Computer Science 52

Final Examination: Topics and Sample Problems

In examinations the foolish ask questions that the wise cannot answer. —Oscar Wilde, 1894

Topics

SML and Functional Programming
- Recursion (all kinds, including use of accumulators)
- Type signatures and type checking
- Functional style and pattern-matching
- Higher-order functions, like `map` and `fold`
- Anonymous functions
- Data types, including recursive ones
- Exceptions
- Correctness
- Lazy structures

Concepts
- Induction (all kinds)
- *O*-notation

Models of Computation
- Bits and propositional logic
- Circuits and gates
- Integers and words, unsigned and signed representation, bit-wise operations
- CS41B, including memory and addresses, subprograms, and stack frames
- Alphabets and languages
- Finite automata
- Turing machines

Applications
- `cs52int`
- Coding and RSA
- Exhaustive search

Sample Problems These problems are provided to give you an idea of the kinds of problems that will appear on the exam. They give you an opportunity to practice and to test your knowledge.

However, they are not complete, in that they do not cover every conceivable part of the course. Also, many of the exercises are longer or less precisely-worded than real exam problems. One problem is completely irrelevant.

1. A *queue* is a data structure in which insertions are done on one end of a list and deletions on the other. The acts of insertion and deletion are called "enqueue" and "dequeue," respectively.

a. Here is a naive implementation of a queue, using a SML list.

```
exception EmptyQueue;

fun enqueue elt queue = queue @ [elt];

fun dequeue nil           = raise EmptyQueue;
  | dequeue (first::rest) = (first, rest);
```

The idea for `dequeue` is that it returns a pair—the element that was removed and the resulting queue. Suppose that someone starts with an empty queue, carries out n `enqueue` operations and then n `dequeue` operations. What is the time complexity of that process? An answer in O-notation is acceptable.

b. A more efficient implementation of a queue is to represent the queue as a pair of lists. The `enqueue` operation puts a new element on the front of the first list. The `dequeue` operation acts on the first element of the second list. If second list is empty when a `dequeue` operation is attempted, the second list is replaced by the reversal of the first list. Write the functions `enqueue` and `dequeue` for this implementation.

c. For your implementation in part b, what is the time complexity of the process of inserting n elements into an empty queue and then removing them?

2. A *stack* is a data structure in which insertions and deletions are done on the same end of the list. The words "push" and "pop" are used for insertion

and deletion, respectively. Normally, the pop operation returns the element that has just been removed from the data structure.

Use SML lists to implement a stack three functions: `isEmpty`, `push`, and `pop`. Write `pop` so that it returns the element removed *and* the new stack, as you did above for a queue.

3. Consider the following SML data structure. It is essentially a binary tree with operations at the internal nodes and numbers at the leaves.

```
datatype aExpn = NUMBER of int
               | ADDN   of aExpn * aExpn
               | SUBTR  of aExpn * aExpn
```

a. Write an SML function that evaluates such an expression.

b. Write another SML function that translates such an expression into a fully-parenthesized algebraic expression, like `"((1+23)-(32-9))"`.

4. Consider an SML data structure for expressions in propositional logic in which the variables are indexed by integers: $v_0, v_1, v_2, \ldots$.

```
datatype pExpn = FALSE
               | TRUE
               | VARIABLE of int
               | NOT      of pExpn
               | AND      of pExpn * pExpn
               | OR       of pExpn * pExpn
```

a. Write an SML function `vlist` that, given a `pExpn`, returns a list of the variables in the expression.

b. Write an SML function `simplify` that simplifies an expression by removing unnecessary instances of the constants `FALSE` and `TRUE`. It will replace, for example,

- `NOT TRUE` with `FALSE`,
- `NOT FALSE` with `TRUE`,
- `AND(TRUE,`anything`)` with anything,

- `AND(FALSE,`anything`)` with `FALSE`,
- `OR(FALSE,`anything`)` with anything, and
- `OR(TRUE,`anything`)` with `TRUE`.

c. Write an SML function `substitute` that accepts a `pExpn`, a variable, and another `pExpn` and substitutes the second expression for every occurrence of the variable in the first expression. For example,

- `substitute (VARIABLE 3) (VARIABLE 3) TRUE` returns `TRUE`, and
- `substitute expn (VARIABLE 5) any` returns `expn` unchanged, if `VARIABLE 5` does not appear in `expn`.
- `substitute (AND (VARIABLE 3, u)) (VARIABLE 3) v` returns `(AND (v,u))`.

d. A logical expression is *satisfiable* if there is some assignment to its variables that makes the expression true. Using the functions that you have written, write an SML function `isSat` that tells whether a `pExpn` is satisfiable. Observe that an expression without variables simplifies to `FALSE` or `TRUE`, and an expression `e` with variable `VARIABLE k` is satisfiable if and only if one of the following is satisfiable.

```
substitute e (VARIABLE k) FALSE
substitute e (VARIABLE k) TRUE
```

5. What is an exception? What happens when one is raised? Give some examples of when an exception should (or should not) be used.

6. Let A and B be two sets of ordered pairs. The set-theoretic *join* of A and B is the set of all pairs (s, t) such that there is an element u with (s, u) in A and (t, u) in B. Write an SML function that computes the join of two *lists* of pairs. It will have the type signature

```
join : ('a * ''c) list -> ('b * ''c) list -> ('a * 'b) list
```

Motivation: The join described here is a simplified version of a common operation on databases. Think of A as representing classes and times; a pair (c, h) is in A if c is a class that meets at time h. Then the join of A with itself is a list of pairs of classes that conflict with each other.

7. A *relation* is a set of ordered pairs. A relation R is *transitive* if, whenever (a, b) and (b, c) are in R, the pair (a, c) is also in R. In SML, we can think of a list of ordered pairs instead of a set.

a. Assume that the `member` function is present, and write an SML predicate that identifies transitive relations.

b. What is the complexity of your predicate?

8. Write a function that takes a string `u` and another string `v` and creates a new string by substituting `v` for every occurrence of the character `#"X"` in `u`. Assume that the character `#"X"` does not appear in `v`. This is a simple version of the "textual transformation of programs" that could be done with a Turing machine.

9. Although the list `[1,[2,3]]` is a conceptually-natural object, the strict type-checking in SML will not accept it. One cannot mix integers and lists of integers in the same list. We can circumvent the limitation by creating a new type of "generalized lists."

```
datatype 'a genlist =
    glNil
  | glAtom of 'a
  | glList of 'a genlist list;
```

Then the list above can be represented as

```
glList [glAtom 1, glList [glAtom 2, glAtom 3]] : int genlist
```

Write a function `flatten` that takes out the extra structure and returns an ordinary list of atoms. The result of applying `flatten` to the list above is `[1,2,3]`. The result of applying `flatten` to a `glAtom` is a singleton list.

```
flatten : 'a genlist -> 'a list
```

10. What is the source of the names of the characters in the Liars Problem of Assignment 9? Can you come up with better names?

11. Prove by induction that $2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1$ for $0 \le n$.

12. Rewrite the lazy list data type from Assignment 9 so that the head, as well as the tail, is evaluated lazily.

13. [Longest common sublists] The list `slst` is a *sublist* of the list `lst` if the elements of `slst` occur, in order, in `lst`. For example, the list `[1,1,2,3]` is a sublist of `[1,1,1,2,2,3,4]`.

Given two lists, we can think about their common sublists. For example,

- `[1]` is a common sublist of `[1,1,2]` and `[2,2,1]`,
- `[2]` is a common sublist of `[1,1,2]` and `[2,2,1]`, and
- no list with more than one element is a common sublist of `[1,1,2]` and `[2,2,1]`.

Apparently, there may be more than one common sublist of maximum length.

Our problem is to write an SML function to find the length of a longest common sublist. Give arguments justifying the assertions below, and then use them to write the SML function.

Base cases: If one of the lists is empty, then any common sublist is also empty.

Recursive steps Suppose that the lists are `x::xs` and `y::ys`. If `x` = `y`, then a longest common sublist has the form `x::lcs`, where `lcs` is a longest common sublist of `xs` and `ys`. If `x` $\neq$ `y`, then a longest common sublist is either a longest common sublist of `x::xs` and `ys` or a longest common sublist of `xs` and `y::ys`.

Comment: The solution suggested here runs in $O(2^n)$ time, where n is the sum of the lengths of the two lists. (Verify!) In an algorithms course, you will learn about dynamic programming. Using that technique, one can find a longest common substring—not just its length—in $O(n^2)$ time.

14. What is the largest two's complement signed integer that can be expressed with five bits? The smallest (*i.e.,* most negative)? What are the largest and smallest *unsigned* five-bit integers?

15. Express −47 as an eight-bit two's complement binary integer.

16. Your invisible classmate, Witt O'Taclu, thinks that one can negate a two's complement signed integer by simply changing the sign bit.

a. Using the operators `&`, `|`, and `^` to represent, respectively, bitwise "and," "or," and "xor," write an expression that transforms a 32-bit signed integer k as Witt suggests. You will, or course, have to use k. You may also use `maxInt` and `minInt`, the largest and smallest (most negative) signed integers.

b. What is the value of the 32-bit signed integer k after Witt has "negated" it?

17. Suppose that a number k is represented as a 32-bit two's complement integer. Suppose that k is negated, and the resulting bits are viewed as an *unsigned* number. In terms of k, what is the value of that number? (Hint and/or caution: The original value of k may be negative, zero, or positive.)

18. When adding two bits, what logic gate (or, equivalently, operation from propositional logic) represents the sum bit? Which one represents the carry bit?

19. Write a power function that computes b^e using only $O(\lg e)$ multiplications. (The base-two logarithm of e, written $\lg e$, is one less than the number of bits in the binary representation of e.)

20. This problem is about finding a particular substring in a string of characters. Work over the two-character alphabet containing 0 and 1. Draw the diagram of a DFA that accepts the strings which contain the substring 01011. (Caution: Clearly, your DFA will match one character at a time, but it may not go all the way back to the start state when a match fails. Think about what happens when your DFA processes the string 01010111.)

21. Consider the substring search problem: Given two strings s and t, determine whether s appears as a substring of t. The Python function `string.find(`s`,`t`)` solves the problem; it returns the first index in t which begins a substring equal to s or -1 if there is no such substring.

Here is a slightly different SML function that converts strings to lists of characters and returns a boolean value.

```
fun substr s t =
    let
        fun isPrefix ul      nil     = null ul
          | isPrefix nil     _       = true
          | isPrefix (u::us) (v::vs) =
                u=v andalso isPrefix us vs;
        fun isSubstr ul nil     = null ul
          | isSubstr ul (v::vs) =
                isPrefix ul (v::vs) orelse isSubstr ul vs;
    in
        isSubstr (explode s) (explode t)
    end;
```

a. Suppose that s and t have lengths m and n, respectively. In the worst case, when s is not a substring of t, how many character comparisons are made by `substr`?

b. Write a different SML function that gives the same results as `substr` but does so in $O(m + n)$ time. Assume a two-letter alphabet. (Hint: Construct a DFA that looks for the substring s and run it on t. See Problem 20.)

22. Consider the ten-character alphabet consisting of the digits 0 through 9, and let the digit-sum of a string be the arithmetic sum of the characters. The digit-sum of the empty string is zero. Draw the diagram of a DFA that accepts all the strings whose digit sum is a multiple of 5.

23. Using the ten-digit alphabet, write a three-state NFA that accepts the set of all strings that end with the digits 47.

24. Design a Turing machine that operates on the ten-digit input alphabet of the previous problem. It begins with a single string on the input tape, viewed as the decimal representation of a number N. It halts with the decimal representation of $2N$ on the tape. (Make the following simplifying assumptions: At the start, the head is scanning the *rightmost* character of the input. When the machine halts, its head is scanning the *leftmost* character of the input.)

25. Consider a Turing machine C_w which erases its input, writes w on the tape, and halts while scanning the leftmost character of w.

a. Design the Turing machine C_{011}.

b. Think about how you would design the Turing machine C_w for some other word w. Imagine how you would design a Turing machine D (if you were asked to do so on a final exam, for example) that takes a word w and produces a description of C_w.

26. In class, we often used the function `uniquify` as an example. Write a version of `uniquify` that runs on a list of length n in $O(n^2)$ time (worst case). Verify the time bound. You may assume that the `member` function is available and runs in time proportional to the length of the list.

27. Here are two implementations of Euclid's algorithm for finding the greatest common divisor of two positive integers. In terms of the number of bits required to represent the original arguments `a` and `b`, what is the (worst case) time complexity of each one?

```
fun gcdOne (a,b) =
    if a = 0
       then b
    else if b = 0
       then a
    else if a < b
       then gcdOne(a,b-a)
       else gcdOne(a-b,a);
```

```
fun gcdTwo (a,b)
    if a = 0
       then b
       else gcdTwo(b, a mod b);
```

28. What is the purpose of a stack frame? What values are normally stored in one?

29. Write a short CS41B subprogram to copy the contents of a block of memory from one location to another. Follow the usual CS41B calling conventions. Before the call to the subprogram, two arguments are pushed onto the stack, in order:

- the address of the beginning of the source block and
- the address of the beginning of the destination block.

On entry to the subprogram, register `r3` will contain a third parameter, the size of the transfer or the number of bytes that are to be copied. Remember that the CS41B word-size is two bytes. Assume that the source and destination regions do not overlap. If this were a real final examination, you would be given a list of CS41B instructions. (Actually, if this were a real final examination, you would not be asked to write CS41B code. The problem is here to give you practice with the aspects of the computational model on which you will be tested.)

30. Most programming languages have a provision for floating point constants, like 123.45 and $-0.67e{+}2$. Here is one possible syntactic specification for floating point constants.

$$\begin{aligned}
F &::= [S](G \mid H)\\
G &::= D^+ \,.\, D^+\\
H &::= 0.\, D^+ e\, S\, D^+\\
D &::= 0 \mid 1 \mid \ldots \mid 9\\
S &::= + \mid -
\end{aligned}$$

Let $\mathcal{A}$ be the alphabet consisting of the ten digits, the plus and minus signs, the decimal point, and the letter e.

a. Design an NFA over the alphabet $\mathcal{A}$ that accepts the words that are correct floating point constants according to the above specification.

b. [Long and complicated] Design a DFA that accepts floating point constants.