Practice Midterm Questions

Help

Warning: The hard deadline has passed. You can attempt it, but **you will not get credit for it**. You are welcome to try it as a learning exercise.

These are the instructions that will also be on the real midterm, but of course this is just a practice midterm:

You have 65 minutes to complete the exam. Only your first submission will count toward your grade.

You may use any course materials (videos, slides, reading notes, etc.). You may use the ML REPL and a text editor. You may use the ML standard-library documentation.

You may not use the discussion forum. You may not use other websites related to programming or ML. (Sites like dictionaries for translating English words are okay to use.)

- [ ] **In accordance with the Coursera Honor Code, I (KL Tah) certify that the answers here are my own work.**

Question 1

[8 points total] Check a box if and only if it is an accurate description of ML

- [ ] Function arguments are evaluated before being passed to functions.
- [ ] ML is dynamically scoped.
- [ ] All functions can be called recursively.
- [ ] Functions are first-class expressions.

Question 2

[8 points total] Check a box if and only if it is an example of unnecessary function wrapping

- [ ]
```
fun increment x = x + 1;
```

- [ ]
```
fun map x y = List.map x y;
```

- [ ]
```
fun foo f xs = 1 +
        foldr (fn (x,y) => x * (y+1))  0 (map f xs)
```

- [ ]
```
fun bar xs = if xs = []
             then 0
             else 1 + bar xs
```

Question 3

[14 points total] Lexical scoping is a crucial part of code execution in many programming languages, including ML. For each statement below, check the box if and only if the statement is true regarding this ML code. Consider each statement *after* the identified line is executed.

```
1-   val x = 50
2-   val y = 3
3-   val z = 10
4-   val f = fn z => z
5-   val a =
6-   let
7-     val x = 3*x
8-     val z = y*z
9-   in
10-    x*z
11-  end
12-  fun f x z = x + y + z
13 -
```

- [ ] On line 4, the variable z inside the function body is bound to 10.
- [ ] On line 7, x is bound to 150.
- [ ] On line 8, z is bound to 30.
- [ ] On line 10, z is bound to 10.

- [ ] On line 12, the variable x inside the function body is bound to 50.
- [ ] On line 12, the variable y inside the function body is bound to 3.
- [ ] On line 13, x is bound to 50.

Question 4

[10 points total] For each type below, check the box if and only if the type is a valid type for the function `foo`. Do not only select the most general type, also select less general types.

```
fun foo f x y z =
        if x >= y
        then (f z)
        else foo f y x (tl z)
```

- [ ] `(int -> real) -> int -> int -> int -> real`
- [ ] `(string list -> bool list) -> int -> int -> string list -> bool list`
- [ ] `('a list -> 'b list) -> int -> int -> 'a list -> 'b list`
- [ ] `(int list -> 'b list) -> int -> int -> 'b list -> int list`
- [ ] `('a list -> string list) -> int -> int -> 'a list -> 'a option list`

Question 5

[10 points total] Several correct implementations of the `factorial` function appear below. Check the box next to a definition if and only if all recursive functions calls (possibly including recursive helper functions) are tail calls.

- [ ]
```
fun factorial i =
  if i = 0
  then 1
  else i * factorial (i - 1)
```

- [ ]
```
fun factorial i =
let
  fun factorialhelper (accum,i) =
    if i = 0
    then accum
    else factorialhelper (accum*i, i-1)
in
  factorialhelper (1,i)
end
```

- [ ]
```
fun factorial i =
let
  fun factorialhelper (start,i) =
    if start <> i
    then start * factorialhelper (start+1, i)
    else start
in
  if i=0
  then 1
  else factorialhelper (1,i)
end
```

- [ ]
```
fun factorial i =
  case i of
      0 => 1
    | x => x * factorial (i-1)
```

Question 6

[8 points total] Partial application involves passing less than the full number of arguments to a curried function. Given the curried function below, check the box if and only if the given function call is paired with a correct type for the returned function.

```
fun baz f a b c d e = (f (a ^ b))::(c + d)::e
```

- [ ]
```
Call: baz (fn z => 3)
Return type: string -> string -> int -> int -> int list -> int list
```

- [ ]
```
Call: baz (fn z => 10) "foo"
Return type: string -> int -> int -> int list -> int list
```

- [ ]
```
Call: baz (fn z => 10) "foo"
Return type: int -> string -> int -> int list -> int list
```

- [ ]
```
Call: baz (fn z => 10) "foo" "bar"
Return type: int -> int -> int list -> int list
```

Question 7

[18 points total] Consider the two functions `maybeEven` and `maybeOdd` below, which are *mutually recursive*. For each statement below, check the box if and only if the statement is true regarding this ML code. Notice that these functions have some unconventional behaviour.

```
fun maybeEven x =
        if x = 0
        then true
        else
        if x = 50
        then false
        else maybeOdd (x-1)

and maybeOdd y =
        if y = 0
        then false
        else
        if y = 99
        then true
        else maybeEven (y-1)
```

- [ ] Evaluation of the call `maybeEven 50` requires 25 calls to `maybeOdd`.
- [ ] The call `maybeOdd ~1` does not terminate.

- [ ] The call `maybeEven 1` does not terminate.
- [ ] Evaluation of the call `maybeOdd 6` requires 3 calls to `maybeEven`.
- [ ] Every call from `maybeEven` to `maybeOdd` or from `maybeOdd` to `maybeEven` is a tail call.
- [ ] Evaluating any call to `maybeEven` will always involve a call to `maybeOdd`.
- [ ] The functions `maybeEven` and `maybeOdd` have the same type.
- [ ] For input x > 50, `maybeEven` always returns `false`.
- [ ] The return types of `maybeEven` and `maybeOdd` are different.

Question 8

[9 points total for questions 8, 9, and 10 together] The next three questions, including this one, relate to this situation: Types are often abstract representations for real world values. For each problem below, decide which type is the best choice to represent the given data.
This problem: Values of the type will represent multiple country names.

- ○ `int`
- ○ `string`
- ○ `int list`
- ○ `string list`
- ○ `(string * int) list`

Question 9

This problem: Values of the type will hold a person's last name.

- ○ `int`
- ○ `string`
- ○ `int list`
- ○ `string list`
- ○ `(string * int) list`

Question 10

This problem: Values of the type will hold a collection of student names and their grades on an assignment.

- `int`
- `string`
- `int list`
- `string list`
- `(string * int) list`

Question 11

[15 points total for questions 11, 12, 13, 14, and 15 together] The next 5 questions, including this one, are similar. Each question uses a slightly different definition of an ML signature `DIGIT` with the same structure definition `Digit` below. The `Digit` structure implements one-digit numbers that wrap around when you increment or decrement them.

```
structure Digit :> DIGIT =
struct
type digit = int
exception BadDigit
exception FailTest
fun make_digit i = if i < 0 orelse i > 9 then raise BadDigit else i
fun increment d = if d=9 then 0 else d+1
fun decrement d = if d=0 then 9 else d-1
val down_and_up = increment o decrement (* recall o is function composition
 *)
fun test d = if down_and_up d = d then () else raise FailTest
end
```

In each problem, the definition of `DIGIT` matches the structure definition `Digit`, but different signatures let clients use the structure in different ways. You will answer the same question for each `DIGIT` definition by choosing the best description of what it lets clients do.

In this question, the definition of `DIGIT` is:

```
signature DIGIT =
```

```
sig
type digit = int
val make_digit : int -> digit
val increment : digit -> digit
val decrement : digit -> digit
val down_and_up : digit -> digit
val test : digit -> unit
end
```

- The type-checker prevents the client from calling `Digit.test` with the expression `Digit.test e`, for any expression `e` that evaluates to a value `v`.
- There are calls by clients to `Digit.test` that can type-check, but `Digit.test 10` does not type-check.
- The client call `Digit.test 10` type-checks and causes the `Digit.FailTest` exception to be raised.
- The client call `Digit.test 10` type-checks and evaluates without raising an exception.

Question 12

In this question, the definition of `DIGIT` is:

```
signature DIGIT =
sig
type digit = int
val make_digit : int -> digit
val increment : digit -> digit
val decrement : digit -> digit
val down_and_up : digit -> digit
end
```

- The type-checker prevents the client from calling `Digit.test` with the expression `Digit.test e`, for any expression `e` that evaluates to a value `v`.
- There are calls by clients to `Digit.test` that can type-check, but `Digit.test 10` does not type-check.

- The client call `Digit.test 10` type-checks and causes the `Digit.FailTest` exception to be raised.
- The client call `Digit.test 10` type-checks and evaluates without raising an exception.

Question 13

In this question, the definition of `DIGIT` is:

```
signature DIGIT =
sig
type digit = int
val make_digit : int -> digit
val increment : digit -> digit
val decrement : digit -> digit
val test : digit -> unit
end
```

- The type-checker prevents the client from calling `Digit.test` with the expression `Digit.test e`, for any expression `e` that evaluates to a value `v`.
- There are calls by clients to `Digit.test` that can type-check, but `Digit.test 10` does not type-check.
- The client call `Digit.test 10` type-checks and causes the `Digit.FailTest` exception to be raised.
- The client call `Digit.test 10` type-checks and evaluates without raising an exception.

Question 14

In this question, the definition of `DIGIT` is:

```
signature DIGIT =
sig
type digit
val make_digit : int -> digit
val increment : digit -> digit
```

```
val decrement : digit -> digit
val down_and_up : digit -> digit
val test : digit -> unit
end
```

- The type-checker prevents the client from calling `Digit.test` with the expression `Digit.test e`, for any expression `e` that evaluates to a value `v`.
- There are calls by clients to `Digit.test` that can type-check, but `Digit.test 10` does not type-check.
- The client call `Digit.test 10` type-checks and causes the `Digit.FailTest` exception to be raised.
- The client call `Digit.test 10` type-checks and evaluates without raising an exception.

Question 15

In this question, the definition of `DIGIT` is:

```
signature DIGIT =
sig
type digit
val increment : digit -> digit
val decrement : digit -> digit
val down_and_up : digit -> digit
val test : digit -> unit
end
```

- The type-checker prevents the client from calling `Digit.test` with the expression `Digit.test e`, for any expression `e` that evaluates to a value `v`.
- There are calls by clients to `Digit.test` that can type-check, but `Digit.test 10` does not type-check.
- The client call `Digit.test 10` type-checks and causes the `Digit.FailTest` exception to be raised.
- The client call `Digit.test 10` type-checks and evaluates without raising an exception.

☐ **In accordance with the Coursera Honor Code, I (KL Tah) certify that the answers here are my own work.**

Submit Answers | Save Answers

You cannot submit your work until you agree to the Honor Code. Thanks!