MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Compvter Science

6.821 Jeopardy: The Home Version

The game that turns 6.821 into 6.82fun!

This is the home version of the 6.821 Jeopardy game played in class. It includes questions and answers for all categories.

Round 1: Jeopardy

Dynamic Semantics

100 This parameter passing mechanism is indicated by the following transition rules:

$$\frac{\langle E_0, s\rangle \Rightarrow \langle E_0', s'\rangle}{\langle (E_0 \;\; E_1), s\rangle \Rightarrow \langle (E_0' \;\; E_1), s'\rangle}$$

$$\frac{\langle E_1, s\rangle \Rightarrow \langle E_1', s'\rangle}{\langle (V \;\; E_1), s\rangle \Rightarrow \langle (V \;\; E_1'), s'\rangle}$$

$$\langle (\texttt{(lambda (}I\texttt{)}\;\; E_b\texttt{)}\;\; V), s\rangle \Rightarrow \langle [V/I]E_b, s\rangle$$

200 This is the value of the following expression in a dynamically scoped version of FL:

```
(let ((make-sub (lambda (x)
                  (lambda (n) (- n x))))
      (apply (lambda (f x) (f x))))
  (let ((x 1))
    (apply (make-sub 20) 300)))
```

300 In a version of FL! supporting `label` and `jump`, this is the value of the following expression.

```
(let ((r (cell 1)))
  (let ((top (label x x)))
    (if (> (cell-ref r) 10)
        (cell-ref r)
        (begin
          (cell-set! r (* 2 (cell-ref r)))
          (jump top)))))
```

400 In a language with termination semantics for dynamic exceptions, this is the value of:

```
(handle (err (y) (+ y 200))
  (let ((f (lambda (x)
             (+ (raise err x) 1000))))
    (handle (err (z) (+ z 500))
      (f 4))))
```

500 Suppose language L has a direct semantics with the following:

```
Procedure = Denotable* → Store → Result
ℰ : Exp → Environment → Store → Result
```

If the language L is call-by-name, this is the definition of the Denotable domain.

Type Reconstruction

100 Can Hindley-Milner type reconstruction reconstruct a type for the following SCHEME/R expression? Explain. (Give the type or explain why the algorithm cannot reconstruct one.)

```
(lambda (f)
  (let ((g f))
    (if (g #t) (g 1) (g 2))))
```

200 This is the type reconstructed for the following SCHEME/R expression.

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

300 Consider the following definition of the Y operator in SCHEME/R. Is its type reconstructible? Explain.

```
(define y
  (lambda (g)
    (let ((s (lambda (x) (g (x x)))))
      (s s))))
```

400 List all of the following expressions that are reconstructible in SCHEME/R:

```
1. (letrec
       ((id (lambda (a) a))
        (test (lambda ()
                 (if (id #t) (id 1) 2))))
     (test))
2. (letrec ((id (lambda (a) a))
            (test (lambda (y)
                     (if (id #t) y 2))))
    (test (id 1)))
3. (letrec ((id (lambda (a) a))
            (test (lambda (x y)
                     (if x y 2))))
    (test (id #t) (id 1)))
```

500
This type schema is bound to x in the body of the following SCHEME/R expression.

```
(letrec ((x (lambda () (x))))
  body)
```

Pragmatics

100 This compiler pass must precede closure conversion with flat environments, but need not precede closure conversion with nested environments.

200 After CPS conversion, this is the syntactic form of the continuation for a tail call in the source program.

300 Suppose a language has the construct `(value` E`)` that evaluates E to a symbol, and returns the value bound to that symbol. E.g.,

```
(let ((x (symbol a))
      (a 3))
  (value x))
; Value = 3
```

What change in the compiler would have to be made to environment representations in order to accomodate this construct?

400 If a SCHEME/R program type checks, will the program resulting from CPS conversion also type check? Explain.

500 What additional run-time support is needed for SCHEME/XSP's `plambda` expression? Be specific.

Program Translations

100 Under this variable scoping mechanism for FL, the following desugaring is *invalid*.

`(lambda (`I_1 I_2`)` E`)`
$\Rightarrow$
`(lambda (`I_1`) (lambda (`I_2`)` E`))`

(Assume all applications of the procedure are appropriately modified.)

200 Is the following a safe (i.e., semantics-preserving) transformation in a strictly functional language? Explain.

`(if` E_1 E_1 E_2`)`
$\Rightarrow$
`(let ((`I_1 E_1`))` `(if` I_1 I_1 E_2`))`

300 Is the following desugaring valid in SCHEME/R? Explain.

`(let ((`I_1 E_1`))` E_2`)`
$\Rightarrow$
`((lambda (`I_1`)` E_2`)` E_1`)`

400 This problem is encountered in using the following two rules in a simplifier for PostFix+{`dup`}.

```
(Q) . exec . S  ==>  Q @ S

V . dup . S     ==>  V . V . S
```

500 Is the following a valid transformation in every functional language? Explain.

`(let ((`I_1 E_1`))` E_2`)` $\Rightarrow$ $[E_1/I_1]E_2$

Trivia

100 This OODL (Object Oriented Dynamic Langauge), once backed by Apple Computer, is noted for its sophisticated run-time memory management and functional style, including first class and anonymous functions.

200 This is widely recognized as the first object-oriented language.

300 Alan Perlis says that syntactic sugar causes this.

400 This language designer once quipped that he could be called both by name and by value.

500 This one of the following is *not* the title or subtitle of a Steele & Sussman Scheme paper.

a. Lambda the Ultimate Declarative

b. Lambda the Ultimate GOTO

c. Lambda the Ultimate Imperative

d. Lambda the Ultimate Objective

e. Lambda the Ultimate Opcode

Round 2: Double Jeopardy

Semantics Fundamentals

200 Suppose domain A has 4 elements and domain B has 3 elements. There are this many set-theoretic functions from A to B.

400 Suppose Bool $=$ $\{true, false\}$. There are this many elements in the domain:

$$(\text{Bool}_\perp \times (\text{Bool}_\perp + \text{Bool}_\perp))_\perp$$

600 What is wrong with the following denotational semantics for a language that supports recursion?

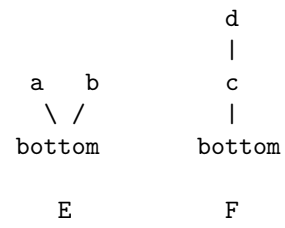
$e \in \texttt{Environment} = \text{Identifier} \to (\texttt{Expressible} + \texttt{Unbound})$
extend-env $: \texttt{Environment} \to \text{Identifier} \to \texttt{Expressible} \to \texttt{Environment}$
$\mathcal{E} : \text{Exp} \to \texttt{Environment} \to (\texttt{Expressible} + \texttt{Error})$

$\mathcal{E}[\![\texttt{(rec } I \texttt{ } E\texttt{)}]\!] =$
$\lambda e_0 \,.\, (\mathcal{E}[\![E]\!] \ (\mathbf{fix}_{\text{Environment}}(\lambda e_1 \,.\, (\textit{extend-env} \ \ e_0 \ \ I \ \ (\mathcal{E}[\![E]\!] \ \ e_1)))))$

800 Suppose $b \in \text{Bool}_\perp$, $f \in \text{Bool}_\perp \to \text{Bool}_\perp$, and *or* is strict boolean disjunction. The following functional has this many fixed points.

$$\lambda f \,.\, \lambda b \,.\, (or \ \ b \ \ (f \ \ b))$$

1000 Consider the following domains E and F:



There are this many monotonic functions from E to F.

Memory Management

200 Could dangling references to freed memory ever cause a program to run out of memory? Explain.

400 In implementations of dynamically-typed languages, tag bits are often used to encode the types of run-time objects. Even though SCHEME/X is statically typed, every word of memory would still need one tag bit. Why?

600 This many words of memory (including header words) are required to represent the following value in the TORTOISE compiler.

```
(cons 1 (cons 2 (null)))
```

800 Write a SCHEME expression that generates a run-time structure that is not collectible by a reference-counting garbage collector.

1000 Below is the non-zero portion of the lower semispace of a 40-word memory. We represent a tagged value as *integer, tag bit(s)*. Show the non-zero portion of the upper semispace (starting at address 20) of memory after a stop-and-copy garbage collection is performed with a register containing `4 , 01`. We have omitted the type field in the header word, so you should assume that all words in the block must be scanned. Recall that `0` tags an integer (or header word), and `01` tags a pointer.

```
0 : 3, 0        6 : 2, 0
1 : 4, 0        7 : 9, 01
2 : 7, 01       8 : 0, 01
3 : 6, 0        9 : 2, 0
4 : 1, 0        10: 4, 01
5 : 9, 01       11: 9, 0
```

Types

200 Consider the set of all syntactically legal SCHEME/R expressions that contain no free variables. This is the shortest (in terms of fewest characters) expression in this set that is not well-typed.

400 Does the following SCHEME/XSP expression type check? Explain.

```
(plambda (int)
  (lambda ((x int))
    (+ x 5)))
```

600 There are this many distinct values with the following type (assume that our language does not have side-effects or divergence).

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

800 This is the SCHEME/XSP type checking rule for applications with one argument: `(`E_0 E_1`)`

1000 This is the typing rule in SCHEME/R for `letrec` with a single binding:

`(letrec ((`I_1 E_1`))` E_B`)`

(Recall that $Gen(T, A)$ appropriately forms a type schema given type T and type environment A.)

Programming Paradigms

200 How can a run-time lock system dynamically check for deadlock?

400 Recall that all PostFix programs terminate. Is it possible to write a PostFix command sequence that computes the absolute value of a number at the top of the stack? Explain.

600 What is the value of the following object-oriented programming system expression:

```
(let ((ob2 (object (method value (self) 2)))
      (ob3 (object (method value (self) 3))))
  (let ((ob5 (object
              (method value (self) 5)
              (method compute (self)
                (send * (send value ob3)
                        (send value self))))))
    (send compute (object ob2 ob3 ob5))))
```

800 Let S and F be the success and failure continuations. Fill in the desugaring for the pattern matching operators.

$\mathcal{D}_{\text{clauseseq}}[\![[], [], V, F]\!] =$ **??**
$\mathcal{D}_{\text{pat}}[\![\_, V, S, F]\!] =$ **??**
$\mathcal{D}_{\text{patseq}}[\![\_, [], V, F]\!] =$ **??**

1000 List all possible values for the following control-parallel expression:

```
(let ((x (cell 3)))
  (let ((thread (fork (cell-set! x
                       (+ (cell-ref x)
                          (cell-ref x))))))
    (begin (cell-set! x 7)
           (join thread)
           (cell-ref x))))
```

Potpourri

200 In a purely functional programming language, can the meaning of an expression under call-by-value semantics differ from the meaning of the same expression under call-by-name semantics? Explain.

400 What problem arises if we add the following procedure to a language with references?

```
free : (poly (t)
         (-> ((ref-of t)) unit))
```

Given a reference value, `free` deallocates the storage occupied by the value pointed to by the reference so that it can be reused.

600 The terms "bug" and "compile" were coined by this Navy rear admiral who designed COBOL.

800 Name the (1) parameter-passing mechanism and (2) variable scoping mechanism implied by the following domain definitions and signatures:

```
Procedure = Denotable* → Environment → Expcont → Answer
Denotable = Expcont → Answer
Expcont = Expressible → Answer
ℰ : Exp → Environment → Expcont → Answer
```

1000 Suppose `(pairof` T_1 T_2`)` is a type constructor for heterogeneous pairs. Give a type T that makes the following two types equivalent:

a. `(recof s (pairof int (pairof bool s)))`

b. `(pairof int` T`)`

Round 3: Final Jeopardy

Types

What is the reconstructed type for the following SCHEME/R expression?

```
(letrec
    ((accumulate
       (lambda (combiner seed lst)
         (if (null? lst)
             seed
             (combiner
               (car lst)
               (accumulate combiner
                           seed
                           (cdr lst)))))))
  accumulate)
```

Answers

Round 1: Jeopardy

Dynamic Semantics

100 What is call-by-value?

200 What is 0?

300 What is 16?

400 What is 504?

500 What is Denotable = Store → Result?

Type Reconstruction

100 No — the expression uses first-class polymorphism.

200 `(-> ((-> (?b) ?c) (-> (?a) ?b)) (-> (?a) ?c))`

300 No — self application of a non-generic variable `x` fails (fails in occurs check).

400 Only number 3 is reconstructible; in the other cases, `id` is constrained by the fact that the `letrec`-bound variables aren't generic within the right-hand-side expressions.

500 What is `(generic (t) (-> () t))`?

Pragmatics

100 What is Assignment Conversion?

200 What is an identifier?

300 Environments must hold names as well as values.

400 Yes. Basically, each continutation is only used once to return the value of a procedure to the rest of the computation. Thus all the continuations have type `(-> (T1) T2)` where `T1` is the return type of the procedure the continuation is used with and `T2` is the result of the entire program.

500 None. `plambda` expressions have no run-time aspect — they are only used during type checking and thus the value of a `plambda` is just the value of its body.

Program Translations

100 What is dynamic scoping? References to I_1 within E will not be handled correctly.

200 No; Name capture of I can occur in E_2.

300 No. *E* could be polymorphic in the first expression, but the desugaring would require first-class polymorphism to support the same semantics.

400 What is simplifier non-termination?

500 In call-by-value, doesn't preserve termination. E.g

```
(let ((x ((lambda (y) (y y)) (lambda (y) (y y)))))
  3)
```

Trivia

100 What is Dylan?

200 What is Simula 67?

300 What is cancer of the semi-colon?

400 Who is Niklaus Wirth? His name is pronounceable both as "Vert" and "Worth". Some people go further and call him "Nickle's-worth."

500 What is "Lambda the Ultimate Objective"?

Round 2: Double Jeopardy

Semantics Fundamentals

200 What is 81? $|B|^{|A|} = 3^4 = 81$.

400 What is 19?

600 The domain Environment isn't pointed, and **fix** is only well defined over pointed CPOs.

800 What is 6? Since *or* is strict, *f* must map $\perp$ to $\perp$. Since *or* is disjunction, it consistent for *f* to map *true* to either $\perp$ or *true*, and to map *false* to any element of Bool. Since there are two independent choices for *true*, and three for *false*, there are six possible fixed points.

1000 What is 14? Here are the 14 possibilities:

9 if $E_\perp \Rightarrow F_\perp$, then each of `a` and `b` can map to all 3 elts of F

4 if $E_\perp \Rightarrow$ `c`, then each of `a` and `b` can map to `c` and `d`

1 if $E_\perp \Rightarrow$ `d`, then each of `a` and `b` must map to `d`

Memory Management

200 Yes. In a stop-and-copy garbage collector, all memory pointed to by pointers accessible from the root set (including the dangling reference) would be copied over, despite the fact the program explictly freed it. So the memory pointed to by a dangling reference is never actually released by the garbage collector.

400 The single tag bit is used by the garbage collector, not the typing system. It is crucial for distinguishing pointers from non-pointers.

600 What is 6? A cons cell has one header word and two field words. The cdr of the first cons cell is a pointer to a second cons cell, which has a header word and two immediate values in its car and cdr. `(null)` is represented by immediate integer `0`.

800 Reference counting garbage collectors don't collect structures with cycles, so we need to construct a cyclic structure. The easiest way to do this in SCHEME is with cells; for example, the following expression returns a reference cell that points to itself.

```
(let ((a (cell 0)))
  (begin (cell-set! a a)
         a))
```

Even if SCHEME did not support side effects, it would still be possible to create cyclic runtime structures via `letrec`, since a procedures created in a letrec binding has a pointer to itself through the environment. Therefore, another cycle-creating expression is:

```
(letrec ((f (lambda () (f))))
  f)
```

In fact, *any* procedure created by `letrec`, should work, e.g.:

```
(letrec ((g (lambda () 3)))
  g)
```

However, since compiler optimizations might remove the cyclic dependencies in a situation like `g`, it's safer to stick with a truly recursive function like `f`.

1000 The non-garbage in the given memory consists of a block A of size 1 that points to a block B of size 2 whose first slot points to A and whose second slot contains an immediate 9. These two blocks get copied into the first five words of the second semispace as shown below. Note that the second semispace begins at location 20.

```
...
20: 1, 0
21: 22, 1
22: 2, 0
23: 20, 1
24: 9, 0
...
```

Types

200 What is `(1)`? (Similar variants have the same length.)

400 It is not well-typed. `+` expects two arguments of base type `int`, but `x` is of some polymorphic type, `int`. The typing rule for `plambda` prohibits the `plambda`-bound identifier from clashing with names that appear in the types of free variables within the body (in this case, `+` uses the base type `int`).

600 What is two? The two are the polymorphic function of three arguments of the same type that returns true, and a similar function that returns false. The result can't possibly depend on the three arguments because there are no polymorphic predicate or comparison operators.

800 What is

$$\frac{\begin{array}{c} A \vdash E_0 : T_1 \ \rightarrow \ T_2 \\ A \vdash E_1 : T_1' \\ T_1' \sqsubseteq T_1 \end{array}}{A \vdash \texttt{(}E_0 \ \ E_1\texttt{)} : T_2} \qquad [\textit{application}]$$

1000 What is

$$\frac{\begin{array}{c} A[I_1 : T_1] \vdash E_1 : T_1 \\ A[I_1 : \mathit{Gen}(T_1\texttt{, A})] \vdash E_B : T_B \end{array}}{A \vdash \texttt{(letrec ((}I_1 \ \ E_1\texttt{)) } E_B\texttt{)} : T_B} \qquad [\textit{letrec}]$$

Programming Paradigms

200 By constructing a dependancy graph and looking for cycles. Consider each process a node, and insert a directed edge from P_1 to P_2 iff P_2 holds a lock P_1 is trying to aquire. A cycle indicates deadlock.

400 No. Computing the absolute value requires two references to the number: comparing it to zero and possibly negating it. Without `dup` (or some means of naming a value), PostFix is unable to make more than one reference to any value.

600 The value is 6. Even though `ob5` responds to the `compute` message, its variable `self` is bound at that point to the object composed of `ob2`, `ob3`, and `ob5`; a `value` message sent to this object is handled by `ob2`, returning 2. A `value` message sent directly to `ob3` returns 3.

800 $\mathcal{D}_{\text{clauseseq}}[\![[], [], V, F]\!] = \texttt{(}F\texttt{)}$
$\mathcal{D}_{\text{pat}}[\![\_, V, S, F]\!] = S$
$\mathcal{D}_{\text{patseq}}[\![\_, [], V, F]\!] = S$

1000 6, 7, 10, and 14.

Potpourri

200 Yes, it affects termination. Example:

```
((lambda (a) 3)
 ((lambda (x) (x x))
  (lambda (x) (x x))))
```

400 Dangling pointers can result from a use of `free`.

600 Grace Murray Hopper.

800 The domains are from a standard semantics for a functional programming language. (1) Call-by-name (in call-by-value, Denotable would be Expressible) (2) Dynamic scoping (because elements of the Procedure domain take an environment).

1000 The `recof` types are equivalent if their infinite expansions are equal. The first type denotes an infinite list of alternating `int` and `bool`, so T must denote an infinite list of alternating `bool` and `int`.

$$T = \texttt{(recof t (pairof bool (pairof int t)))}$$

Round 3: Final Jeopardy

```
(-> ((-> (?s ?t) ?t) ?t (list-of ?s)) ?t)
```