

Fall 2002 - Programming Languages Qualifier

Computer Science Department
University of Massachusetts Lowell
Lowell, MA 01854

Sept. 28, 2002.

Each problem is worth 10 points.

Part A. Do any 4.

1. **Environments of Evaluation.** Consider the following Scheme code in which one function is passed as an argument to another function. Note that the function called `pass-me` is variadic.

```
(define pass-me
  (lambda args
    (apply * args)))
```

```
(define calculate
  (lambda (proc a b c)
    (proc a b c)))
```

Using the diagramming notation of either Manis & Little or Abelson & Sussman, show the complete environment of evaluation during evaluation of `(calculate pass-me 4 5 6)`

2. **Recursion.** Consider the following recursive implementation, which filters a list to find all of the integers.

```
(define foo
  (lambda (lis)
    (helper lis '())))
```

```
(define helper
  (lambda (lis acc)
    (if (null? lis)
        '()
        (if (integer? (car lis))
            (cons (car lis) (helper (cdr lis) acc))
            (helper (cdr lis) acc))))))
```

Assume that you are using a Scheme interpreter that knows how to recognize and optimize tail recursion. Tell whether or not the function called "helper" can be optimized for tail-recursion. Support your answer with an analysis of worst-case behavior.

3. **Referential Transparency.** Discuss the relationship between the `ref` type constructor in ML and referential transparency.

Hints: what is the `ref` type constructor? How is it related to assignment? What kind of assignment (if any) does ML support? What is Referential Transparency? Why does the substitution model of computation break down in the presence of assignment? Give appropriate examples to illustrate your claims.

4. **Functions as Objects.** Given the Scheme definition

```
(define (cons x y)
  (define (set-x! v) (set! x v))
  (define (set-y! v) (set! y v))
  (define (dispatch m)
    (cond ((eq? m 'car) x)
          ((eq? m 'cdr) y)
          ((eq? m 'set-car!) set-x!)
          ((eq? m 'set-cdr!) set-y!)
          (else (error "Undefined Operation -- CONS" m))))
  dispatch)
```

- (a) (4 pts) Provide appropriate definitions for `car` and `cdr`, given this versions of `cons`.
- (b) (4 pts) Provide appropriate definitions for `set-car!` and `set-cdr!`, given this versions of `cons`.
- (c) (2 pts) If we define

```
(define test (cons 1 (cons 2 (cons 3 4))))
```

What will `(car test)` return? What will `(cdr test)` return? Explain carefully and precisely.

Hint: `(cons 1 2)` returns an object which is a **function**.

5. **Streams.**

- (a) (5 pts) Define the second-order function `stream-map`, which takes as arguments a function of one argument and a stream and returns the stream obtained by applying the function of one argument to each element of the original stream.
- (b) (5 pts) Show what YOUR `stream-map` would return if you were to execute the call:

```
(stream-map (lambda (x) (* 2 x)) integers)
```

where `integers` is defined by

```
(define ones (cons-stream 1 ones))
(define integers (cons-stream 1 (add-streams ones integers)))
```

and `add-streams` acts appropriately.

Part B. Do any 6.

6. Data Types.

- (a) (4 pts) Describe Inclusion Polymorphism and Parametric Polymorphism.
- (b) (4 pts) Give at least one clear example of each.
- (c) (2 pts) Explain how this is, or is not, relevant to the C language.

7. Parameter Passing Mechanisms. Discuss, with appropriate examples

- (a) Pass-by-name
- (b) Pass-by-reference
- (c) Pass-by-result
- (d) Pass-by-value
- (e) Pass-by-value-result.

For each mechanism you should mention a language that supports it. If you don't know, just say so.

8. Scoping Rules. Explain - being specific and with detail - the implementation problems and their solutions for

- (a) Static Scoping
- (b) Dynamic Scoping.
- (c) What will be printed by `sub1` in the following Pascal program under Static Scoping? Under Dynamic Scoping?

```
program main;
  var x : integer;
  procedure sub1;
  begin
    writeln('x = ', x)
  end;
  procedure sub2;
  var x : integer;
  begin
    x := 19;
    sub1
  end;
begin
  x := 5;
  sub2
end.
```

9. **Prolog Programming.** Consider the Prolog assertion (using standard Edinburgh syntax - lower case initiated strings denote constants, upper case initiated ones denote variables):

`equal(X, X).`

and the query

`equal(foo(X, Y), foo(w(Y), x(X)))?`

- What is the *occur-check*?
- What is *Unification*?
- Presented with the query above, what would a Prolog implementation that does not enforce the occur-check return and why?
- If you believe a Unifying Transformation for `foo(X, Y)` and `foo(w(Y), x(X))` exists, produce it, if you don't, explain why it cannot be produced.

10. **Encapsulation.**

- (3 pts) Define Encapsulation
- (7 pts) What are the mechanisms provided for the support of private types in Ada, ML and C++? What are their limitations?

11. **Denotational Semantics.** Assume the language of Regular Expressions has

Syntactic Categories

$A \in \mathbf{Alpha}$ *the alphabet*

$R \in \mathbf{RE}$ *regular expressions*

Where the regular expressions have BNF definition

$$R ::= A \mid \emptyset \mid (R + R) \mid (R \cdot R) \mid R^*$$

Value Domains

$\mathbf{Lang} = \text{formal languages (sets of string over } \mathbf{Alpha})$

Semantic Functions

$\mathcal{A} : \mathbf{Alpha} \rightarrow \mathbf{Lang}$

$\mathcal{D} : \mathbf{RE} \rightarrow \mathbf{Lang}$

Starting with the semantic function definitions:

$$\mathcal{A} \parallel A \parallel = \{A\}$$

$$\mathcal{D} \parallel A \parallel = \mathcal{A} \parallel A \parallel$$

where $\{A\}$ is the language containing the single character string, denoted by $\mathcal{A} \parallel A \parallel$

- (5 pts) Extend the semantic functions to provide the denotational semantics for regular expressions.

- (5 pts) Apply the semantic functions to obtain the denotation of the regular expression $A \cdot B \cdot (A^* + B^*)$, over the alphabet $\{A, B\}$. Show the details.

12. **Fixed Points.** Consider the function

$$F = \lambda g. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times g(n - 1)$$

- (a) What is the type of this function? Hint: use the type conventions for ML with regard to elementary types and their default operations.
- (b) Construct a sequence of approximations converging to its least fixed point.
- (c) What is the **Y** combinator?
- (d) How is the construction in (b) related to the **Y** combinator?

13. **Functions, Types and Currying.** Assume that f and g are two functions of one argument such that $\text{Range}(g) \subseteq \text{Domain}(f)$. The *composition* of f and g , $f \circ g$, is defined by

$$(f \circ g)(x) \equiv f(g(x)).$$

- (a) (5 pts) Define a function of two arguments, **compose**, which returns the composition of the two functions. What is the type of this function?
- (b) (5 pts) Define a *curried* version of **compose**. Can you think of a use for it? Any yes or no answer must be followed by an explanation.