Name: Matriculation Number:

Final Exam
General CS II (320201)

May 23., 2012

You have two hours(sharp) for the test;
Write the solutions to the sheet.

The estimated time for solving this exam is 115 minutes, leaving you 5 minutes for revising your exam.

You can reach 115 points if you solve all problems. You will only need 100 points for a perfect score, i.e. 15 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.	1.1	1.2	2.1	2.2	2.3	3.1	4.1	5.1	6.1	Sum	grade
total	15	5	12	10	10	18	18	15	12	115	
reached											

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

- "Prove or refute" means: If you think that the statement is correct, give a formal proof. If not, give a counter-example that makes it fail.
- 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 Circuits and Positional Number Systems

Problem 1.1 (13-bit TCN) 5pt 5min

1. What is the largest number (in decimal) that can be written using 13-bit two's complement number notation?
2. Convert the obtained decimal number into hexadecimal.
3. What is the smallest number (in decimal) that can be written using 13-bit two's complement number notation?

Solution:

1. The largest number is $\langle\!\langle 0111111111111 \rangle\!\rangle^{2s} = 2^{12} - 1 = 4095_{10}$.
   (2 points)
2. In hexadecimal: $4095_{10} = \langle\!\langle 1111\,1111\,1111 \rangle\!\rangle = FFF_{16}$.
   (2 points)
3. The smallest number is $\langle\!\langle 1000000000000 \rangle\!\rangle^{2s} = -2^{12} = -4096_{10}$.
   (1 point)

Problem 1.2 (Designing a Robot Logic Circuit) 15pt 15min

You are designing a robot to move toward a light source. Three photo-sensors SL, SC, and SR are mounted at the front of the robot pointing 45° to the left, straight ahead, and 45° to the right, respectively. Two wheels, WL and WR, are powered depending on the output of the sensors. If SL detects light and SR does not ($SL = SC = 1$, $SR = 0$ or $SL = 1$, $SC = SR = 0$), the robot is pointing too far to the right, and the right wheel WR must be powered up to turn the robot to the left. The opposite is necessary if SR receives light and SL does not ($SR = 1$, $SL = SC = 0$ or $SR = SC = 1$, $SL = 0$). If only the forward-pointing sensor SC is lit, or SL and SR are lit at the same time, then both wheels WL and WR should be powered to push the robot forward. Finally, if none of the sensors detect light, the wheels should not be powered.

If we treat the sensors as having binary outputs, i.e., either on (1) or off (0), and the powered wheels as being on or off, a simple logic circuit can be used to actuate the wheels under each sensor condition.

1. Write down the truth table for the circuit and use any method to simplify the resulting logic expression.

2. Create such a logic circuit using only AND, OR, NOT gates, and using the least number of these. Explain what signals you have used for the unspecied condition.

3. Implement the same circuit (same logic expression) using the least number of NAND gates

---

Note: For the last part, you do not have to prove that the number of gates you used is minimal. However the number of gates should be smaller than in the circuit obtained just by directly expressing each OR/AND/NOT gate with NAND gates.

Solution:

2 Machine Languages

12pt
12min

Problem 2.1 (ASM max)

Write an ASM program which replaces the first element of a non-empty list of integers by the maximum value in the list. $P(0)$ contains the number of elements in the list. It is guarantied that the list contains at least one element. $P(1)$ through $P(P(0))$ contain the elements of the list. After you program terminates, $P(1)$ must contain the value of the maximum element in the list, $P(0)$ can contain any value you like and all the other cells must retain their initial values.

Note that you are only allowed to use labels in jump statements. You are very much encouraged to write comments and even a description of your approach, which can help the grader award partial points. Here is a list of ASM commands (only for reference).

load i	jump(=) i
store i	jump(>) i
loadi i	jump(<) i
addi i	jump($\geq$) i
subi i	jump($\leq$) i
add i	sub i
move S T	jump($\neq$) i
loadin i j	storein i j
jump i	stop 0

Hint: You may want to make use of the "storein" and "loadin" instructions which deal with relative addressing. After all, at the beginning $P(s+2) = P(P(0)+2)$.

Hint: There might just be a "hidden" induction somewhere which could make life easy for you.

Solution: This is an induction over the size s of the list. The recursion is simple: if $P(0) = 1$ then there is nothing to do, if $P(P(0)) > P(1)$, replace $P(0)$ by $P(P(0))$, decrement $P(0)$

Here is the code with comments

```
# get the size of the list
load 0
# check if we are at the base case
subi 1
jump(<=) 10
# store the (already computed) $s$−1 in register IN1 − it's a useful asset
move ACC IN1
# load the value of the last element in the list (note that we need an offset of
# 1 for IN1)
loadin 1 1
# save the value in case it turns out to be a current maximum
move ACC IN2
# compare to the current maximum from D(1)
sub 1
```

```
jump(<=) 3
# if the last element was indeed greater, update the current maximum D(1)
move IN2 ACC
store 1
# move $s-1$ into the accumulator for the next iteration
move IN1 ACC
# no need to update D(0). We can simply use ACC and IN1, so we jump to where the
# decrementing takes place at the beginning
jump -10
stop 0
```

The code without comments:

```
load 0
subi 1
jump(<=) 10
move ACC IN1
loadin 1 1
move ACC IN2
sub 1
jump(<=) 3
move IN2 ACC
store 1
move IN1 ACC
jump -10
stop 0
```

Problem 2.2 (Draw the Call Stack) 10pt 10min

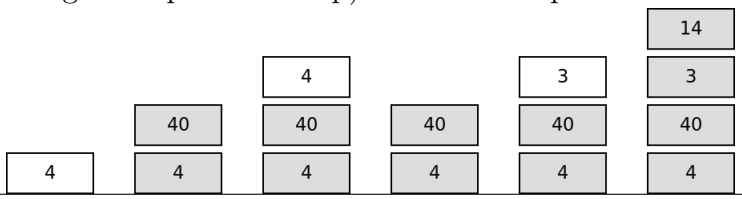
You are given the following two mutually recursive functions:

$$odd(x) = \begin{cases} 0 & \text{if } x = 0 \\ even(x-1) & \text{else} \end{cases} \qquad \text{and} \qquad even(x) = \begin{cases} 1 & \text{if } x = 0 \\ odd(x-1) & \text{else} \end{cases}$$

The corresponding $\mathcal{L}(\texttt{VMP})$ code is written below. The line numbers denote the number of the instruction that is first on that row.

```
0  proc 1 18          18  proc 1 18
3  arg 1              21  arg 1
5  cjp 10             23  cjp 10
7  con 1 arg 1 sub    25  con 1 arg 1 sub    36  con 4
12 call 18            30  call 0             38  call 0
14 return             32  return             40  halt
15 con 0              33  con 1
17 return             35  return
```

In lines 36-38 we want to evaluate $odd(4)$. Your task is to draw the VM stack at each computation step done by the virtual machine (you may treat every line in the above code as a single computation step). The first steps are shown below:



Solution:

Problem 2.3 (Fibonacci checker) 10 pt 10 min

Design a $\mathcal{L}(\texttt{VMP})$ program that, given a positive integer n in $S(0)$, will write 1 or 0 in $S(1)$, depending on whether n is a Fibonacci number or not, respectively. Recall that the Fibonacci numbers are defined by $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$, for $n \geq 2$.

Solution:

```
con n
con 0
con 0 con 1
peek 2 peek 0 leq
cjp 19
peek 2 peek 0 sub
cjp 7
con 0 poke 1 halt
con 1 poke 1 halt
peek 2 peek 3 add
peek 3 poke 2
poke 3
jp −35
```

3 Turing Machines

18pt
18min

Problem 3.1 (Distinct Strings)

Design a Turing Machine that decides whether all **pairs of consecutive** strings in a list of binary strings are distinct, and appends "#1" to the right of the input if so. If any of the strings are equivalent, just halt.
All strings are of the same length and are separated by #'s.

Input (x_i is a binary string):	$\#x_1\#x_2\#\ldots\#x_k$
Output if strings are distinct:	$\#x_1\#x_2\#\ldots\#x_k\#1$

Assume the head is positioned at the first #. You may use additional symbols in your alphabet.
Describe the idea behind your algorithm and provide a transition (action) table.

Solution:

4 Internet

18pt
18min

Problem 4.1 (Internet Overview)

Please explain with as much detail as you can:

1. Through what layers does a packet go when traveling from one host to another? Start with application layer from host A and finish with application layer in host B. Provide a graph and give a detailed explanation of what happens in each layer.

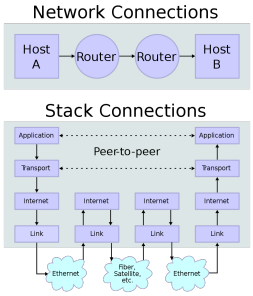
2. What are TCP and UDP? What is the difference between the two? List at least three differences.

3. What is an URI? Explain what is the syntax of an URI. You can use the following example to help you in your explanation:
   `http://thelast.net:80/problem?course=gencs2#solution`

4. What is TLS? Specify the steps which are executed by a client and a server in a TLS handshake.

Solution:

1. The packet goes down from the application layer on A to the link layer. Then on each routing hop it goes up to the Internet link, where based on the destination IP address it is decided where to route it. At the destination B it goes up from the link layer to the application layer. Graph can be seen on slide 340 in the notes:



2. TCP Transmission Control Protocol
   UDP User Datagram Protocol

   TCP is manly used in the Internet layer, UDP in the transport layer.
   TCP is used for commutation. UDP for multicasting and broadcasting.
   TCP is connection oriented, UDP is connectionless.
   TCP provides reliable communication, UDP doesn't and may require retransmission.

3. A uniform resource identifier (URI) is a global identifiers of network- retrievable documents (web resources). URIs adhere a uniform syntax (grammar) .

`http://thelast.net:80/problem?course=gencs2#solution`
scheme authority path query fragment

4. Transport layer security (TLS) is a cryptographic protocol that encrypts the segments of network connections at the transport layer, using asymmetric cryptography for key exchange, symmetric encryption for privacy, and message authentication codes for message integrity.

   (a) Client presents a list of supported encryption methods

   (b) Server picks the strongest and tells client (C/S agree on method)

   (c) Server sends back its public key certificate (name and public key)

   (d) Client confirms certificate with CA (authenticates Server if successful)

   (e) Client picks a random number, encrypts that (with servers public key) and sends it to server.

   (f) Only server can decrypt it (using its private key)

   (g) Now they both have a shared secret (the random number)

   (h) From the random number, both parties generate key material

---

5 Problem Solving and Search

15pt
15min

Problem 5.1 (The noodle master)

It is The Day of The Final in GenCS town! You, as the Noodle Master, have to prepare a marvelous meal for all the tired students that will take the exam. However, the Gods don't favor you, and you lack all the ingredients that you need for your recipes.

The map of GenCS is given as a table, where each cell has associated the cost of visiting it. In order to acquire all the ingredients, you must **visit all the corners** of the table, where the farmer markets are placed! From any given cell, you can visit any neighboring cell by going up, down, right or left. You cannot leave the town, and you start the search from the central cell of the table (coordinates $\langle 3, 3 \rangle$).

1	10	10	10	1
4	3	2	5	3
10	2	1	3	2
9	9	2	8	3
1	2	2	2	1

You are requested to:

1. Give a problem formulation that will describe the statement and that will respect all the restrictions. You can specify the transition function only partially (at least 5 transitions).
2. Give an admissible heuristic function, applicable to your problem formulation. Explain why the heuristic is admissible.
3. Apply the A^* algorithm by expanding enough nodes to reach one corner. Show the fringe and how nodes are added to it at each step.

Solution:

1. As soon as we identify each cell by a letter (from A to Y), we can define a state to be a pair $\langle c, m \rangle$, where c is a letter of a cell and m is a mask, where bit i is 0 if the assoicated corner was not visited and 1 if it was visited. Therefore, the initial state is $\langle M, 0 \rangle$ and the goal states are $\langle A, 15 \rangle$, $\langle E, 15 \rangle$, $\langle U, 15 \rangle$, and $\langle Y, 15 \rangle$. Transitions: $\langle M, 0 \rangle \rightarrow \langle H, 0 \rangle$, weighted by the value of the destination cell (here, 2).
2. In a grid, the straight line distance is always an admissible function. For simplicity, we can use the floor of the straight line distance.
3. A^* will be applied according to the rules.

---

6 Programming in Prolog

12pt
12min

Problem 6.1 (The Unzip Function)

For the last problem of your final exam in GenCS you have to implement another function from the famous SML grand tutorial: the `unzip` function.

So now you are given a list containing lists of two numbers, and at most one list containing a single number. Your task is to unzip this list into two lists, one of which contains all the first elements from these lists, and the other one - all the second elements. Create a `ProLog` predicate with 3 arguments: the first would be the list you want to unzip, and the other two would be the two lists containing the elements. For instance:

```
?− unzip([[1,2],[3,4],[5,6],[7,8]],A,B).
A = [1, 3, 5, 7],
B = [2, 4, 6, 8].

?− unzip([[1,2],[3]],A,B).
A = [1, 3],
B = [2].
```

Feel free to implement any helper functions.