

I. COMPUTABILITY: 1/17 Lecture, Ch. 2 of text book

1. Why are programs best considered to represent partial rather than total functions?
2. For each of the following function definitions, give the graph of the function. Say whether this is a partial function or a total function. If the function is partial, say where the function is defined and undefined:
 - (a) $f(x) = \text{if } x + 2 > 3 \text{ then } x * 5 \text{ else } 1 / x$
 - (b) $f(x) = \text{if } x < - 2 \text{ then } 3 \text{ else } f(x-2)$
 - (c) $f(x) = \text{if } x = 0 \text{ then } 1 \text{ else } f(x-1)$(c.f., Exercise 2.1, Mitchell, p. 16.)
3. What are three formal definitions of the class of computable functions?
4. What does it mean to say that a language is Turing-complete?
5. List three computer languages (including one not mentioned in class) that are Turing-complete?
6. Name one language (not mentioned in class) that you think is not Turing-Complete? What is it about the language that makes you think it is not Turing-Complete? If you use any web sites or other references to figure this out, please cite them.
7. On a Solaris machine (e.g., `pandora.cs.utsa.edu`), look at the man page for `cs`, do you think `cs` is Turing-complete? Why or why not?
8. What does the Church-Turing thesis tell us about the relative power (in terms of the kinds of functions that are computable) of the various general purpose programming language?
9. Usually, most general purpose programming languages are considered to be Turing-Complete; this is a slight simplification. What limitation(s) does a real program in a real programming language running on a real computer have that would prevent it from successfully executing an algorithm that solves a problem that is, in principle, computable?
10. Assuming you are given a program Halt_0 that can be used to determine whether a program that requires no input halts (c.f., Exercise 2.2, Mitchell, p. 17.), is it possible to solve the Halting Problem using Halt_0 ? If so, explain your answer by describing how a program solving the halting problem would work. If you do not think Halt_0 would allow one to solve the Halting Problem, explain briefly, why you think so?

II. LISP/SCHEME: 1/22, 1/24 Lecture; Ch. 3 of text book

1. What was the original application domain of the Lisp language?
2. List 4 major innovations of Lisp?
3. What are the differences between the `eq`, `eql`, `equal` and `=` functions in Lisp. What are the names of the equivalent functions in Scheme? (If you use an external reference, please cite your source.)
4. What list does the scheme program:

```
(cons
  'a
  (cons
    (car (cons 'b '()))
    (cdr (cons 'c (cons 'd '())))))
```

evaluate to? Diagram the in-memory layout of the data structure representing this list.

5. Do Exercise 3.1 in Mitchell (cons cells)
6. Write a Scheme function that, when called with `(factorial n)`, calculates the n^{th} factorial.
7. Write a Scheme function that, when called with `(count lst sym)`, counts the number of occurrences of symbols equal? to `sym` in the list `lst`, in any nested lists embedded in `lst` in any lists nested nested lists) inside `lst`, or For example, `(count '(a b (a b ((b)))) b) 'b)` should evaluate to 4.
8. Do Exercise 3.4, Mitchell, p. 42 about the `compose2` function
9. Do Exercise 3.5, Mitchell, p. 42 about garbage collection and computability