Practise Exercises

1 Concepts, terminology, etc.

Note that I am not likely to ask you to define anything on a midterm/exam. So, memorizing the definitions is not a good strategy for test preparation. Instead, I suggest that you make sure you understand each of the concepts below: can explain them, give examples in various languages, etc.

1. What is a language?
2. What is a programming language?
3. What are syntax and semantics of a programming language?
4. Describe the translation process.
5. What is the difference between compilation and interpretation? What are the benefits of compilation? Of interpretation?
6. What is declarative programming?
7. On a simple example demonstrate the difference between denotational and operational semantics (other than the example from the lecture slides).
8. State five properties of a "good" programming language.
9. List three features of functional programming.
10. What is referential transparency? Give an example.
11. What is manifest interface principle?
12. There is no assignment statement in functional programming. Why does `let` not violate this principle?
13. Define: linear recursion, tail recursion, structural recursion, mutual recursion, flat vs. deep/tree recursion. Give an example of each in Scheme.
14. What is the scope of a variable?
15. What is the difference between Scheme `let`, `let*`, and `letrec`.
16. Give an example of using an accumulator variable to improve efficiency.
17. Why is tail recursion required in all implementations of Scheme?
18. In what way does induction correspond to recursion? Give an example.
19. How can I write a function in Scheme that accepts variable number of arguments?
20. What is Continuation Passing Style? Give an example.
21. What is currying? Give an example of a curried function. Give an example of a non-curried function.
22. How is the `let` statement "syntactic sugar"?
23. For each of the following, provide a definition / explanation and an example in ML.
    (a) Type safe
    (b) Static vs. dynamic type-checking
    (c) Type inference
    (d) A sound type system

(e) Pattern matching
(f) Currying
(g) Type synonym
(h) Enumerated types
(i) Variant types
(j) Recursive types
(k) Mutually recursive types
(l) Parametric polymorphism (lists, functions)
(m) A type variable

24. Explain: "List is a parametric type".
25. Explain: "an ML function takes exactly one argument".
26. What is name resolution?
27. What is lexical scope? Give an example.
28. What is dynamic scope? Give an example.
29. Why do we use exceptions?
30. Why do we need exceptions in a strongly typed language?
31. What is an immutable data type?
32. How is the type of an ML reference cell determined?
33. How are assignment restrictions in ML enforced by the type system? Why is it important?
34. List three language features that distinguish Haskell from ML.
35. Explain lazy evaluation on an example.
36. Explain what overloading is.
37. Why do we like type classes?
38. What is a type class restriction, type context? Give an example.
39. Give an example of a curried value constructor. Give an example of a curried type constructor.
40. Give an example of a definition of and and a call to a function, which results in different behaviour in a lazy and in a eager programming language.
41. What do we mean when we say Haskell is pure?
42. State three features of Logic Programming.
43. What is the differenence between function and relation?
44. Give a (recursive) definition of a valid FOL formula.
45. Give a (recursive) definition of a valid Prolog term.
46. What is a Horn clause? A Prolog fact? A Prolog rule?
47. How is disjunction ';' in Prolog merely syntactic sugar?
48. How does Prolog interpret free variables in a rule?

49. What is unification? How is it different from, say, pattern matching in SML?
50. What is a substitution? Instantiation? Common instance? Unifier? Give examples of each.
51. What is the most general unifier? Give an example.
52. Give the three steps of an execution of a Prolog program. Again.
53. How is arithmetic in Prolog special?
54. What is negation as failure? How does it relate to "true" logical negation? Give an example that illustrates the difference. Explain the term "closed world assumption", with an example.
55. When is using negation (`\+`, `not`) in Prolog "safe"? What is a guard?
56. How does cut work? Give an example. Now give another example. Draw Prolog search trees for both examples.
57. Which language(s) that we saw in this course do you like most / least? Why?

2 Scheme

In the following, `fold` means either `foldr` or `foldl`.

1. Write a procedure that, given a list of non-empty lists, returns the list of first elements of the sub-lists. Use `map`.
2. Write a procedure that, given a list of non-empty lists, returns the list of first elements of the sub-lists. Use `fold`.
3. Write a procedure that returns the sum of all elements in the input list. Use `fold`.
4. Write a procedure that returns the sum of all elements in the input list. Use `apply`.
5. Write a procedure `(filter f l)` that returns a list of elements of `l` that satisfy `f`. Use `fold`.
6. Write a procedure that counts the number of elements in the list, on any nesting level.
7. Rewrite the above procedure using `fold`.
8. Rewrite the above procedure so that it is tail-recursive.
9. Write a procedure that returns a list of elements from the input list, on any nesting level. That is, it "flattens" the list. Which, if any, higher-order procedures would you like to use here?
10. Rewrite the above procedure (flatten) so that it is tail-recursive.
11. Redo all the matrix manipulation examples yourself, without looking at the notes.

3 ML

1. For each procedure in section 2, determine whether the corresponding function can be written in ML. If you think it can,
   (a) determine what type the function should have, and
   (b) implement the function in ML
2. For any valid ML expression that involves the types covered in class (unit, booleans, integers, reals, strings, tuples, lists, functions), you should be able to determine the type of the given expression. Make up some to practise and check your answers with the SML interpreter.

3. Consider the following datatype:

```
datatype 'a NonEmptyTree = Node of 'a * ('a NonEmptyTree list);
```

Implement the following functions:

- `numLeaves : 'a NonEmptyTree -> int` which returns the number of leaves in the input tree.
- `numNodes : 'a NonEmptyTree -> int` which returns the number of nodes in the input tree.
- `height : 'a NonEmptyTree -> int` which returns the height of the input tree.
- `tmap : ('a -> 'b) -> 'a NonEmptyTree -> 'b NonEmptyTree` which returns the tree which results from applying the input function to every node of the input tree.

Think carefully about using higher-order functions (map, fold) in your solution.

4. Let's change the datatype:

```
datatype 'a NonEmptyTree' = Node' of 'a * 'a Forest
     and 'a Forest = Empty | Trees of 'a NonEmptyTree' * 'a Forest;
```

Rewrite all of the above functions to work with this datatype.

5. Write a function `curry` that returns a curried version of its binary input function. What is the type of `curry`?

6. Write a function `curry` that returns a curried version of its ternary (three arguments) input function. What is the type of `curry`?

7. Write a function `uncurry` that returns a non-curried version of its binary curried input function. What is the type of `uncurry`?

8. Write a function `uncurry` that returns a non-curried version of its ternary (three arguments) curried input function. What is the type of `uncurry`?

4 Haskell

1. For each procedure in section 2, determine whether the corresponding function can be written in Haskell. If you think it can,
   (a) determine what type the function should have, and
   (b) implement the function in Haskell

2. Define a function `double` which returns `[2*a0, 2*a1, 2*a2, ...]` when called with an input list `[a0, a1, a2, ...]`. What is the type of `double`?

3. Trace every step of the evaluation of `take 3 (double nats)`.

4. The Taylor series for the exponential function e^x at $a = 0$ is

$$1 + x^1/1! + x^2/2! + x^3/3! + \ldots$$

Define the function `expTaylor x` which returns a list, so that summing this list gives the above series.

5. Consider the following datatype:

```
data BST a = Empty | Node a (BST a) (BST a)
```

Suppose we add

```
data BST a = Empty | Node a (BST a) (BST a) deriving Eq
```

Explain what this does.

Suppose we want to define equality of `BST`s differently: we consider two `BST`s equal if their in-order traversals visit exactly the same elements in the same order. Implement this equality.

5 Prolog

- Suppose we implement sets with Prolog lists. Implement the following set operations:
  - subset
  - superset
  - set equality
  - intersection
  - union
- Define a predicate `replace(?oldList, ?oldElem, ?newElem, ?newList)` which holds iff `newList` is the same as `oldList`, except all occurrences of `oldElem` are replaced with `newElem`.
- Suppose we have the following database:

```
(1)  holiday(friday,april14).
(2)  weather(friday,fair).
(3)  weather(saturday,fair).
(4)  weather(sunday,fair).

(5)  weekend(saturday).
(6)  weekend(sunday).

(7)  picnic(Day) :- weather(Day,fair), weekend(Day).
(8)  picnic(Day) :- holiday(Day,april14).
```

Give all answers, in order, to the query `picnic(When)`. Draw a Prolog search tree. Now rewite `picnic/1` as follows:

```
(7)  picnic(Day) :- weather(Day,fair), !, weekend(Day).
(8)  picnic(Day) :- holiday(Day,april14).
```

Give all answers, in order, to the query `picnic(When)`. Draw a Prolog search tree. Now rewite `picnic/1` as follows:

```
(7)  picnic(Day) :- weather(Day,fair), weekend(Day), !.
(8)  picnic(Day) :- holiday(Day,april14).
```

Give all answers, in order, to the query `picnic(When)`. Draw a Prolog search tree.