (Zero is not one)
Prove or refute that $s(o)$ is different from o.

Note: Please use **only** the Peano Axioms for this proof.

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, and is different from o (axiom P2 and 1.)

□

Problem 0.11 (Natural numbers)
Prove or refute that $s(s(o))$ and $s(s(s(o)))$ are unary natural numbers and that their successors are different.

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

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

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

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

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

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

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

□

3.3 Defining Operations on Natural Numbers

Problem 0.12 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 0.13 (Commutativity of addition)
Prove by induction or refute the commutativity of addition in the case of natural numbers using only its definition from the slides. More specifically prove that: $n \oplus m = m \oplus n$

Hint: You might come to a point in the proof where a new subproblem emerges which needs a to be proven by induction. Better said, you are allowed (and suppose to) use **nested** induction.

Solution:
Proof: Proof by induction over m.

P.1 We have two cases:

P.1.1 Base case: $m = 0$:

P.1.1.1 $n \oplus 0 = n$ by definition *property 1*.

P.1.1.2 Next we have to pove that $0 \oplus n = n$ which was not defined.

P.1.1.3 We prove this by induction over n: Again we have two cases:

P.1.1.3.1 Base case: $n = 0$:

P.1.1.3.1.1 $0 \oplus 0 = 0$ *property 1 of definition*. □

P.1.1.3.2 Step case::

P.1.1.3.2.1 We assume that $0 \oplus n = n$.

P.1.1.3.2.2 Then, applying the definition of addition *property 2*, $0 \oplus s(n) = s(0 \oplus n) = s(n)$ □

P.1.1.4 Thus we know that $n \oplus 0 = 0 \oplus n$ □

P.1.2 Step case::

P.1.2.1 For the step case assume that $n \oplus m = m \oplus n$ and we have to prove that $n \oplus s(m) = s(m) \oplus n$.

P.1.2.2 By definition: $n \oplus s(m) = s(n \oplus m)$.

P.1.2.3 Applying the induction step and again the definition of addition *property 2*, we get that $s(n \oplus m) = s(m \oplus n) = m \oplus s(n) = m \oplus (n \oplus s(0)) = m \oplus (s(0) \oplus n) = (m \oplus s(0)) \oplus n$.

P.1.2.4 'Therefore we have that $n \oplus s(m) = s(m) \oplus n$.

P.1.2.5 This completes the step case. □

P.2 Therefore we have proven the property of commutativity by induction. □

Problem 0.14 (Unary Natural Numbers)
Let $\oplus$ be the addition operation and $\odot$ be the multiplication operation on unary natural numbers as defined on the slides. Prove or refute that:

1. $a \oplus b = b \oplus a$
2. $(a \oplus b) \odot c = a \odot c \oplus b \odot c$
3. $a \odot b = b \odot a$

Solution:

1. **Proof**: We proceed by induction over b:

   P.1 Base case:

   $$a \oplus 0 = 0 \oplus a$$

   P.1 W e have $a \oplus 0 = a$ by the addition axiom, so we need $a = 0 \oplus a$. This is easily proven by induction over a.

P.3 Step case: We first prove by induction over a that

$$s(b) \oplus a = b \oplus s(a)$$

P.3 B ase case: $a = 0$. We need $s(b) \oplus 0 = b \oplus s(0)$. But from the previous step we know:

$$s(b) \oplus 0 = 0 \oplus s(b) = s(b \oplus 0) = b \oplus s(0)$$

Step case, assume $s(b) \oplus a = b \oplus s(a)$. Now:

$$s(b) \oplus s(a) = s(s(b) \oplus a) = s(b \oplus s(a)) = b \oplus s(s(a))$$

Back to the problem, assume $a \oplus b = b \oplus a$ and let's prove that $a \oplus s(b) = s(b) \oplus a$.

P.4 W e have $a \oplus s(b) = s(a \oplus b) = s(b \oplus a) = b \oplus s(a)$. This is equal to $s(b) \oplus a$, which is what we need to prove. □

2. **Proof**: We proceed by induction over c.

P.1 Base case: $(a \oplus b) \odot 0 = a \odot 0 \oplus b \odot 0$.

P.1 T his is true by the fact that $\forall n. n \odot 0 = 0$.

P.3 Step case: Assume that $(a \oplus b) \odot c = a \odot c \oplus b \odot c$. We need $(a \oplus b) \odot s(c) = a \odot s(c) \oplus b \odot s(c)$

P.3 W e have, from the multiplication axioms:

$$\begin{aligned}(a \oplus b) \odot s(c) = (a \oplus b) \oplus (a \oplus b) \odot c = (a \oplus b) \oplus (a \odot c \oplus b \odot c) = \\ = (a \oplus a \odot c) \oplus (b \oplus b \odot c) = a \odot s(c) \oplus b \odot s(c)\end{aligned}$$

□

3. **Proof**: We proceed by induction over b.

P.1 Base case: $a \odot 0 = 0 \odot a$ can be easily proven by induciton over a.

P.2 Step case, assume $a \odot b = b \odot a$. We need $a \odot s(b) = s(b) \odot a$.

P.2 W e have:

$$\begin{aligned}a \odot s(b) = a \oplus a \odot b = s(0) \odot a \oplus b \odot a = \\ = (s(0) \oplus b) \odot a = (b \oplus s(0)) \odot a = s(b) \odot a\end{aligned}$$

where we used the fact that $a = s(0) \odot a$, something that can be quickly proven by induction over a. □

3.4 Naive Set Theory

25pt **Problem 0.15** 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.

15pt **Problem 0.16** Let $A := \{5, 23, 7, 17, 6\}$ and $B := \{3, 4, 8, 23\}$. Which of the relations are reflexive, antireflexive, symmetric, antisymmetric, and transitive?

Note: Please justify the answers.

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

Solution: R_1 is not reflexive since there are not all elements of A are in R_1 as pairs like (a, a) 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, $(23, 7)$ exists and $(7, 23)$ 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 (b,b) where $b \in B$. Therefore, it is not antireflexive. R_2 is not symmetric, because for a given pair (a,b) where $a,b \in B$ there does not exist a pair (b,a). R_2 is, however, antisymmetric since for any "turnable" pair (like $(3,3)$) 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 $(3,23),(3,4)and(4,23)$.

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

Problem 0.17 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 $y \in (B \cap A)$, $(x,y) \in P \Leftrightarrow (x,y) \in R \vee (x,y) \in Q$. 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 $C = \{c\}, A = B = \{a,b\}, R = \{(c,a)\}, Q = \{(c,b)\}$. Then $P = \{(c,a),(c,b)\}$ is not a partial function.

3.5 Talking (and writing) about Mathematics

3pt
3min

Problem 0.18 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

Problem 0.19 (Math Talk)
Write the following statement in mathtalk

"For all mandatory courses, there is a student such that if the student is from the major in which the course is mandatory then the student is not taking that course."

Solution: First we define:
C := "the set of all mandatory courses"
M := "the set of all majors"
S := "the set of all students"
$Mm := \{(x,y) \mid$ "x is a mandatory course in major y"$\}$
$Sm := \{(x,y) \mid$ "x is a student from the major y"$\}$
$Sc := \{(x,y) \mid$ "x is a student taking the course y"$\}$
With these definitions we can write the statement as: $\forall c \in C.\exists s \in S.\exists m \in M.((s,m) \in Sm \wedge (c,m) \in Cm) \Rightarrow (s,c) \notin Sc$

3.6 Relations

Problem 0.20 (Associativity of Relation Composition)

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

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

Solution:
Proof:

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

P.2 $\exists z_1 \in M \mathbf{.} (x, z_1) \in R \wedge (z_1, y) \in (T \circ S)$

P.3 $\exists z_1, z_2 \in M \mathbf{.} (x, z_1) \in R \wedge ((z_1, z_2) \in S \wedge (z_2, y) \in T)$

P.4 $\exists z_2 \in M \mathbf{.} (x, z_2) \in (S \circ R) \wedge (z_2, y) \in T$

P.5 $(x, y) \in (T \circ (S \circ R))$. □

3.7 Functions

Problem 0.21 Are the following functions total, injective, surjective and/or bijective?

- $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2 - 3x + 6$
- $g : \mathbb{N} \to \mathbb{N}, g(x)$ represents the number of distinct prime divisors of x
- $h \colon \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$,

$$h(m,n) := \frac{(m+n-2)(m+n-1)}{2} + m$$

Prove your answers.

Solution: f is total because $f(x)$ is defined for all real values of x. f is not injective because $f(0) = f(9) = 6$. f is also not surjective because the minimal value that f can obtain is $f(4.5) = 12.75$.
g is total because every natural number can be represented as a number of prime divisors. g is not injective because both 4 and 9 have the same amount of prime divisors(1). g is surjective because for any number of natural divisors k, the number $p_1 \cdot p_2 \cdot \ldots \cdot p_k$ will be a number that has exactly k prime divisors.
h is total because it takes a defined value for every two natural numbers m and n. h is not injective because $h(1,0) = h(1,1) = 1$. h is surjective(I will write the proof soon).

Problem 0.22 (Function Property)
15pt Prove or refute: if a mapping of a finite set to itself is injective then it is surjective as well.

Solution: Let A be a finite set and let $f\colon A \to A$. Let $a \in A$. Define $f^0(a) = a$. Because A is finite, then $f^i(a), 0{\leq}in$ cannot be all distinct if n is large enough. Thus at some point we have $f^r(a) = f^s(a)$, where $s{\leq}r$. Because f is injective, there is an inverse mapping f^{-1}. $f^{r-s}(a) = f^0(a) = a$ and so $a = f(f^{r-s-1}(a))$, where $f^{r-s-1}(a)$ is in A. Thus f is surjective.

Problem 0.23 (Composition of functions and relations)
Prove or refute that the composition of functions, as defined in the lecture, is compatible with the definition of functions as special relations. That is: Let $F \subseteq A \times B$ and $G \subseteq B \times C$ be relations that are total functions, and let $H \subseteq A \times C$ be defined as $H := G \circ F$, i. e. the composition of the relations F and G. Show that

1. the relation H is a total function.
2. for all $x \in A$, we have $h(x) = g(f(x))$.

Note: Note: F, G and H are written in capital letters here to point out that we are talking about relations. If we treat them as functions, they would usually be written in lowercase.

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

Proof: If F and G are total functions, then for every $a \in A$ there is exactly one $b \in B$ with $(a,b) \in F$, and for every $b' \in B$ there is exactly one $c \in C$ with $(b',c) \in G$. We have to prove that for every $a \in A$ there is exactly one $c \in C$ with $(a,c) \in H$. We first prove that H is total (i. e. that there is at least one such c) and then prove that H is a function (i. e. that there is at most one such c).

P.1 $G \circ F$ is total, because F is total. Thus, for every $a \in A$, there is an $F(a) \in B$ and hence a $G(F(a)) \in C$ so that $(a, G(F(a))) \in (G \circ F)$.

P.2 In order to show that there is at most one $c \in C$ with $(a,c) \in (G \circ F)$, we choose two arbitrary pairs (a,c) and $(a,c') \in (G \circ F)$. By the definition of composition of relations, there are two elements b and $b' \in B$ with $(a,b),(a,b') \in F$ and $(b,c),(b',c') \in G$. As F is a function, $b = b'$ follows; as G is also a function, we obtain $c = c'$. □

Chapter 4

Computing with Functions over Inductively Defined Sets

4.1 Standard ML: Functions as First-Class Objects

Problem 0.24 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 0.25 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 ?prob.member?

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 0.26 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] $\rightsquigarrow$ [[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]] $\rightsquigarrow$ [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 0.27 (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 0.28 (Decompressing binary lists)
Write an inverse function UNZIPit of the one written in ?prob.zipbin?.

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;
```

15pt **Problem 0.29** Program the function f with $f(x) = x^2$ on unary natural numbers without using the multiplication function.

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 0.30 (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) $\rightsquigarrow$ "~317":string
- string2int converts a suitable string to an integer, i.e. string2int("444") $\rightsquigarrow$ 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 0.31 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 0.32 Write a non-recursive variant of the member function from ?prob.member? using the foldl function.

Solution:

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

Problem 0.33 (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: 15pt 10min

```
- 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 0.34 (List functions via folding)

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 the previous problem.

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 0.35 (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 0.36 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 0.37 (A function on natural numbers)
Figure out the operation η on unary natural numbers defined by the following equations: 15pt

5min

$$\eta(o) = o$$

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

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

Solution:

The function η halves its argument.

Problem 0.38 In class, we have been playing with defining equations for functions on the natural numbers. Give the defining equations for the function σ with $\sigma(x) = x^2$ without using the multiplication function (you may use the addition function though). Prove from the Peano axioms or refute by a counterexample that your equations define a function. Indicate in each step which of the axioms you have used. 15pt

---

Solution:

Lemma 4.2.1 *The relation defined by the equations $\sigma(o) = o$ and $\sigma(s(n)) = +((+((\sigma(n), n)), n))$ 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 $\sigma(s(n)) = +((+((\sigma(n), n)), n))$

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 0.39** 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 *)
```

4pt 8min **Problem 0.40** 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)).

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));
```

20pt **Problem 0.41** 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 * arithexp (* division *)
                  | aemod of arithexp * arithexp (* modulo *)
                  | aev of int (* variable *)
```

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

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

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

Solution:

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

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

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

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

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

Problem 0.42 (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 0.43 (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 0.44 (*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 ?prob.natoper?. 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 2 2 3 = 12.

Solution:

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

Chapter 5

Abstract Data Types and Term Languages

5.1 Abstract Data Types and Ground Constructor Terms

Problem 0.45 Translate the abstract data types given in mathematical notation into SML datatypes 5pt 5min

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

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 0.46 Translate the given SML datatype 5pt

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

5min

into abstract data type in mathmatical notation.

Solution:

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

Problem 0.47 (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 $[3, 4, [7, [8, 2], 9], 122, [2, 2]]$. 20pt

Solution: We choose the abstract data type

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

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.

5.2 A First Abstract Interpreter

30pt **Problem 0.48** 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

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

15pt **Problem 0.49** Using the abstract data type of truth functions from ?prob.truth-values?, 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.

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

6pt **Problem 0.50** Consider the following abstract data type:

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

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	/	/

5.3 Substitutions

Problem 0.51 (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 $\tau(t)$). 4pt 5min

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

Definition 5.3.2 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 $\mathbf{supp}(\sigma) := \{x_i \mid 1\leq i\leq n\}$

Problem 0.52 Prove or refute that σ is idempotent, if $\mathbf{intro}(\sigma) \cap \mathbf{supp}(\sigma) = \emptyset$. 30pt

Problem 0.53 (Substitution Application)
Consider the following SML data type of terms: 30pt

```
  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;
```

5.4 Terms in Abstract Data Types

5.5 A Second Abstract Interpreter

20pt **Problem 0.54** 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 $\{(s(s(n)), n) \in (\mathbb{N} \times \mathbb{N}) \,|\, n \in \mathbb{N}\}$ (or $(n+2, n)$)
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 $2n-1 \mapsto n$.

5.6 Evaluation Order and Termination

Problem 0.55 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. 4pt 10min

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

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

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

Problem 0.56 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 0.57 Give the recursion relation of the abstract procedures in ?prob.square?, ?prob.truth-values?, ?prob.if?, and ?prob.max? and discuss termination. 15pt

Solution:
?prob.square?: $\{(s(n), n) \,|\, n \in \mathbb{N}\}$
?prob.truth-values?: all recursion relations are empty
?prob.if?: the recursion relation is empty
?prob.max?: the recursion relation for g is $\{(s(n), n) \,|\, n \in \mathbb{N}\}$, the one for max is empty

Chapter 6

More SML

6.1 More SML: Recursion in the Real World

No problems supplied yet.

6.2 Programming with Effects: Imperative Features in SML

6.2.1 Input and Output

nothing here yet.

6.2.2 Even more SML: Exceptions and State in SML

Problem 0.58 (Integer Intervals)

Declare an SML data type for natural numbers and one for lists of natural numbers in SML. Write an SML function that given two natural number n and m (as a constructor term) creates the list [n,n+1,\ldots,m−1,m] if $n{\leq}m$ and raises an exception otherwise. 5pt 10min

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 0.59 (Operations with Exceptions)

Add to the functions from ?prob.natoper? 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 0.60 (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 0.61 (Transformations with Errors)

10pt

Extend the function from ?prob.ML-int2string? 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]

Problem 0.62 (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;
```

[1] EdNote: need one; please help

```
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 0.63 (Strings and numbers)

Write two SML functions 10pt

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 and Real.fromString but do the string parsing yourself. You may however use the char_to_int from above.

- 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 0.64 (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 0.65 (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. 10pt

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

For example we have

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

Solution:

```
exception InvalidVariable of int;

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

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

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

fun evaluate_list l = evaluate_list_helper l nil;
```

Solution:

```
(* TEST CASES *)
val EPSILON = 0.0001;
fun eq a b = abs( a − b) < EPSILON;
fun eql nil nil = true
```

```
  | eql l nil = false
  | eql nil l = false
  | eql (a::l) (b::m) = (eq a b) andalso (eql l m);

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

val test5 = eql ( evaluate_list [var(~1)] ) [1.0]
                 handle InvalidVariable v => true| other => false;
val test6 = eql ( evaluate_list [var(0)] ) [1.0]
                 handle InvalidVariable v => true| other => false;
val test7 = eql ( evaluate_list [var(1)] ) [1.0]
                 handle InvalidVariable v => true| other => false;
val test8 = eql ( evaluate_list [num(1.0),var(1)] ) [1.0]
                 handle InvalidVariable v => true| other => false;
```

Problem 0.66 (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 0.67 (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. 10pt

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;
```

Bibliography

[Koh11a] Michael Kohlhase. General Computer Science; 320101: GenCS I Lecture Notes. Online course notes at `http://kwarc.info/teaching/GenCS1/notes.pdf`, 2011.

[Koh11b] Michael Kohlhase. General Computer Science: 320201 GenCS II Lecture Notes. Online course notes at `http://kwarc.info/teaching/GenCS2/notes.pdf`, 2011.

[Koh11c] Michael Kohlhase. General Computer Science; Problems for 320101 GenCS I. Online practice problems at `http://kwarc.info/teaching/GenCS1/problems.pdf`, 2011.

[Koh11d] Michael Kohlhase. General Computer Science: Problems for 320201 GenCS II. Online practice problems at `http://kwarc.info/teaching/GenCS2/problems.pdf`, 2011.