CPSC 201: Introduction to Computer Science
Carsten Schürmann
Date: April 29, 2003

Final Examination

Name: SAMPLE SOLUTION

Instructions

- This is a closed book, closed notes, closed computer examination.
- There are 20 pages including 5 worksheets.
- This examination consists of 6 questions worth 200 points. The point value of each question is given with the question.
- Read each question completely before attempting to solve any part.
- Write your answers legibly in the space provided on the examination sheet. If you use the back of a sheet or a worksheet, indicate *clearly* that you have done so on the front.
- The worksheets attached to the end of this examination are for your own use; they will not be used in grading unless stated otherwise.

Question	1	2	3	4	5	6	Total
Score	40	20	40	20	40	40	200
Maximum	40	20	40	20	40	40	200

1 Induction and Recursion [40 points]

We define binary trees by

```
datatype 'a tree = Empty | Node of 'a * 'a tree * 'a tree
```

We can uniquely identify a subtree of a binary tree by specifying a *path* from the root to the subtree: At each step during the top-down traversal we are instructed that we have arrived (`H`), that we have to go left (`L`), or that we have to go right (`R`). We thus represent paths by

```
datatype path = H | L of path | R of path
```

We call a path *p valid in a tree t* if the path designates some subtree of t. For example, the root of any tree has address `H`, the left subtree has address `L(H)`, the right subtree has address `R(H)`. The addresses `L(H)`, `R(H)` are not valid in the empty tree `Empty`.

1.1 [10 points]

Write an ML function

```
val subtree : path * 'a tree -> 'a tree
```

such that `subtree` (p, t) returns the subtree of t at address P or raises the exception `Path` if the subtree does not exist.

1.2 [15 points]

Write an ML function

```
val paths : 'a tree -> path list
```

such that `paths` (t) returns a list containing precisely the valid paths in t.

1.3 [15 points]

We define

```
fun mirror H = H
  | mirror (L p) = R (mirror p)
  | mirror (R p) = L (mirror p)
fun reflect Empty = Empty
  | reflect (Node (d, left, right)) =
      Node (d, reflect (right), reflect (left))
```

Prove that `reflect (subtree (`p`, `t`))` $\equiv$ `subtree (mirror (`p`), reflect (`t`))` for any tree t, and path p valid in t. Here $e_1 \equiv e_2$ means that e_1 and e_2 evaluate to the same value.

2 Hardware [20 points]

In this question you are asked to develop a hardware circuit to compute one parity bit for 3 bit words. A parity bit is 1, if the three bit word contains an *odd* number of 1 bits and it is 0 if the three bit word contains an *even* number of 1 bits. For example, the parity bit for 000 would be 0.

Parity bits have played an important role in computer science because with their help it is possible to determine if a 3 bit word read from memory has been corrupted or not. Your solution should be directly implementable in hardware using only NAND gates.

2.1 [5 points]

Give a truthtable for the operation that computes a parity bit from a three bit word.

2.2 [10 points]

Give a Boolean expression to determine the parity bit using only NAND gates.

2.3 [5 points]

Give a circuit using gates that computes the parity bit.

3 Turing machines [40 points]

Write a Turing program that sorts a block of 0's and 1's into ascending order. The block of 0's and 1's is represented on the Turing tape. One possible starting configuration is

...		1	0	0	0	1	1	0	0		...
		⇑									

Note that that the block of digits is delimited by two blank symbols. The read/write head is positioned over the first field of the block, here the left most 1. The outcome of a successful run of your Turing program is depicted below.

...		0	0	0	0	0	1	1	1		...
										⇑	

Note that there are as many 0's and 1's on the tape after the Turing program has terminated as there were before. Your Turing machine should only use the three symbols "0", "1", and "   ". Do not introduce new symbols into the alphabet.

3.1 [10 points]

Describe each state of your Turing machine, informally, but carefully.

Answer:

- `State 1`: Scanning the leading block of 0's.
- `State 2`: Scanning the second block of 1's.
- `State 3`: 10 found. Exchange order. The 0 is to the right and already replaced by a 1.
- `State 4`: Scroll to the left most character of the current block.

3.2 [20 points]

Write out your Turing program. You can do this either in form of a table, a set of rules, or an automaton.

Answer:

```
val sort  = Program
  [Command ((State 1, Zero), SOME (Zero, State 1, Right)),
   Command ((State 1,  One), SOME (One, State 2, Right)),
   Command ((State 1,Blank), NONE),
   Command ((State 2, Zero), SOME (One, State 3,  Left)),
   Command ((State 2,  One), SOME (One, State 2, Right)),
   Command ((State 2,Blank), NONE),
   Command ((State 3, One),  SOME (Zero, State 4, Left)),
   Command ((State 4, Zero), SOME (Zero, State 4, Left)),
   Command ((State 4,  One), SOME (One, State 4, Left)),
   Command ((State 4,Blank), SOME (Blank, State 1, Right))]
```

3.3 [10 points]

Show an example run of your Turing machine using the following initial configuration.

...					1	1	0					...
					1 ⇑							

...					1	1	0					...
						2 ⇑						

...					1	1	0					...
							2 ⇑					

...					1	1	1					...
						3 ⇑						

...					1	0	1					...
					4 ⇑							

...					1	0	1					...
				4 ⇑								

...					1	0	1					...
					1 ⇑							

...					1	0	1					...
						2 ⇑						

...					1	0	1					...
					⇑							

...												...

...												...

...												...

...												...

...												...

...												...

4 Computability [20 points]

Consider the problem of defining a function `find` of type `(int -> bool) -> bool` which for an ML function p of type `int -> bool` (i.e., a predicate) returns true if and only if $p\ n$ evaluates to true for some $n \geq 0$. Show that it is *not* possible to define the function `find` in ML in such a way that

1. `find` p always returns either `true` or `false`, and
2. `find` p evaluates to `true` if and only if there is some $n \geq 0$ such that $p\ n$ evaluates to `true`.

Note, that the obvious sequential search testing $p\ n$ for $n = 0, 1, 2, ...$ doesn't work as an implementation of `find` because p may not terminate on m even though there is an $n > m$ such that $p\ n$ evaluates to `true`.

Hint: Show that if one indeed could write `find` then one could write a function `halt : (unit -> unit) -> bool`, which is a decision procedure for the halting problem. This means that `halt` f evaluates to `true` if f halts, and to `false` otherwise.

5 Interpreters [40 points]

In this question we consider the problem of parsing nested lists of integers, which are separated through spaces. As examples consider `(1 2 (5 6))`, or `(((1 2) (3 4)))` which should be accepted by the parser. `(1 (2 4)`, on the other hand, should be rejected, because the parentheses don't match.

Assume that a lexer returns values of type `token` defined by

```
datatype Token = INT of int | LPAREN | RPAREN
```

The parser is supposed to map a list of tokens into a value of type `Exp` and the remaining list of tokens.

```
datatype Exp
  = Int of int
  | List of Exp list
```

Consider two example runs of the parser on the examples from above.

1.
```
parse [LPAREN, INT 1, INT 2, LPAREN, INT 5, INT 6, RPAREN, RPAREN]
⇓ (List [Int 1,Int 2,List [Int 5,Int 6]],[])
```

2.
```
parse [LPAREN, LPAREN, LPAREN, INT 1, INT 2, RPAREN,
       LPAREN, INT 3, INT 4, RPAREN, RPAREN, RPAREN]
⇓ (List [List [List [Int 1,Int 2],List [Int 3,Int 4]]],[])
```

The following is the beginning of an implementation.

```
exception Error of string
fun parseExp (LPAREN :: s) k = parseList s nil  k
  | parseExp (INT (n) :: s) k = k s (Int (n))
  | parseExp nil k = raise Error "Incomplete expression"
  | parseExp (RPAREN :: s) k = raise Error "Unexpected right paren"
and parseList (RPAREN :: s) stack k = k s (List (List.rev (stack)))
  | parseList s stack k =
      ... gap₁ ...
fun parse s = parseExp s (... gap₂ ...)
```

5.1 [10 points]

Define a grammar for expressions.

Terminal symbols:

Non-terminal symbols:

Grammar rules:

Start symbol:

5.2 [5 points]

Give the type of `parse`.

5.3 [5 points]

Give the type of `parseExp`.

5.4 [5 points]

Give the type of `parseList`.

5.5 [10 points]

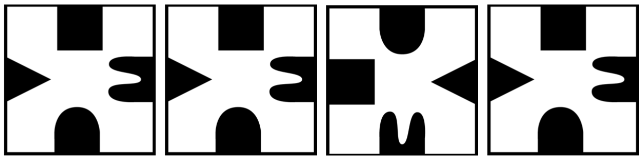
Fill in … gap_1 ….

5.6 [5 points]

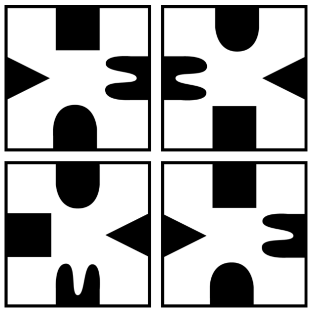
Fill in … gap_2 ….

6 Artificial Intelligence [40 points]

In this problem, you are asked to give a solution to a puzzle that involves a set of square playing cards. Four (not necessarily different) symbols are printed on each card, in such a way that each of the symbols touches one edge of the card.



A solution to the puzzle is an arrangement of the cards into a 2 by 2 grid, in such a way that any two symbol that share a common edge coincide. One possible solution of the puzzle is therefore as follows.



Now you are asked to devise an algorithm that attempts to construct a solution to this puzzle with 4 cards (which may of course be different from the example cards above). If the puzzle cannot be solved your algorithm should report failure as well.

6.1 [10 points]

Not taking any symmetries into account, in how many different ways can you arrange the cards on a 2 by 2 grid?

Answer:

$$4! * 4^4 = 384$$

6.2 [20 points]

Sketch the algorithm that solves the puzzle with 4 cards in pseudo code. You may be informal, but your description must be unambiguous. If you want to use ML code, that's fine, too.

Hint: Structure your solution into four parts: the representation of cards, the representation of the 2 by 2 grid, an algorithm that determines if the puzzle is solved, and the algorithm that tries to solve the puzzle.

Answer:

1.
```
datatype Symbol = Circle | Square | Triangle | Wave
datatype Card = Card of Symbol * Symbol * Symbol * Symbol
```

2.
```
datatype Board = Board of Card * Card * Card  * Card
```

3.
```
fun evaluate (Board (Card (A1, B1, C1, D1),
                     Card (A2, B2, C2, D2),
                     Card (A3, B3, C3, D3),
                     Card (A4, B4, C4, D4))) =
  C1=B2 andalso D2=A4 andalso B4=C3 andalso A3=D1
```

4.
```
fun generate [] [C1, C2, C3, C4] =
      evaluate (Board (C1, C2, C3, C4))
  | generate Cs A =
      pick (Cs, []) A
and pick ([], _) A = false
  | pick (C :: Cs, Cs') A =
      pick (Cs, C :: Cs') A
      orelse rotate (Cs @ Cs') (C, 4) A
and rotate Cs (C, 0) A = false
  | rotate Cs (C, n) A =
      generate Cs (C :: A)
      orelse rotate Cs (turn C, n-1) A
```

6.3 [10 points]

How would the runtime of the algorithm behave if we were to generalize it in a straightforward way to n^2 cards and not just 4? Express the runtime in big O notation. There is no need to give the generalized algorithm here, just the runtime, please. Justify your answer.

Answer:

In general, we have $n!4^n$ possible board configurations. If we generate and test each one of them, our algorithm will have a running time of $O(n!4^n)$. We notice by an easy mathematical argument.

Name: ______________________________

Worksheet

Name: ______________________________

Worksheet

Name: ______________________________

Worksheet

Name: ______________________________

Worksheet

Name: ______________________________________

Worksheet