MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Compvter Science

Final Examination Solutions 2001

Problem 1: Short Answer [20 points]

a. Given the following domains:

$$A = \{1\}_\perp$$
$$B = \{a,\ b\}_\perp$$

(i) [2 points]How many set theoretic functions are there in $A \to B$?

Solution: 9

(ii) [2 points] How many continuous functions are there in $A \to B$?

Solution: 5

b. In Scheme/R (Scheme with type reconstruction) give the most general type schemes of the following expressions, or state why the expression does not have a type:

(i) [1 points]

```
(lambda (f) (lambda (g) (lambda (x) (g (f x)))))
```

Solution:

```
(generic (t₁, t₂, t₃) (-> ((-> (t₁) t₂)) (-> ((-> (t₂) t₃)) (-> (t₁) t₃))))
```

(ii) [1 points]

```
(lambda (x) (x x))
```

Solution: Self application, no type

(iii) [1 points]

```
(letrec ((f (lambda (x) (if (id x) (id 0) (id 1))))
         (id (lambda (y) y)))
        (f #t))
```

Solution: Attempted polymorphic use of id in letrec bindings, no type

(iv) [1 points]

```
(lambda (id) (if (id #t) (id 0) (id 1)))
```

Solution: First class polymorphism, no type

c. Give the equivalent Scheme/XSP expression, and the type thereof, for each of the expressions in the previous part (part b).

(i) [2 points]

Solution:

```
(plambda (t₁ t₂ t₃)
   (lambda ((f (-> (t₁) t₂)))
      (lambda ((g (-> (t₂) t₃)))
         (lambda ((x t₁))
            (g (f x))))))
```

```
(poly (t₁ t₂ t₃)
   (-> ((-> (t₁) t₂)) (-> ((-> (t₂) t₃)) (-> (t₁) t₃))))
```

(ii) [2 points]

Solution:

```
expression:
(plambda (t₁)
   (lambda ((x (recof t₂ (-> (t₂) t₁))))
      (x x)))
```

```
type:
(poly (t₁) (-> (recof t₂ (-> (t₂) t₁)) t₁))
```

(iii) [2 points]

Solution:

```
expression:
(letrec ((f (-> (bool) int) (f (lambda (x)
                                  (if ((proj id bool) x)
                                      ((proj id int) 0)
                                      ((proj id int) 1)))))
         (id (poly (t) (-> (t) t)) (plambda (t) (lambda ((y t)) y))))
   (f #t))
```

```
type:
int
```

(iv) [2 points]

Solution:

```
expression:
(lambda ((id (poly (t) (-> (t) t))))
    (if ((proj id bool) #t) ((proj id int) 0) ((proj id int) 1)))

type:
(-> (poly (t) (-> (t) t)) int)
```

d. [2 points] Give the desugaring of the following Scheme/R expression

```
(match z
   ((cons 1 x) x)
   (x (cons 1 x)))
```

Solution:

```
(let ((Itop z))
   (let ((Ifail (lambda () (let ((x Itop)) (cons 1 x)))))
      (cons~ Itop (lambda (I1 I2)
                     (if (= I1 1)
                         (let ((x I2)) x)
                         (Ifail)))
             Ifail)))
```

e. [2 points] Use define-datatype to define the (queueof T) datatype that represents a queue with a list. For example, a (queueof int) would be represented by an integer list.

Solution:

```
(define-datatype (queueof T) (list->queue (listof T))))
```

Problem 2: State: FLK# [20 points]

Sam Antics is working on a new language with hot new features that will appeal to government customers. He was going to base his language on Caffeine from Moon Microsystems, but negotiations broke down. He has therefore decided to extend FLK! and has hired you, a top FLK! consultant, to assist with modifying the language to support these new features. The new language is called FLK#, part of Sam Antics' new .GOV platform. The big feature of FLK# is user tracking and quotas in the store. An important customer observed that government users tended to use the store carelessly, resulting in expensive memory upgrades. To improve the situation, the FLK# store will maintain a per-user quota. (A quota restricts the number of cells a particular user can allocate.) The Standard Semantics of FLK! are changed as follows:

$$w \in \text{UserID} = \text{Int}$$
$$q \in \text{Quota} = \text{UserID} \rightarrow \text{Int}$$
$$\gamma \in \text{Cmdcont} = \text{UserID} \rightarrow \text{Quota} \rightarrow \text{Store} \rightarrow \text{Expressible}$$

error-cont : Error $\rightarrow$ Cmdcont
  $= \lambda y . \ \lambda w . \ \lambda q . \ \lambda s . \ (\text{Error} \mapsto \text{Expressible} \ y)$

UserID is just an integer. User ID 0 is reserved for the case when no one is logged in. Quota is a function that when given a UserID returns the number of cells remaining in the user's quota. The quota starts at 100 cells, and a user's quota is tracked throughout the lifetime of the program (i.e., the quota is not reset upon logout). Cmdcont has been changed to take the currently logged in user ID, the current quota, and the current store to yield an answer. Plus, FLK# adds the following commands:

E	$::=$	$\ldots$	[Classic FLK! expressions]
	$\mid$	(login! w)	[Log in user w]
	$\mid$	(logout!)	[Log out current user]
	$\mid$	(check-quota)	[Check user quota]

(login! w) - logs in the user associated with the identifier w; returns w (returns an error if a user is already logged in or if the UserID is 0)

(logout!) - logs the current user out; returns the last user's identifier (returns an error if there is no user logged in)

(check-quota) - returns the amount of quota remaining

The definition of $\mathcal{E}[\![(\texttt{check-quota})]\!]$ is:

$\mathcal{E}[\![(\texttt{check-quota})]\!] =$
  $\lambda ekwq . \ \textbf{if } w = 0$
        $\textbf{then } \textit{error-cont} \text{ no-user-logged-in } \ w \ \ q$
        $\textbf{else } (k \ (\text{Int} \mapsto \text{Value } (q \ w)) \ w \ \ q) \ \textbf{fi}$

a. [5 points] Write the meaning function clause for $\mathcal{E}[\![(\texttt{login! } E)]\!]$.

   Solution:
   $\mathcal{E}[\![(\texttt{login! } E)]\!] =$
     $\lambda ek . \ \mathcal{E}[\![E]\!] \ e \ (\textit{test-int} \ \ \lambda iw . \ \textbf{if } (w = 0) \text{ and } (i \neq 0)$
                                      $\textbf{then } (k \ (\text{Int} \mapsto \text{Value } i) \ i)$
                                      $\textbf{else } \textit{error-cont} \text{ already-logged-in } \ w \ \textbf{fi})$

b. [5 points] Write the meaning function clause for $\mathcal{E}[\![(\texttt{logout!})]\!]$.

   Solution:
   $\mathcal{E}[\![(\texttt{logout!})]\!] =$
     $\lambda ekw . \ \textbf{if } (w = 0) \ \textbf{then } (\textit{error-cont} \text{ not-logged-in } \ w) \ \textbf{else } (k \ (\text{Int} \mapsto \text{Value } w) \ 0) \ \textbf{fi}$

4

c. [5 points] Give the definition of $\mathcal{E}[\![(\text{cell } E)]\!]$. Remember you cannot create a cell unless you are logged in.

Solution:
$\mathcal{E}[\![(\text{cell } E)]\!] =$
$\quad \lambda ek \, . \ \mathcal{E}[\![E]\!] \, e$
$\qquad (\lambda v w_1 q s \, . \ \textbf{if } (w_1 = 0) \text{ or } ((q \ w_1) = 0)$
$\qquad\qquad\quad \textbf{then } \textit{error-cont } \text{ error } w_1 \, q \, s$
$\qquad\qquad\quad \textbf{else } (k \ (\text{Location} \mapsto \text{Value } (\textit{fresh-loc } s))$
$\qquad\qquad\qquad\qquad w_1$
$\qquad\qquad\qquad\qquad (\lambda w_2 \, . \ \textbf{if } w_2 = w_1$
$\qquad\qquad\qquad\qquad\qquad\quad \textbf{then } (q \ w_1) \text{ -1}$
$\qquad\qquad\qquad\qquad\qquad\quad \textbf{else } (q \ w_1) \ \textbf{fi})$
$\qquad\qquad\quad (\textit{assign } (\textit{fresh-loc } s) \ v \ s))$
$\qquad\qquad\quad \textbf{fi})$

d. [5 points] Naturally, Sam Antics wants to embed some "trap doors" into the .GOV platform to enable him to "learn more about his customers." One of these trap doors is the undocumented `(raise-quota! n)` command, which adds n cells to the quota of the current user and returns 0. Give the definition of $\mathcal{E}[\![(\text{raise-quota! } E)]\!]$.

Solution:
$\mathcal{E}[\![(\text{raise-quota! } E)]\!] =$
$\quad \lambda ek \, . \ \mathcal{E}[\![E]\!] \, e \ (\textit{test-int } \ \lambda i w_1 q \, . \ \textbf{if } w_1 = 0$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{then } \textit{error-cont } \text{ error-no-user-logged-in } \ w \ \ q$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad \textbf{else } (k \ (\text{Int} \mapsto \text{Value } 0) \ w_1$
$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \lambda w_2 \, . \ \textbf{if } w_2 = w_1 \ \textbf{then } (q \ \ w_1) + i \ \textbf{else } (q \ \ w_1)) \ \textbf{fi}$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad \textbf{fi})$

5

Problem 3: Explicit Types [20 points]

Louis Reasoner has had a hard time implementing letrec in a call-by-name version of Scheme/XSP, and has decided to use the fixed point operator FIX instead. For example, here the the correct defintion of factorial in Louis' approach:

```
(let ((fact-gen (lambda ((fact (-> (int) int)))
        (lambda ((n int)) (if (= n 0) 1 (* n (fact (- n 1))))))))
  ((proj fix (-> (int) int)) fact-gen))
```

Thus `fix` is a procedure that computes the fixed point of a generating function. Ben Bitdiddle has been called on the scene to help, and he has ensured that Louis' Scheme/XSP supports recursive types using RECOF (see Appendix **??**).

a. [4 points] What is the type of `fact-gen`?

   Solution:

   ```
   (-> ((-> (int) int)) (-> (int) int))
   ```

b. [3 points] What is the type of `fix`?

   Solution:

   ```
   (poly (t) (-> ((-> (t) t)) t))
   ```

c. [3 points] What is the type of `((proj fix (-> (int) int)) fact-gen)`?

   Solution:

   ```
   (-> (int) int)
   ```

Ben Bitdiddle defined the call-by-name version of fix to be:

```
(let ((fix (plambda (t) (lambda ((f T1))
            (lambda ((x T2)) (f (x x))) (lambda ((x T2)) (f (x x)))))))
    ... fix can be used here ...

  )
```

d. [3 points] What is T1?

   Solution:

   ```
   T1 = (-> (t) t)
   ```

e. [4 points] What is T2?

   Solution:

   ```
   T2 = (recof x (-> (x) t))
   ```

f. [3 points] Louis has decided that he would like (`fix E`) to be a standard expression in his language. What is the typing rule for (`fix E`)?

Solution:

$$\frac{A \vdash E : (\text{-> (T) T})}{A \vdash (\text{fix } E) : \text{T}} \qquad [\textit{fix}]$$

Problem 4: Type Reconstruction [20 points]

With sales declining and customers flocking to competitors' products, the board of directors at Prophet.com has decided to oust CTO Louis Reasoner and has assigned you and Alyssa P. Hacker as the pro tempore co-CTOs. Alyssa believes the secret to regaining market share is to make Scheme/R more Internet-friendly. The next generation product, code-named Scheme/R $9i$ (the i stands for Internet), contains socket functionality to make it easier to write Internet servers.

A socket is like a stream or a channel in that you can read data from and write data to sockets. Sockets are named by a port number and also have a specific data type associated with them that determines the type of data that can be transmitted or received over the socket. (For the purpose of this problem, you can ignore any problems involved with opening more than one socket on the same port.)

We introduce a new type (socketof T) and six new forms:

- (int-socket E_{port}) returns a new integer socket.

- (bool-socket E_{port}) returns a new boolean socket.

- (unit-socket E_{port}) returns a new unit socket.

- (symbol-socket E_{port}) returns a new symbol socket.

- (read-all! E_{socket} E_{reader}) takes a socket and calls procedure E_{reader} once for each item remaining in the socket to be read; returns #u.

- (write! E_{socket} E_{datum}) Writes E_{datum} into the socket and returns #u.

Alyssa has written the following Scheme/R type rules to get you started:

$$\frac{A \vdash E : \texttt{int}}{A \vdash \texttt{(int-socket } E\texttt{)} : \texttt{(socketof int)}} \qquad [\textit{int-socket}]$$

$$\frac{A \vdash E : \texttt{int}}{A \vdash \texttt{(bool-socket } E\texttt{)} : \texttt{(socketof bool)}} \qquad [\textit{bool-socket}]$$

$$\frac{A \vdash E : \texttt{int}}{A \vdash \texttt{(unit-socket } E\texttt{)} : \texttt{(socketof unit)}} \qquad [\textit{unit-socket}]$$

$$\frac{A \vdash E : \texttt{int}}{A \vdash \texttt{(symbol-socket } E\texttt{)} : \texttt{(socketof symbol)}} \qquad [\textit{symbol-socket}]$$

$$\frac{A \vdash E_{socket} : \texttt{(socketof T)} \quad A \vdash E_{reader} : \texttt{(-> (T) unit)}}{A \vdash \texttt{(read-all! } E_{socket}\ E_{reader}\texttt{)} : \texttt{unit}} \qquad [\textit{read-all!}]$$

$$\frac{A \vdash E_{socket} : \texttt{(socketof T)} \quad A \vdash E_{datum} : \texttt{T}}{A \vdash \texttt{(write! } E_{socket}\ E_{datum}\texttt{)} : \texttt{unit}} \qquad [\textit{write!}]$$

She has also agreed to write the implementation. Because you are a high-paid 6.821 consultant, your part is to write the type reconstruction algorithm for these constructs.

a. [4 points] Give the type reconstruction algorithm for (int-socket E_{port}).

Solution:

$R[\![\texttt{(int-socket } E\texttt{)}]\!]\ A\ S = $ **let** $\langle T,\ S_1 \rangle = R[\![E]\!]\ A\ S$
$\qquad\qquad\qquad\qquad\qquad$ **in** $\langle \texttt{(socketof int)},\ U(T,\ \texttt{int},\ S_1) \rangle$

b. [4 points] Give the type reconstruction algorithm for (write! E_{socket} E_{datum}).

Solution:

$R[\![(\text{write! } E_{socket} \ E_{datum})]\!] \ A \ S = $ **let** $\langle T_1, \ S_1 \rangle = R[\![E_{socket}]\!] \ A \ S$
in **let** $\langle T_2, \ S_2 \rangle = R[\![E_{datum}]\!] \ A \ S_1$
in **let** $S_3 = U(T_1, (\text{socketof } ?t), S_2)$
in $\langle unit, \ U(T_2, ?t, S_3) \rangle$

c. [4 points] Give the type reconstruction algorithm for (read-all! E_{socket} E_{reader}).

Solution:

$R[\![(\text{read-all! } E_{socket} \ E_{reader})]\!] \ A \ S = $ **let** $\langle T_1, \ S_1 \rangle = R[\![E_{socket}]\!] \ A \ S$
in **let** $\langle T_2, \ S_2 \rangle = R[\![E_{reader}]\!] \ A \ S_1$
in **let** $S_3 = U(T_1, (\text{socketof } ?t), S_2)$
in $\langle unit, \ U(T_2, (\text{-> } (?t) \ \text{unit}), S_3) \rangle$

d. [4 points] As part of Louis's severance agreement, he agreed to stay on for one month to write a proxy server for Prophet.com's intranet (by proxy server we mean something that reads data on one socket and writes it to another). He wrote the following code:

```
(letrec ((proxy (lambda (socket-in socket-out)
                  (read-all! socket-in
                             (lambda (x) (write! socket-out x)))))
         (do-proxy-http (lambda () (proxy (symbol-socket 80)
                                          (symbol-socket 8080))))
         (do-proxy-ftp (lambda () (proxy (int-socket 20)
                                         (int-socket 8020)))))
  (begin
    (do-proxy-http)
    (do-proxy-ftp)))
```

Unfortunately, on his way out on his last day, he gives you the code and tells you it doesn't type check. Give a semantically equivalent (i.e., preserves procedures and procedure calls) replacement for Louis' code that does type check.

Solution: It doesn't type check because in the definition of do-proxy-http, proxy is resolved to have type (-> ((socketof symbol) (socketof symbol)) unit), and thus canont be used polymorphically by (do-proxy-ftp) to have type (-> ((socketof int) (socketof int)) unit).

The following code does not exhibit this problem.

```
(let ((proxy (lambda (socket-in socket-out)
               (read-all! socket-in
                          (lambda (x) (write! socket-out x))))))
  (let (do-proxy-http (lambda () (proxy (symbol-socket 80)
                                        (symbol-socket 8080))))
       (do-proxy-ftp (lambda () (proxy (int-socket 20)
                                       (int-socket 8020)))))
    (begin
      (do-proxy-http)
      (do-proxy-ftp))))
```

e. [4 points] Being the astute 6.821 consultant that you are, you also discover that Louis has used a construct that doesn't have a type reconstruction algorithm in the book: begin. Give the type reconstruction algorithm for (begin E_1 E_2).

Solution:

$R[\![(\texttt{begin } E_1 E_2 ... E_n)]\!] \; A \; S =$ **let** $\langle T_1, \; S_1 \rangle = R[\![E_1]\!] \; A \; S$
                                         in ...

   let $\langle T_n, \; S_n \rangle = R[\![E_n]\!] \; A \; S_{n-1}$
   in $\langle T_n, \; S_n \rangle$

Problem 5: Compiling [20 points]

a. [7 points] What source code generated the following output from the Tortoise compiler?

```
(program
 (define *top*
   (%closure (lambda (.closure8. x) x)))
 (call-closure
  *top*
  (%closure
   (lambda (.closure7. f .k1.)
     (call-closure
      .k1.
      (%closure
       (lambda (.closure6. x .k2.)
         (call-closure
          (%closure-ref .closure6. 1)
          x
          (%closure
           (lambda (.closure5. .t3.)
             (call-closure (%closure-ref .closure5. 1)
                           .t3.
                           (%closure-ref .closure5. 2)))
           (%closure-ref .closure6. 1)
           .k2.)))
      f))))))
```

Solution:

```
(lambda (f) (lambda (x) (f (f x))))
```

b. [7 points] The meaning of $(\text{COND } (P_1 \ E_1) \ (P_2 \ E_2) \ (\text{else } E_3))$ is E_1 if P_1 is true, E_2 is P_1 is false and P_2 is true, and E_3 otherwise.

What is $\mathcal{MCPS}[\![(\text{COND } (P_1 \ E_1) \ (P_2 \ E_2) \ (\text{else } E_3))]\!]$?

Solution:
$\mathcal{MCPS}[\![(\text{COND } (P_1 \ E_1) \ (P_2 \ E_2) \ (\text{else } E_3))]\!]$
$\quad = \lambda m \ .$
$\qquad (\text{let } ((k \ (\text{lambda } (v) \ [m \ v])))$
$\qquad\quad \lambda m \ . \ \mathcal{MCPS}[\![P_1]\!] \ [\lambda v_1 \ .$
$\qquad\qquad\qquad\qquad (\text{IF } v_1$
$\qquad\qquad\qquad\qquad\quad [\mathcal{MCPS}[\![E_1]\!][exp \to meta\text{-}cont \ k]]$
$\qquad\qquad\qquad\qquad\quad [\mathcal{MCPS}[\![P_2]\!][\lambda v_2 \ .$
$\qquad\qquad\qquad\qquad\qquad\qquad (\text{IF } v_2$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad [\mathcal{MCPS}[\![E_2]\!][exp \to meta\text{-}cont \ k]]$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad [\mathcal{MCPS}[\![E_3]\!][exp \to meta\text{-}cont \ k]])]])]$

c. Louis Reasoner has decided to add garbage collection to a language that previously employed explicit storage allocation with MALLOC and FREE opeators. His new implementation ignores FREE and reclaims space using a brand new and correct garbage collector. The garbage collector has more than twice as much heap space as the old explictly managed heap. As soon as this new version of the language is released, several programs that used to run fine – crash!!

(i) [3 points] What is the problem?

Solution: The programs crash because they run out of storage. Pointers to unused storage are not being destroyed, and thus the GC can not reclaim storage that was previosly freeded with FREE.

(ii) [3 points] How can the programmers fix the problems with their programs?

Solution: Zero all pointers to storage that are no longer in use.