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. Since we would like to reuse many of these questions in future versions of the game, we request that you don’t give this handout to students taking this course in later terms. Here’s a special note directed at any current or future 6.821 student who happens to have obtained a copy of this handout: *Put this document down immediately, and don’t look at it until after you’ve played the game!*
**Dynamic Semantics**

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

\[<E_0, s> \rightarrow <E'_0, s'>\]

\[<(E_0 E_1), s> \rightarrow <(E'_0 E_1), s'>\]

\[<E_1, s> \rightarrow <E'_1, s'>\]

\[<(V E_1), s> \rightarrow <(V E'_1), s'>\]

\[<((\text{abs (I) } E_b), v), s> \rightarrow <[v/I] E_b, s>\]

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

\[
\text{(let ((make-sub (abs (x) (abs (n) (- n x)))}
  (let ((x 1))
    (app (make-sub 20) 300)))
\]

300 In a version of FLIC supporting **label** and **jump**, what is the value of the following expression?

\[
\text{(let ((r (cell 1)))}
  (let ((top (label x x)))
    (if (> (^ r) 10)
      (^ r)
      (begin
        (:= r (* 2 (^ r)))
        (jump top top)))
\]

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

\[
\text{(handle err (abs (y) (+ y 200))}
  (let ((f (abs (x))
      (+ (raise err x) 1000)))
    (handle err (abs (z) (+ z 500))
      (f 4)))
\]

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

\[
\text{Proc} = \text{Nameable}^* \rightarrow \text{Store} \rightarrow \text{Result}
\]

\[
\mathcal{E}: \text{Exp} \rightarrow \text{Env} \rightarrow \text{Store} \rightarrow \text{Result}
\]
If the language L is call-by-name, what is the definition of the Nameable domain?

**Type Reconstruction**

**100** Can Hindley-Milner type reconstruction reconstruct a type for the following FLARE expression? Explain.

```scheme
(abs (f)
   (let ((g f))
     (if (g #t) (g 1) (g 2))))
```

**200** What is the type reconstructed for the following FLARE expression?

```scheme
(abs (g f)
   (abs (x) (g (f x))))
```

**300** Consider the following definition of the Y operator in FLARE. Is its type reconstructible? Explain.

```scheme
(define y
  (abs (g)
     (let ((s (abs (x) (g (x x)))))
       (s s)))
```

**400** List all of the following expressions that are reconstructible in FLARE:

1. (letrec
   ((id (abs (a) a))
    (test (abs ()
           (if (id #t) (id 1) 2)))))
   (test))

2. (letrec ((id (abs (a) a))
    (test (abs (y)
            (if (id #t) y 2)))))
   (test (id 1)))

3. (letrec ((id (abs (a) a))
    (test (abs (x y)
             (if x y 2)))))
   (test (id #t) (id 1)))
500
What type schema is bound to x in the body of the following FLARE expression?

\[
\text{(letrec } ((x (abs () (x))) \text{)}\text{body})
\]

Pragmatics

100 What compiler pass must precede closure conversion with flat environments but need not precede closure conversion with nested environments?

200 After CPS conversion, what can you say about the syntactic form of the continuation for a tail call in the source program?

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

\[
\text{(let } ((x (symbol a)))
  \text{(a 3))}
\text{(value} x))
; Value = 3
\]

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

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

500 What additional run-time support is needed for FLEX/SP’s \text{pabs}\ expression? Be specific.

Program Translations

100 Under what variable scoping mechanism for FL is the following desugaring invalid?

\[
\text{(abs } (I1 I2) E) => (abs } (I1) (\text{abs } (I2) E))
\]

(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.

\[
\text{(if } E1 E1 E2) => \text{(let } ((I1 E1)) \text{(if } I1 I1 E2))
\]

300 Is the following desugaring valid in FLARE? Explain.
(let ((I1 E1)) E2) => ((abs (I1) E2) E1)

400 What problem is encountered in using the following two rules in a simplifier for POSTFix+{dup}?

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

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

(let ((I1 E1)) E2) => [E1/I1] E2

Trivia

100 This OODL (Object Oriented Dynamic Language), 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 Which of the following is not the title or subtitle of a Steele & Sussman Scheme paper?

a. Lambda the Ultimate Imperative
b. Lambda the Ultimate Declarative
c. Lambda the Ultimate Objective
d. Lambda the Ultimate GOTO
e. Lambda the Ultimate Opcode
Round 2: Double Jeopardy

Semantics Fundamentals

200 Suppose domain \( A \) has 4 elements and domain \( B \) has 3 elements. How many set-theoretic functions are there from \( A \) to \( B \)?

400 Suppose \( \text{Bool} = \{\text{true}, \text{false}\} \). How many elements are there 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?

\[
\begin{align*}
\epsilon & \in \text{Env} = \text{Ident} \rightarrow (\text{Expressible} + \text{Unbound}) \\
\text{extend-env} & : \text{Env} \rightarrow \text{Ident} \rightarrow \text{Expressible} \rightarrow \text{Env} \\
\mathcal{E} & : \text{Exp} \rightarrow \text{Env} \rightarrow (\text{Expressible} + \text{Error})
\end{align*}
\]

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

800 Suppose \( b \in \text{Bool}_\bot, f \in \text{Bool}_\bot \rightarrow \text{Bool}_\bot \), and \( \text{or} \) is strict boolean disjunction. How many fixed points does the following functional have?

\[
\lambda f . \lambda b . (\text{or } b \ (f \ b))
\]

1000 Consider the following domains \( E \) and \( F \)

| d | \( | \) |
|---|---|
| a | b | c |
| \( / \) | \( | \) |
| bottom | bottom |

How many monotonic functions are there from \( E \) to \( F \)?

Memory Management

200 Could dangling references 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 FLEX is statically typed, every word of memory would still need one tag bit. Why?
How many words of memory are required to represent the following value in the TORTOISE compiler? Include header words.

(cons 1 (cons 2 (null)))

Write a FL 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 memory (word addresses 0 – 19). We represent a tagged value as integer, tag bit(s). Show the non-zero portion of the upper semispace of memory after a stop-and-copy garbage collection is performed with the register labelled ROOT as root. We have omitted the type field in the header word, so you should assume that all words in the block must be scanned.

ROOT: 4 , 01

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

Types

Consider the set of all syntactically legal FLARE expressions that contain no free variables. What is the shortest (in terms of fewest characters) expression in this set that is not well-typed?

Does the following FLEX/SP expression type check? Explain.

(pabs (int)
  (abs ((x int))
    (+ x 5)))

How many distinct values are there with the following type (assume that our language does not have side-effects or divergence)?

(forall (t) (-> (t t t) bool))

Give the FLEX/SP type checking rule for applications with one argument: \((E_0 \ E_1)\)

Give the typing rule in FLARE for letrec with a single binding where \(E_h\) is pure:
\[ (\text{letrec } ((I, \, E_i)) \, E_B) \]

Recall that \( \text{Gen}_{\text{Pure}}(E, \, T, \, A) \) appropriately forms a type schema given type \( T \) and type environment \( TE \).

### 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** Desugar \((\text{cobegin} \, E_1 \ldots \, E_n)\) into \((\text{fork} \, E)\) and \((\text{join} \, E)\).

**800** What is the value of the following OOPS expression:

\[
(\text{let } ((\text{ob2} \, (\text{object} \, \text{method} \, \text{value} \, \text{(self)} \, 2)))
(\text{ob3} \, (\text{object} \, \text{method} \, \text{value} \, \text{(self)} \, 3))))
(\text{let } ((\text{ob5} \, (\text{object} \\
\quad \text{method} \, \text{value} \, \text{(self)} \, 5) \\
\quad \text{method} \, \text{compute} \, \text{(self)} \\
\quad \text{(send} \, * \, \text{(send} \, \text{value} \, \text{ob3} \\
\quad \text{(send} \, \text{value} \, \text{self}))))))))
(\text{send} \, \text{compute} \, \text{(object} \, \text{ob2} \, \text{ob3} \, \text{ob5}))))
\]

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

\[
(\text{let } ((x \, (\text{cell} \, 3)))
(\text{let } ((\text{thread} \, (\text{fork} \, :\!= \, x \\
\quad (+ \, (* \, x) \\
\quad (* \, x)))))) \\
(\text{begin} \, :\!= \, x \, 7 \\
(\text{join} \, \text{thread} \\
(* \, x)))))
\]

### 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?

\[
\text{free} : (\text{forall} \, (t) \\
\quad (\rightarrow \, ((\text{ref-of} \, t)) \, \text{unit}))
\]

8
Given a reference value, \texttt{free} deallocates the storage occupied by the value pointed to by the reference so that it can be reused.

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

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

\begin{align*}
\text{Proc} &= \text{Nameable}^* \to \text{Env} \to \text{Expcont} \to \text{Answer} \\
\text{Nameable} &= \text{Expcont} \to \text{Answer} \\
\text{Expcont} &= \text{Expressible} \to \text{Answer} \\
\mathcal{E} : \text{Exp} &\to \text{Env} \to \text{Expcont} \to \text{Answer}
\end{align*}

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. \((\text{recof } s \ (\text{pairof } \text{int} \ (\text{pairof } \text{bool} \ s)))\)

b. \((\text{pairof } \text{int} \ T)\)
Round 3: Final Jeopardy

Types

What is the reconstructed type for the following FLARE expression?

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

accumulate)
Answers

Round 1: Jeopardy

Dynamic Semantics

100 Call-by-value

200 0

300 16

400 504

500 Nameable = 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. (This problem is based on a comment made by Nate Osgood.)

500

(generic (t) (-> () t))

Pragmatics

100 Assignment Conversion.

200 It must be 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. pabs expressions have no run-time aspect — they are only used during type checking and thus the value of a pabs is just the value of its body.

Program Translations
Dynamic scoping — references to $I_1$ within $E$ will not be handled correctly.

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

No. $E_1$ could be polymorphic in the first expression, but the desugaring would require first-class polymorphism to support the same semantics.

The simplifier may not terminate.

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

\[
\text{(let ((x ((abs (y) (y y)) (abs (y) (y y)))) 3)}
\]

Trivia

Dylan

Simula 67

Cancer of the semi-colon

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

Lambda the Ultimate Objective

Round 2: Double Jeopardy

Semantics Fundamentals

\[ |B|^{|A|} = 3^4 = 81 \]

19

The domain Environment isn’t pointed, and FIX is only well defined over pointed CPOs.

6. Since or is strict, $f$ must map $\bot$ to $\bot$. Since or is disjunction, it consistent for $f$ to map true to either $\bot$ 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.

Here’s why:

* if E_bot -> F_bot, each of a and b can map to all 3 els of f => 9
* if E_bot -> c, each of a and b can only map to c and d => 4
* if E_bot -> d, each of a and b *must* map to d => 1

---

14
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.

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.

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 FL is with cells; for example, the following expression returns a reference cell that points to itself.

```lisp
(let ((a (cell 0)))
  (begin (:= a a)
    a))
```

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

```lisp
(letrec ((f (abs () (f))))
  f)
```

In fact, \textit{any} procedure created by \texttt{letrec}, should work, e.g.:

```lisp
(letrec ((g (abs () 3)))
  g)
```

However, since compiler optimizations might remove the cyclic dependencies in a situation like \texttt{g}, it’s safer to stick with a truly recursive function like \texttt{f}.

The non-garbage in the given memory consists of a block \texttt{A} of size 1 that points to a block \texttt{B} of size 2 whose first slot points to \texttt{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, 01
22: 2, 0
23: 20, 01
24: 9, 0
...```
Types

It is not well-typed. + expects two arguments of base type int, but x is of some polymorphic type, int. The typing rule for pabs prohibits the pabs-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).

Two, 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.

\[
\begin{align*}
A \vdash E_0 : (\rightarrow (T_1) T_2) \\
A \vdash E_1 : T_1' \\
T_1' \equiv T_1 \\
\hline \\
A \vdash (E_0 E_1) : T_2
\end{align*}
\]

Programming Paradigms

By constructing a dependency 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 dup (or some means of naming a value), POSTFIX is unable to make more than one reference to any value.

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

Potpourri

Yes, it affects termination. Example:
((abs (a) 3)
  ((abs (x) (x x)))
  (abs (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, Nameable would be Expressible) (2) Dynamic scoping (because elements of the Proc domain take an environment).

1000 The reoident 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.

\[ T = \text{reoident t (pairof bool (pairof int t))} \]

Round 3: Final Jeopardy

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