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:

\[
\begin{align*}
\langle E_0, s \rangle &\Rightarrow \langle E'_0, s' \rangle \\
\langle (E_0 \ E_1), s \rangle &\Rightarrow \langle (E'_0 \ E'_1), s' \rangle \\
\langle E_1, s \rangle &\Rightarrow \langle E'_1, s' \rangle \\
\langle (V \ E_1), s \rangle &\Rightarrow \langle (V \ E'_1), s' \rangle \\
\langle ((\lambda (I) \ E_0) \ V), s \rangle &\Rightarrow \langle [V/I] E_0, s \rangle
\end{align*}
\]

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

\[
\begin{align*}
(\text{let} ((\text{make-sub} \ (\lambda (x) \ (\lambda (n) (- n x)))) \\
(\text{apply} \ (\lambda (f \ x) \ (f \ x))) \\
(\text{let} ((x \ 1)) \\
(\text{apply} \ (\text{make-sub} \ 20) \ 300)))
\end{align*}
\]

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

\[
\begin{align*}
(\text{let} ((r \ \langle \text{cell} \ 1 \rangle)) \\
(\text{let} ((\text{top} \ (\text{label} \ x \ x))) \\
(\text{if} \ (> \ (\text{cell-ref} \ r) \ 10) \\
(\text{cell-ref} \ r) \\
(\text{begin} \\
(\text{cell-set!} \ r \ (* \ 2 \ (\text{cell-ref} \ r))) \\
(\text{jump} \ \text{top})))))
\end{align*}
\]

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

\[
\begin{align*}
(\text{handle} \ (\text{err} \ (y) \ (+ \ y \ 200)) \\
(\text{let} ((f \ (\lambda (x) \ (+ \ (\text{raise} \ \text{err} \ x) \ 1000)))) \\
(\text{handle} \ (\text{err} \ (z) \ (+ \ z \ 500)) \\
(f \ 4)))
\end{align*}
\]

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

\[
\begin{align*}
\text{Procedure} &\rightarrow \text{Denotable*} \rightarrow \text{Store} \rightarrow \text{Result} \\
\mathcal{E} &\rightarrow \text{Exp} \rightarrow \text{Environment} \rightarrow \text{Store} \rightarrow \text{Result}
\end{align*}
\]
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**
This compiler pass must precede closure conversion with flat environments, but need not precede closure conversion with nested environments.

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

Suppose a language has the construct \( \textit{value} \ E \) that evaluates \( E \) to a symbol, and returns the value bound to that symbol. E.g.,

\[
\begin{align*}
\text{(let ((x (symbol a)) )} \\
\phantom{\text{(let ((x (symbol a)))}} \ (a 3) \\
\phantom{\text{(let ((x (symbol a)))}} \ (\text{value} \ x)) \\
; \text{Value} = 3
\end{align*}
\]

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

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

What additional run-time support is needed for \textit{SCHEME}/XSP’s \textit{plambda} expression? Be specific.

**Program Translations**

Under this variable scoping mechanism for FL, the following desugaring is \textit{invalid}.

\[
\begin{align*}
\text{(lambda} \ (I_1 \ I_2) \ E) \\
\Rightarrow \\
\text{(lambda} \ (I_1) \ (\text{lambda} \ (I_2) \ E))
\end{align*}
\]

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

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

\[
\begin{align*}
\text{(if} \ E_1 \ E_1 \ E_2) \\
\Rightarrow \\
\text{(let} \ ((I_1 \ E_1)) \ (\text{if} \ I_1 \ I_1 \ E_2))
\end{align*}
\]

Is the following desugaring valid in \textit{SCHEME}/R? Explain.

\[
\begin{align*}
\text{(let} \ ((I_1 \ E_1)) \ E_2) \\
\Rightarrow \\
((\text{lambda} \ (I_1) \ E_2) \ E_1)
\end{align*}
\]

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

\[
\begin{align*}
\text{(Q)} \ . \ \text{exec} \ . \ S \quad \Rightarrow \quad Q \ \emptyset \ S \\
\text{V} \ . \ \text{dup} \ . \ S \quad \Rightarrow \quad V \ . \ V \ . \ S
\end{align*}
\]

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

\[
\begin{align*}
\text{(let} \ ((I_1 \ E_1)) \ E_2) \ \Rightarrow \ [E_1/I_1]E_2
\end{align*}
\]
**Trivia**

100 This OODL (Object Oriented Dynamic Language), 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 $\text{Bool} = \{true, false\}$. There are this many elements in the domain:

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

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

$$e \in \text{Environment} = \text{Identifier} \rightarrow (\text{Expressible + Unbound})$$

$$\text{extend-env} : \text{Environment} \rightarrow \text{Identifier} \rightarrow \text{Expressible} \rightarrow \text{Environment}$$

$$\mathcal{E} : \text{Exp} \rightarrow \text{Environment} \rightarrow (\text{Expressible + Error})$$

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

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

$$\lambda f. \lambda b. (\text{or } b (f b))$$

1000 Consider the following domains $E$ and $F$:

```
d |
 a b c
 \ / |
 bottom bottom
```

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

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.

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Data 1</th>
<th>Data 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3, 0</td>
<td>6, 2, 0</td>
</tr>
<tr>
<td>1</td>
<td>4, 0</td>
<td>7, 9, 01</td>
</tr>
<tr>
<td>2</td>
<td>7, 01</td>
<td>8, 0, 01</td>
</tr>
<tr>
<td>3</td>
<td>6, 0</td>
<td>9, 2, 0</td>
</tr>
<tr>
<td>4</td>
<td>1, 0</td>
<td>10, 4, 01</td>
</tr>
<tr>
<td>5</td>
<td>9, 01</td>
<td>11, 9, 0</td>
</tr>
</tbody>
</table>
```

Types

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.

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

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

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))
```

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

This is the typing rule in SCHEME/R for \texttt{letrec} with a single binding:

\[
(letrec ((I_1 \ E_1)) \ E_B)
\]

(Recall that \texttt{Gen}(T, A) appropriately forms a type schema given type \(T\) and type environment \(A\).)

Programming Paradigms

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

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.

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

\[
\text{(cons 1 (cons 2 (null)))}
\]
(let ((ob2 (object (method value (self) 2)))
       (ob3 (object (method value (self) 3)))
       (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.

\[ D_{\text{clauseseq}}[[\_], \_], V, F] = ?? \]
\[ D_{\text{pat}}[\_ V, S, F] = ?? \]
\[ D_{\text{patseq}}[\_ \_], V, F] = ?? \]

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?

\[ \text{free} : (\text{poly} \ (t) \to ((\text{ref-of} \ t)) \text{unit}) \]

Given a reference value, \text{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:

\[ \text{Procedure} = \text{Denotable}^* \to \text{Environment} \to \text{Expcont} \to \text{Answer} \]
\[ \text{Denotable} = \text{Expcont} \to \text{Answer} \]
\[ \text{Expcont} = \text{Expressible} \to \text{Answer} \]
\[ \mathcal{E} : \text{Exp} \to \text{Environment} \to \text{Expcont} \to \text{Answer} \]

1000 Suppose \((\text{pairof} \ T_1 \ T_2)\) 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?

```scheme
(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  \((-\rightarrow (\rightarrow (?b) ?c) (\rightarrow (?a) ?b)) (\rightarrow (?a) ?c)\)

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 \texttt{letrec} bound variables aren’t generic within the right-hand-side expressions.

500  What is \((\text{generic } t) (\rightarrow () t))\)?

Pragmatics

100  What is Assignment Conversion?

200  What is an identifier?

300  Environments must hold names as well as values.

400  Yes. Basically, each continuation is only used once to return the value of a procedure to the rest of the computation. Thus all the continuations have type \((-\rightarrow (T1) T2)\) 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. \texttt{plambda} expressions have no run-time aspect — they are only used during type checking and thus the value of a \texttt{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 \(E_2\).
No. $E$ could be polymorphic in the first expression, but the desugaring would require first-class polymorphism to support the same semantics.

What is simplifier non-termination?

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

$$(\text{let } ((x ((\text{lambda } (y) (y y)) \text{ (lambda } (y) (y y))))) 3)$$

Trivia

What is Dylan?

What is Simula 67?

What is cancer of the semi-colon?

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

What is “Lambda the Ultimate Objective”?

Round 2: Double Jeopardy

Semantics Fundamentals

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

What is 19?

The domain Environment isn’t pointed, and $\text{fix}$ is only well defined over pointed CPOs.

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

What is 14? Here are the 14 possibilities:

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

4 if $E_{\bot} \Rightarrow c$, then each of $a$ and $b$ can map to $c$ and $d$

1 if $E_{\bot} \Rightarrow d$, then each of $a$ and $b$ must map to $d$

Memory Management

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 explicitly freed it. So the memory pointed to by a dangling reference is never actually released by the garbage collector.
The single tag bit is used by the garbage collector, not the typing system. It is crucial for distinguishing pointers from non-pointers.

What is 6? A cons 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.

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.

```scheme
(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:

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

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

```scheme
(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`.

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

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

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`).
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.

What is

\[
\begin{align*}
A \vdash E_0 : T_1 \rightarrow T_2 \\
A \vdash E_1 : T_1' \\
T_1' \leq T_1
\end{align*}
\]

\[
A \vdash (E_0 \ E_1) : T_2
\]

[application]

What is

\[
\begin{align*}
A[I_1 : T_1] \vdash E_1 : T_1 \\
A[I_1 : \text{Gen}(T_1, A)] \vdash E_B : T_B
\end{align*}
\]

\[
A \vdash (\text{letrec} ((I_1 \ E_1)) E_B) : T_B
\]

[letrec]

### Programming Paradigms

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 acquire. A cycle indicates deadlock.

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

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

6, 7, 10, and 14.

### Potpourri

Yes, it affects termination. Example:

\[
\begin{align*}
((\lambda a \ 3) \\
((\lambda x \ (x \ x)) \\
(\lambda x \ (x \ x)))
\end{align*}
\]

Dangling pointers can result from a use of \texttt{free}.

Grace Murray Hopper.

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).
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.

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

Round 3: Final Jeopardy

$$(-\rightarrow ((-\rightarrow (?s \ ?t) \ ?t) \ ?t (\text{list-of } ?s)) \ ?t)$$