Name: Matriculation Number:

Midterm Exam
General CS I (320101)

October 12, 2010

You have 70 minutes(sharp) for the test;
Write the solutions to the sheet.

The estimated time for solving this exam is 0 minutes, leaving you 70 minutes for revising your exam.

You can reach 0 points if you solve all problems. You will only need 30 points for a perfect score, i.e. -30 points are bonus points.

Different problems test different skills and knowledge, so do not get stuck on one problem.

	To be used for grading, do not write here	
prob.	Sum	grade
total	0	
reached		

Please consider the following rules; otherwise you may lose points:

- Always justify your statements. Unless you are explicitly allowed to, do not just answer "yes" or "no", but instead prove your statement or refer to an appropriate definition or theorem from the lecture.
- If you write program code, give comments!

1 Mathtalk and Induction

Problem 1.1 (Greek Alphabet) 2pt in

Two CS students just noticed a big problem in their Physics Lab Reports which is due in a couple of minutes. The printer apparently didn't output all the symbols Greek alphabet symbols which were represented as

$$\langle symbol \rangle = \langle name\_of\_symbol \rangle$$

but only one of the two parts. Can you help them fill all the missing symbols and symbol names?

Symbol	δ	Υ	ν	ζ				
Name					gamma	mu	xi	Psi

Solution:

Symbol	δ	Υ	ν	ζ	γ	μ	ξ	Ψ
Name	delta	Upsilon	nu	zeta	gamma	mu	xi	Psi

Problem 1.2 (Function Definition) 2pt in

Let A and B be sets. State the definition of the concept of a partial function with domain A and codomain B. Also state the definition of a total function with domain A and codomain B.

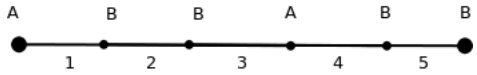
Solution: Let A and B be sets, then a relation $R \subseteq AB$ is called a **partial/total function**, iff for each $a \in A$, there is at most/exactly one $b \in B$, such that $\langle a, b\rangle \in R$.

Problem 1.3 (Segment Partition) 8pt in

Consider a line segment AB and take in its interior some (distinct) points. Denote each of these points either A or B.

Prove by induction or refute by a counterexample that the number of oriented segments AB exceeds by 1 the number of oriented segments BA. Note that oriented segments are minimal, i.e. they do not contain named points.

Consider, for example, the following diagram:



Here, 4 points are taken inside the segment AB and they are denoted B, B, A, B. As one can see, there are 2 oriented segments AB (1st and 4th) and 1 oriented segment BA (3rd).

Solution:

Proof: We prove the assertion by induction on the number on the segment (not counting its end points)

P.1 We have two cases

P.1.1 Base case:

P.1.1.1 If we take $n = 0$ (which means no additional points on the segment), we have just one oriented segment, AB.

P.1.1.2 Hence the number of ABs is 1 and the number of BAs is 0, so the hypothesis is satisfied. □

P.1.2 Step case:

P.1.2.1 Assume we have proven that for any n points on the segment,

$$\#(AB) - \#(BA) = 1 \tag{1}$$

P.1.2.2 Let's take n points on the segment and an arbitrary point X besides these n. Now we'll have $n + 1$ points and we want to prove that (**??**) still holds.

P.1.2.3 Consider our X is denoted A.

P.1.2.4 Then we have 4 cases given by the possible immediate neighbors

P.1.2.4.1 A **inserted between** A **and** A:Nothing will change as no new AB or BA segments are created. □

P.1.2.4.2 A **inserted between** A **and** B:The old AB segment is destroyed and from it the segments AA and AB are created, so the number of AB and BA segments remains constant. □

P.1.2.4.3 A **inserted between** B **and** A:Similarily with case 2, segment BA is destroyed and segments BA and AA are created, so again the number of AB and BA segments remains constant. □

P.1.2.4.4 *A* **inserted between** *B* **and** *B*:Here a segment BB is destroyed and instead a segment BA and a segment AB are created, so the difference between $\#(AB)$ and $\#(BA)$ is conserved. □

P.1.2.5 If X is B, the proof is analogous. □

P.2 Hence we have proven that $\#(AB) - \#(BA) = 1$ for all $n \in \mathbb{N}$ □

---

2 Sets and Functions

Problem 2.1 (Students and their Problems) 8 pt in

Consider that this year the GenCS course has n students and they have just received a huge homework of m problems. Let $1 \leq n, m$; let furthermore S be the set of students and P the set of problems. Also,

$$R := \{\langle s, p\rangle \in (S \times P) \mid \text{“}s \text{ solves problem } p\text{”}\}$$

1. What properties we covered in class hold on this relation?
2. How can one form a bijection between sets S and P? Please provide two solutions.
3. What is the maximum number of subsets of the relation R?

Justify your answers.

Solution:

1. Only antisymmetry will work here, due to the construction of the relation.
2. The only way a bijection can be formed is if $n = m$. The simplest way is to number the problems and assign each problem p_i to the student s_i. Having that, one can now consider all rotations, for example students s_i solve problems p_{i+2}, and of course s_{n-1} solves p_1 and s_n solves p_2.
3. Of course, the students have to realize here that they are asked to provide the cardinality of the powerset of elements in R, hence we are looking for $2^{n \cdot m}$.

Problem 2.2 (Testing for injectivity) 8pt

Check whether the following functions are injective or not :

1. $f\colon \mathbb{R} \to \mathbb{R}$ with $f(f(f(x))) = f(f(x)) + x$
2. $g\colon \mathbb{R} \to \mathbb{R}$ with $g(2^x) + g(3^x) = 1$

Solution:

1. **Proof**: We prove that $\forall x, y \in \Re. f(x) = f(y) \Rightarrow x = y$.

   P.1 Take x, y arbitrary, $f(x) = f(y) \Rightarrow f(f(x)) = f(f(y)) \Rightarrow f(f(f(x))) = f(f(f(y))) \Rightarrow f(f(f(x))) - f(f(x)) = f(f(f(y))) - f(f(y)$ (*)

   P.2 However, from the problem we know that $f(f(f(x))) - f(f(x)) = x$.

   P.3 Thus, from (*) we get $x = y$, which concludes our proof. □

   Proof: We prove that the function is not injective by a counterexample.

2. **P.1** Take $x = y = 1 \Rightarrow f(2) + f(3) = 1 \Rightarrow f(3) = 1 - f(2)$

   Then, consider $x = y = log_3 2 \Rightarrow 2^x = 2^{log_3 2} \wedge 3^x = 2 \Rightarrow f(2^{log_3 2}) + f(2) = 1 \Rightarrow$

   $$f(2^{log_3 2}) = 1 - f(2) \tag{2}$$

   From (1) and (2), $f(3) = f(2^{log_3 2})$, but $3 \neq 2^{log_3 2}$, which concludes our proof. □

---

3 Inductively Defined Functions

Problem 3.1 (Greatest Common Divisor) 50 min

Provide a definition for the function that finds the greatest common divisor of two unary natural numbers, considering the following inductive definitions of addition and multiplication on unary natural numbers. You may also create any other functions you may wish to use.

1. addition: $\alpha(n, o) = n$ and $\alpha(n, s(o)) = s(\alpha(n, m))$
2. multiplication: $\mu(n, o) = o$ and $\mu(n, s(m)) = \alpha(n, \mu(n, m))$

Hint: It is handy to use the Euclidean algorithm here. Recall:

- $gcd(n, 0) = n$
- $gcd(n, m) = gcd(m, n \ mod \ m)$

Create your own auxiliary functions on unary natural numbers and, using them, convert this definition of the greatest common divisor algorithm on the natural numbers into one on the unary natural numbers.

Solution: To convert the formulas for gcd to formulas on UNN, we need the definition of modulus.

The modulus $n \ mod \ m$ equals $(n - m) \ mod \ m$ in case that $n \geq m$ and returns n otherwise. The subtraction function always returns o if $n < m$, so we can make use of this fact. We will do that by applying a boolean function. Since the modulus function can return a value when it has reached a state when $n < m$, we can set the boolean function to true ($s(o)$), and return n. Before that, while we are still in a state when the boolean function returns false (o), we can just calculate $(n - m) \ mod \ m$. To include the effects of the boolean and the subtraction function, without influencing the output, we will be adding their product to $(n - m) \ mod \ m$. This does not change the output, because in case $n < m$, we add $(n - m)$ to zero (which is the remainder of zero divided by anything), and otherwise we add zero (because the boolean evaluates to zero) to $(n - m) \ mod \ m$.

For subtraction we will use the previous number function, i.e. which does the opposite of the successor function.

This is sufficient to convert the whole greatest common divisor formula to one that works on the unary natural numbers.

Here are the inductive definitions:

1. previous element:
   - $\pi(o) = o$
   - $\pi(s(n)) = n$
2. subtraction (which returns o if the $m > n$ in $m - n$):
   - $\delta(o, m) = o$
   - $\delta(n, o) = n$

- $\delta(n, s(m)) = \pi(\delta(n, m))$

3. boolean:
   - $\beta(o) = o$
   - $\beta(s(n)) = s(o)$
4. modulus:
   - $\varphi(o, m) = o$
   - $\varphi(n, m) = \alpha(\varphi(\alpha(n, m), m), \mu(\alpha(n, m), \beta(\delta(m, n))))$
5. greatest common divisor:
   - $\gamma(n, o) = n$
   - $\gamma(n, m) = \gamma(m, \varphi(n, m))$

---

4 Programming in Standard ML

Problem 4.1 (Secret Message) 50pt 50min

A couple of GenCS students wanted to develop a way to communicate with each other in text that makes it impossible for others to understand. Thus they thought of encoding characters as integers (greater than zero): each character from a list (they don't want the whole list of possible characters, just a few) gets an integer from another list. Since they just learned about SML, they considered writing some functions using it to ease their encoding and decoding of characters.

Your task is to implement two functions:

1. a function secretEncode that takes as arguments two lists of equal length: a list of characters and a list of integers and gives a list of pairs of the form (character, integer) and

2. a function getEncoding, that takes an character and the list of pairs (character,integer) and returns the integer encoding of that int (if no encoding found it return -1)

So we have the following signature:

val secretEncode = **fn** : char list $*$ int list $->$ (char $*$ int) list
val getEncoding = **fn** : char $*$ (char $*$ int) list $->$ int

Example:

$-$ val encoding = secretEncode([#"a",#"b",#"c"], [4,1,7]);
val encoding = [(#"a",4),(#"b",1),(#"c",7)] : (char $*$ int) list
$-$ getEncoding(#"b", encoding);
val it = 1 : int

Solution:

```
fun secretEncode([]: char list, lsa: int list) = []
  | secretEncode(ch::chs, a::lsa) = (ch, a)::secretEncode(chs,lsa);

fun getEncoding(ch: char, []) = ~1
  | getEncoding(ch, (c, x)::encods) =
                          if ch=c then x
              else getEncoding(ch, encods);
```

Problem 4.2 (Pearls) 14 min

Ariel, the Little Mermaid, was asked by Ursula to give all her beautiful colored pearls in exchange for getting transformed into a real girl. Ariel wanted very much to meet prince Eric in person, but wouldn't give away all her pearls, so she made a deal with Ursula: she were to keep the k most shiny pearls and give away all the others.

We know that Ariel's pearls were of four types: the carved pearls are not shiny at all and have a "shininess" multiplier of 0; opal-pearls have a multiplier of 2; the faceted pearls have a multiplier of 4; the akoya pearls have the highest multiplier, namely 10. Each pearl also has an intrinsic "shininess" that gets multiplied by the multiplier.

You are given the following SML datatype and function signature and you are asked to return the total value of the "shininess" that Ariel can get:

```
datatype pearl = carved of int | opal of int | faceted of int | akoya of int;
val shiny = fn : pearl list * int -> int;
```

For instance to compute

```
- shiny([carved(2), opal(3), faceted(1), faceted(5), akoya(2), opal(20)], 3);
val it = 80 : int;
```

we first compute the shininess of the five pearls: [0,6,4,20,20,40], and then add up the shininess of the $k = 3$ shiniest ones.

Solution: First idea: sort the pearls, get the k largest and add up their values.

```
fun len([]) = 0 | len (x::L) = 1+len(L);
fun take(L, 0) = [] | take([], _) = [] | take(x::L, n) = x::take(L,n-1);
fun drop(L, 0) = L | drop([], _) = [] | drop(x::L, n) = drop(L, n-1);

datatype pearl=carved of int | opal of int | faceted of int | akoya of int;

fun value(carved(X)) = 0
  | value(opal(X)) = 2*X
  | value(faceted(X)) = 4*X
  | value(akoya(X)) = 10*X;

fun cmp(A, B) = value(A) > value(B);

fun merge([],p,_) = p
  | merge(p,[],_) = p
  | merge(a::p1, b::p2, compare) = if compare(a,b) then a::merge(p1,b::p2,compare) else b::merge(a::p1,p2,comp

fun sort([], _) = []
  | sort([a], _) = [a]
  | sort(L,cmp) = let val l1 = len(L) val p1 = sort(take(L,l1 div 2),cmp) val p2 = sort(drop(L,l1 div 2),cmp)
                  in merge(p1,p2,cmp) end;

fun sum([]) = 0
  | sum(a::L) = value(a) + sum(L);

fun shiny(X,k) = sum(take(sort(X,cmp),k));
```

A nice variant would be to use function intsort that sorts lists of integers and then

fun shiny(X,k) = foldl op+ 0 (take(intsort(map value X),k));

Third idea: we can obtain the result for a list $a :: L$ and k pearls left by either taking the a element into account: $value(a) + shiny(L, k-1)$ or dropping a: $shiny(L, k)$. The following (shorter) implementation uses the idea:

datatype pearl=carved **of** int | opal **of** int | faceted **of** int | akoya **of** int;

fun value(carved(X)) = 0
  | value(opal(X)) = 2∗X
  | value(faceted(X)) = 4∗X
  | value(akoya(X)) = 10∗X;

fun max(a, b) = **if** a>b **then** a **else** b;

fun shiny([], _) = 0
  | shiny(_, 0) = 0
  | shiny(a::L, k) = max(value(a)+shiny(L,k−1), shiny(L,k));
