Name: Matriculation Number:

Final Exam
General CS I (320101)

May 21, 2010

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

The estimated time for solving this exam is 112 minutes, leaving you 8 minutes for revising your exam.

You can reach 87 points if you solve all problems. You will only need 80 points for a perfect score, i.e. 7 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	3.2	4.1	5.1	Sum	grade
total	4	10	10	10	10	12	8	15	8	87	
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 Circuits and binary number systems

4pt
8min

Problem 1.1 (Negative Numbers)

Give the definitions (formulas) of the following representations of negative numbers:

1. $(\langle\!\langle a_n, \ldots, a_0 \rangle\!\rangle^-)$
2. $\langle\!\langle a_n, \ldots, a_0 \rangle\!\rangle_n^{2s}$

What is the shortcoming of the first approach, and what are the advantages of the second one?

Solution:

1. $\langle\!\langle a_{n-1}, \ldots, a_0 \rangle\!\rangle$ if $a_n = 0$ $-\langle\!\langle a_{n-1}, \ldots, a_0 \rangle\!\rangle$ if $a_n = 1$
2. $-a_n \cdot 2n + \langle\!\langle a_{n-1}, \ldots, a_0 \rangle\!\rangle$

In the first approach zero is represented twice (10000 and 00000). Second approach represents zero in an unique way. Can use adders without modifications. (any other advantages?)

Problem 1.2 (2-bit Address Decoder) 10 pt
15 min

Design a two-bit address decoder using only XOR and AND gates.

2 Machines

10pt

Problem 2.1 (Palindrome check) 15min

Write an assembler program that checks whether a word is a palindrome number (i.e. a numbers that read the same from left to right and from right to left, e.g. "101" or "0110".)

Assume that $P(0)$ contains n where n is the number of letters in a word from $L = \{0,1\}^*$. The word itself starts from $P(2)$. Write 1 to $P(1)$ if the word is a palindrome and 0 otherwise.

Solution:

label	instruction	comment
	LOADI 0	beginning
	MOVE ACC $IN2$	
	LOAD 0	end
	MOVE ACC $IN1$	
		check if beginning > end
$\langle loop \rangle$	MOVE $IN2$ ACC	
	STORE 1	
	MOVE $IN1$ ACC	
	SUB 1	
	JUMP$_=$ $< \langle end1 \rangle$	
	LOADIN 2 2	
	STORE 1	
	LOADIN 1 1	
	SUB 1	
	JUMP$_=$ $! = \langle end_0 \rangle$	
	MOVE $IN2$ ACC	
	ADDI 1	
	MOVE ACC $IN2$	
	MOVE $IN1$ ACC	
	SUBI 1	
	MOVE ACC $IN1$	
	JUMP $\langle loop \rangle$	
$\langle end_1 \rangle$	LOADI 1	
	STORE 1	
	JUMP 3	
$\langle end_0 \rangle$	LOADI 0	
	STORE 1	
	STOP 0	

Problem 2.2 (Integral logarithm) 10pt 12min

Write down a static $\mathcal{L}$(VM) procedure that computes the integral log of a number to a given base. The pseudocode of such a function is given below :

```
ilog(n,b) {
        var x = 1,i=0 ;
        while(x<=n){
                x=x*b;
                i = i+1;
        }
        return i;
}
```

You can assume that the input is always valid $(n, b > 0)$.Uncommented code will not be graded.

Hint: Think of modifying the pseudocode to create an alternate function which avoids the use of local variables given in the pseudocode (Remember that peek and poke are not allowed in $\mathcal{L}$(VM) static procedure).

Solution: We first create a function iloghelp (n,b,x,i) which contains the two local variables in it as arguments

//iloghelp(n,b,x,i)
// if ($x <= n$) then
// iloghelp(n,b,x*b,i+1)
// else return i

//ilog(n,b) = iloghelp(n,b,1,0)

`proc` 4 27	iloghelp(n,b,x,i)
`arg` 1 `arg` 3 `leq` `cjp` 7	
`arg` 4 `con` 1 `add`	
`arg` 3 `arg` 2 `mul`	
`arg` 2 `arg` 1 `call` 0 `return`	
`arg` 4 `return`	
`proc` 2 13	ilog(n,b)
`con` 0 `con` 1 `arg` 2 `arg` 1	
`call` 0	

Problem 2.3 (Zeros) 10pt 12min

Design a Turing machine that takes an input of the form $\#w_1\#w_2\#$, where $w_1, w_2 \in \{0,1\}^*$ and terminates in the state:

- 'yes', if the number of zeros in w_1 equals the number of ones in w_2
- 'no', otherwise.

w_1 and w_2 can have any lengths, including 0. The initial position of the head is on the first character in w_1. All the other tape positions in the input are assumed to be written with hashes. You can add new symbols to the tape alphabet if you wish.

It is enough to explain precisely what *each* state does, taking care of all the possible cases. *No transition table required.*

Example:

- input: $\#\underbrace{0010}_{3 \text{ zeros}}\#\underbrace{1101000}_{3 \text{ ones}}\#\#\ldots$ TM terminates in state 'yes'
- input: $\#\underbrace{11}_{0 \text{ zeros}}\#\underbrace{0000}_{0 \text{ ones}}\#\#\ldots$ TM terminates in state 'yes'
- input: $\#\underbrace{1001}_{2 \text{ zeros}}\#\underbrace{100}_{1 \text{ ones}}\#\#\ldots$ TM terminates in state 'no'

Solution: Systematically write *s over pairs of 0 from w_1 and 1 from w_2. If at the end there are still 0s in w_1 or 1s in w_2, the TM halts with 'no', otherwise with 'yes'.

- State 1 skips over w_1 and jumps to state 2 on the last position before the middle #.
- State 2 goes left until it finds '0', writes * and jumps to state 3. If it doesn't find any '0' (all '0's have a pair), it jumps to state 6.
- State 3 is a transit state that goes right until it finds # and switches to state 4.
- State 4 goes right until it finds '1', writes * and jumps to state 5. If it doesn't find any, the TM halts with 'no'.
- State 5 is a transit state that goes left until it finds # and switches to state 2.
- State 6 is a transit state that goes right until it finds # and switches to state 7.
- State 7 is used when pairs have been found for all '0's from w_1, and checks whether there is any '1' left without a pair. If it finds one, the TM halts with 'no', otherwise with 'yes'.

```
1, #, 2, #, < 1, _ , 1, _, >
2, 1, 2, 1, < 2, *, 2, *, < 2, 0, 3, *, > 2, #, 6, #, >
3, #, 4, #, > 3, _, 3, _, >
4, 0, 4, 0, > 4, *, 4, *, > 4, 1, 5, *, < 4, #, no
5, #, 2, #, < 5, _, 5, _, <
6, #, 7, #, > 6, _, 6, _, >
7, 0, 7, 0, > 7, 1, no 7, *, 7, *, > 7, #, yes
```

Note: _ matches against any (remaining) character, and it implies that the same character is written to the tape.

3 Problem Solving and Search

12pt
15min

Problem 3.1 (Robots)

You are designing a new software package to coordinate a pair of robots transporting items around a building. Each robot has a transmitter used to communicate with the other robot. When the two robots are too far apart, they lose the signal and stop functioning. You have to plan a path such that both robots get to their respective destinations in minimum number of moves, while keeping close enough to each other.

Hint: The building plan can be given as a graph, where every node denotes a particular position in the building (discretized of course). Consider all edges bidirectional and having cost 1, for simplicity.

Your tasks are the following:

1. Formulate the problem precisely. Please spend some time on considering this as it has to be done rigurously and it shapes the way in which the following tasks will be solved. Also consider that you have to clearly describe how to construct the states for your state space and the successor function in your state space.

2. Suppose you want to use $A*$ but need to decide on a heuristic function h. What heuristic would you use? Describe the best possible heuristic you can think of, and argue why it is good.

Solution:

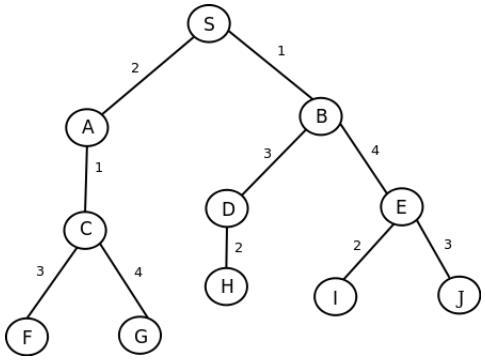
1. For robot A you are given a start and destination node (a, b); similarly, for robot B you are given a start and destination node (c, d). The two initial nodes, a and c, are at most t edges apart. The two destination nodes, b and d, are at most t edges apart. We require that robot A and robot B at any point in time must be at most t edges apart from each other. At any particular point in time, only one robot is allowed to move across an edge. Use two-tuples (x, y) to denote the state where x is the node robot A is in and y is the node robot B is in, of the building graph. State (x, y) and (y, x) are different unless $x = y$. A state is valid if and only if the shortest distance between x and y is at most t edges. The shortest distance between x and y is the number of edges they are apart. Hence, the shortest distance between x and x is 0. By this notation, if there are n nodes in the building graph, well have at most $2n$ states in the state space. The start state is hence (a, c) and the goal state will be (b, d). A greedy way to construct the state space would be to run a depth-rst-search with depth bound t for every node n. For each child c of node n, create a state (n, c). The depth bound will ensure that all created states are valid. The successor function would be constructed by enumerating the neighbors of both of x and y. A valid move is that either A or B moves across one edge and the new locations of A and B are still less than or equal to t edges apart. $(x, y) - > (xnew, y)$ such that $D(xnew, y) <= tand(x, y) - > (x, ynew)$ such that $D(x, ynew) <= t$ where $D(x, y)$ is the number of edges of the shortest path between x and y.

2. Dene: n is a state (x, y). Let $h(n) = D(x, b) + D(y, d)$ $h(n)$ is good because it is admissible. $h(n)$ accommodates the shortest distance from each robot from its goal. Without the 'less

than t edges apart' constraint, the actual cost for robot A to move from x to b will be larger than $D(x, b)$, that is, $D(x, b)$ will not overestimate the actual cost. Same for robot B. Moreover, with the 'less than t edges apart' constraint, the total cost for robot A and B to their respective destinations will be larger than that without the constraint. Hence, $h(n)$ will not overestimate the actual cost. Also, the smaller $h(n)$ is, the closer the robots to the goal, hence the better. It gives us a way to evaluate how close the robots are to their goals.

Problem 3.2 (Search in a graph) 8pt 8min

Look at the following graph and complete the tables bellow. S is the start state and G is the goal state, the step costs are given as labels on the edges.



In the table below, enter the labels of the nodes in the order they are visited by the respective search strategy. Remember that the search ends once the goal state is visited. If two nodes have equal chances to be visited, take the leftmost one first.

Search method	Sequence of nodes
BFS	
DFS	
Uniform cost	
Iterative deepening (step size 2)	

In the table below, complete the table with **Yes** or **No** for the respective search strategies and properties.

Property	BFS	DFS	Uniform Cost	Iterative Deepening
Optimal				
Complete				

Solution:

Search method	Sequence of nodes
BFS	S A B C D E F G
DFS	S A C F G
Uniform cost	S B A C D E F H G
Iterative deepening (step size 2)	S A C B D E S A C F G

Property	BFS	DFS	Uniform Cost	Iterative Deepening
Optimal	N	N	Y	N
Complete	Y	Y	Y	Y

---

4 Prolog

15pt
15min

Problem 4.1 (Ini Mini Miny Moe)

You all probably used some sort of counting rhyme when you were small, in order to decide who takes a certain role in your childhood games. The way it usually works is that for each word in the counting rhyme, you count one person, while everyone stays in a circle. When the rhyme is over, the last person counted is out and you continue the same procedure starting from the next person. When there is only one person left, then that person is 'it'.

Write a `ProLog` predicate `count(N, List, It)` that given N — the number of words in the rhyme and *List* — a list of names, decides who is 'It', the last person remaining. For instance

```
?- count(9,[ana, mihai, stefan, gloria],It).
It = stefan ;
false.
```

Solution:

```
#!/usr/bin/swipl -s

count(N, List, Sol):-drop1(N, 1, [], List, Sol).

notbasecase(Prev, Next):-append(Prev, Next, All),
                  not(length(All, 1)).

drop1(_, _, [], [X], X).
drop1(N, N, Prev, [X|Next], Sol):-notbasecase(Prev, [X|Next]),
                           drop1(N, 1, Prev, Next, Sol).
drop1(N, Current, List, [], Sol):-drop1(N, Current, [], List, Sol).
drop1(N, Current, Prev, [X|Next], Sol):-notbasecase(Prev, [X|Next]),
                                 not(N=Current),
                                 NewCurrent is Current+1,
                                 append(Prev, [X], NewPrev),
                                        drop1(N, NewCurrent, NewPrev, Next, Sol).
```

---

5 Internet

8pt

Problem 5.1 (Transfer time) 12min

While thinking about the final, 2 of your friendly GenCS TAs were sending drafts of the problems to each other over the Jacobs network. Given that:

- the Maximum Transmission Unit (**MTU**) on the network is $1500 bytes$ (in other words packets sent can not exceed this size)
- the total size of the headers in each packet is $54 bytes$ (leaving the rest to the actual data)
- the Round-Trip Time (**RTT**) - time for a packet to make a round-trip between a host and a destination - $10ms$

calculate how much time it will take to receive a file of size $100kb$ in each of the following scenarios. Consider that no network errors (bad packets, lost packets, etc.) occurred unless stated otherwise:

1. We want to make sure that the data has been transferred successfully, therefore after sending a packet of data, we are waiting for an acknowledgment (**ACK**) from the destination that the packet has been received. The time between receiving a packet and sending ACK and the time between receiving ACK and sending the next packet can be neglected.
2. Same as in 1 however this time 10 packets got lost on the way to the destination and therefore have to be retransmitted. Sender retransmits a packet if no ACK was received after $15ms$.