Name: Matriculation Number:

General CS II (320201) Final Exam
May 23. 2006

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

The estimated time for solving this exam is 23 minutes, leaving you 97 minutes for revising your exam.

You can reach 16 points if you solve all problems. You will only need 70 points for a perfect score, i.e. -54 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	Sum	grade
total	4	12	16	
reached				

Good luck to all students who take this test

1 Computational Logic

Problem 1.1 (Basics of Resolution) 4pt 8min

What are the principal steps when you try to prove the validity of a propositional formula by means of resolution calculus? In case you succeed deriving the empty clause, why does this mean you have found a proof for the validity of the initial formula?

1.1 Virtual Machines

Problem 1.2 **(Binary Conversion in $\mathcal{L}$(`VM`))** 12pt 15min

Write a $\mathcal{L}$(`VM`) program that converts a binary natural number into a decimal natural number. Suppose that n, the number of digits, is stored in `stack[2]` and n numbers 0 or 1 above it follow, where the top of stack is the least significant bit. `stack[0]` and `stack[1]` are available for your use. Your program should leave only the converted number on the stack (in `stack[0]`). You are allowed to use labels for (conditional) jumps.

For instance an initial stack $\begin{array}{|c|}\hline 1\\\hline 0\\\hline 1\\\hline 3\\\hline ?\\\hline ?\\\hline\end{array}$ should give the result stack $\boxed{5}$.

Solution: `con` (0)
`poke` (0) ; init. result to 0
`con` (1)
`poke` (1) ; init. 2^i to 1
`peek` (1)
`mul` ; multiply with 2^i
`peek` (0)
`add` ; add to result
`poke` (0)
`peek` (1) ; update multiplier
`con` (2)
`mul`
`poke` (1) `con` (1) 1 ; update digit counter
`peek` (2)
`sub`
`poke` (2)
`peek` (2) ; if counter = 0 go out
`cjp` 4 `jp` -26 ; else again
`poke` (1) 1 ; clean stack and stop
`add`
`halt`

1.2 Combinational Circuits

Problem 1.3 (Shift and Duplication on PNS) 8pt 10min

Consider for this problem the signed bit number system and the two's complement number system. Given a binary string $b = a_n \ldots a_0$. We define

1. the duplication function *dupl* that duplicates the leading bit; i.e. it maps the $n+1$-bit number $a_n \ldots a_0$ to the $n+2$-bit number $a_n a_n \ldots a_0$ and
2. the shift function $shift$ that maps the $n+1$-bit number $a_n \ldots a_0$ to the $n+2$-bit number $a_n \ldots a_0 0$

Prove or refute the following two statements

- The $shift$ function has the same effect in both number systems; i.e. for any integer z:
$$(\langle\!\langle shift(B(z)) \rangle\!\rangle^{-}) = \langle\!\langle shift(B_n^{2s}(z)) \rangle\!\rangle_{n+1}^{2s}$$
- The *dupl* function has the same effect in both number systems; i.e. for any integer z:
$$(\langle\!\langle dupl(B(z)) \rangle\!\rangle^{-}) = \langle\!\langle dupl(B_n^{2s}(z)) \rangle\!\rangle_{n+1}^{2s}$$

Solution:

- $(\langle\!\langle dupl(B(z)) \rangle\!\rangle^{-}) = z - 2^{n+1}$ if $z < 0$ else z.
- $(\langle\!\langle shift(B(z)) \rangle\!\rangle^{-}) = 2 * z$
- $\langle\!\langle dupl(B_n^{2s}(z)) \rangle\!\rangle_{n+1}^{2s} = z$
- $\langle\!\langle shift(B_n^{2s}(z)) \rangle\!\rangle_{n+1}^{2s} = 2 * z$

Proof for the last equality:

$$shift(-a_n * 2^n + \sum_{k=0}^{n-1} a_k * 2^k) = -a_n * 2^{n+1} + \sum_{k=0}^{n-1} a_k * 2^{k+1} = 2 * (-a_n * 2^n + \sum_{k=0}^{n-1} a_k * 2^k)$$

Problem 1.4 (Carry Chain and Conditional Sum Adder) 10pt 15min

Determine depth and cost of a 4-bit Carry Chain Adder, a 4-bit Conditional Sum Adder, and a combination of both where two 2-bit Carry Chain Adders are connected with by one Mux.

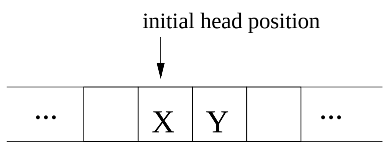
Which adder has lowest cost and depth respectively?

Hint: You don't need to draw the adders. On the other hand don't supply just the result numbers of your cost calculations, since the revisor can differentiate between fundamental and oversights only if he or she can reconstruct the calculation to some extent. Without any comments the wrong result leads to zero points though just a minor mistake was the actual reason.

1.3 Turing Machines

Problem 1.5 (Boolean Equivalence) 11pt 20min

Consider a tape arbitrarily filled with ones and zeros and the head initially positioned over some cell "X" as depicted below



Define a transition table for an always terminating Turing machine TM that computes the boolean equivalence of "X" and "Y": Upon halting, your TM should return the value 1 in cell "X" if the values of the cells "X" and "Y" were initially equal and otherwise 0.

Try to use as few states as possible. The number of points you can obtain for this exercise is $\max(0, 14 - x)$, where x is the number of states of your working TM.

Hint: You only need to consider the two cells "X" and "Y". It does not matter where the head stays when the TM terminates.

Note:

1. Admissible moves are *left*, *right*, and *none* with the obvious meaning.
2. You are free to overwrite the initial value of "Y" and to introduce additional symbols in the alphabet, if you need it for your solution.

Solution: There are lots of possible solutions.

The following four-state solution (including the final state) by Dmakreshanski Pesikan does not use any additional alphabet symbols:

Old	Read	Write	New	Move
s_1	0	0	s_2	right
s_1	1	1	s_2	right
s_2	0	0	s_3	left
s_3	1	0	s_4	left
s_3	0	1	s_4	none

Tanmay Pradhan presented a solution that uses additional symbols and only needs *two states*. This is supposed to be optimal.

Old	Read	Write	New	Move
s_1	0	W	s_2	right
s_1	1	Y	s_2	right
s_2	0	L	s_1	left
s_2	1	L	s_2	left
s_1	W	1	s_1	right
s_1	Y	0	s_1	right
s_2	W	0	s_1	right
s_2	Y	1	s_1	right

If we require that the head must stop on "X" – but we don't , as Darko pointed out! – , it gets more complicated. The following solution by Christoph Lange is quite straight-forward, but not optimal:

Old	Read	Write	New	Move
a	0	0	b	right
a	1	1	b	right
b	0	0	c_0	left
b	1	1	c_1	left
c_0	0	1	$\perp$	none
c_0	1	0	$\perp$	none
c_1	0	0	$\perp$	none
c_1	1	1	$\perp$	none

Andrei Aiordachioaie supposed this four-state solution:

Old	Read	Write	New	Move
s_0	1	X	right	s_x
s_0	0	Y	right	s_y
s_x	1	1	left	s_x
s_x	X	1	none	$\perp$
s_x	0	(anything)	left	s_y
s_x	Y	0	none	$\perp$
s_y	0	0	L	s_y
s_y	Y	1	none	$\perp$
s_y	1	(anything)	left	s_x
s_y	X	0	none	$\perp$

1.4 Problem Solving and Search

Problem 1.6 (Monotone heuristics) 12pt 15min

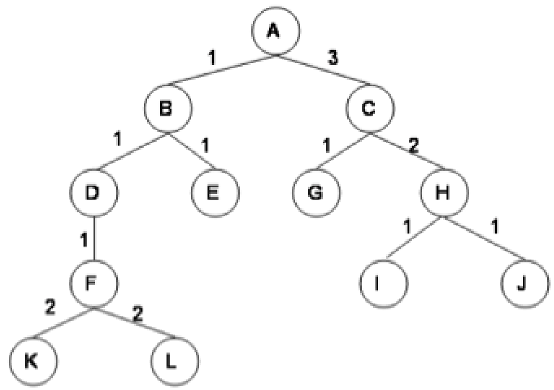
Let $c(n, a, n')$ be the cost for a step from node n to a successor node n' for an action a. A heuristic h is called *monotone* if $h(n) \leq h(n') + c(n, a, n')$. Prove or refute that if a heuristic is monotone, it must be admissible. Construct a search problem and a heuristic that is admissible but not monotone. Note: For the goal node g it holds $h(g) = 0$. Moreover we require that the goal must be reachable and that $h(n) \geq 0$.

Solution: For the heuristic h to be admissible we have to show that $h(x)$ is less or equal the minimum coast to a goal state.

Let n_1 any node different from the goal node g. Suppose $< n_1, n_2, \ldots, n_p, g >$ is the minimum cost path from n_1 to g. Its cost is $C = c(n_1, a_1, n_2) + c(n_2, a_2, n_3) \ldots + c(n_p, a_p, g)$. Using $h(n) - h(n') \leq c(n, a, n')$ we get $C \geq h(n_1) - h(n_2) + h(n_2) - h(n_1) + \ldots + h(n_p) - h(g) = h(n_1) - h(g) = h(n_1)$. Hence we have proven that $h(n_1)$ is admissible.

We consider the minimum distance search problem with three cities A, B, G where G is the goal city and the distances are $dist(A, B) = 2$ and $dist(B, G) = 100$. The heuristic $h(A) = 6, h(B) = 3, h(G) = 0$ is admissble since $h(A) < dist(A, B) + dist(B, G)$. But is is not monotone since $h(A) > h(B) + dist(A, B)$.

Problem 1.7 (Search Strategy Comparison on Tree Search)
Consider the tree shown below. The numbers on the arcs are the arc lengths.



Assume that the nodes are expanded in alphabetical order when no other order is specified by the search, and that the goal is state G. No visited or expanded lists are used. What order would the states be expanded by each type of search? Stop when you expand G. Write only the sequence of states expanded by each search.

Search Type	Sequence of States
Breadth First	
Depth First	
Iterative Deepening (step size 1)	
Uniform Cost	

1.5 Prolog

Problem 1.8 (Greatest Common Divisor) 10pt 15min

Write a `ProLog` program with a ternary predicate `gcd`, such that `gcd(A,B,D)` returns "Yes" if `D` is the greatest common divisor of `A` and `B`. Otherwise it should either return "No" or it should never terminate.

You can make use of any of the following built-in predicates: `<`, `>`, `=<`,`>=`, and `=`, as well as the basic arithmetic operations. But you have to define `mod` on your own if you need it.

Note: The Euclidean algorithm for the look like:

```
fun gcd(x, y) = if y = 0 then x else gcd(y, x mod y);
```

Solution:

```
mod(A,B,A):-A<B.
mod(A,B,C):-A>=B,D is A-B,mod(D,B,C).
gcd(A,0,A).
gcd(A,B,D):-B>0,mod(A,B,C),gcd(B,C,D).
```