**Programming Languages (CS 329)** **Midterm examination**
**Computer Science and Engineering** **2000-09-20**
**Indian Institute of Technology Bombay** **(Open book, open notes)**

This exam consists of pages 1 through 2. Credit for each question is marked alongside and the total is given at the end. For code evaluation, it is advisable to provide some verbal or formal justification, because this will earn you partial credit in case the final answer is wrong.

1. (3 points) Implement a macro `LOG2` which takes a Church numeral $n$ as input and returns $\lfloor \log_2 n \rfloor$. You may not use $Y$ or recursion, but you may invoke any other macro mentioned in class.

2. Complete the following implementation of the factorial function by filling in the blanks.

```
λ  n . (   (λ  m (( (INC n) ____ ) ⊥))
           (λ  f (λ  x ((ZERO? x) 1 (MULT x (____ (DEC x)))) ) )   )
```

   (a) Write out the complete expression. (3 points)

   (b) Explain why it works by tracing the reductions when the expression is applied to argument $\bar{2}$. (2 points)

3. Evaluate the following code using

   (a) Static scoping, pass by value (2 points)

   (b) Dynamic scoping, pass by name (2 points)

```
(let a = 1
    (let f = (lambda (x y) (/ x a))
        (let a = 2
            (f 4 (let a = 0 (f 2 2)))  )  )  )
```

4. Consider the following expression.

```
(let f = (lambda (x y) (x := (+ (read y) (read y))))
    (let a = (new 0), b = (new 0)
        (begin
            (f  a
                (begin
                    (b := (+ (read b) 1))
                    b)    )
            (+ (read a) (read b))
        )    )    )
```

   Evaluate the expression using static scope and

   (a) Pass by value (2 points)

   (b) Pass by name (2 points)

5. In languages like Fortran, all identifiers implicitly represent references to cells which hold values. In such a language FLR!, we can write the code in the previous question in a more readable form:

```
(let f = (lambda (x y) (x := (+ y y)))
    (let a=0, b=0
        (begin
            (f  a
                (begin
                    (b := (+ b 1))
                    b)    )
            (+ a b)
        )    )    )
```

(a) (6 points) Write a translator $\mathcal{T}$ from FLR! to pass-by-value FLK! by completing the following equations.

$$\begin{aligned}
\mathcal{T}[\![\texttt{I}]\!] &= \underline{\quad\quad} \\
\mathcal{T}[\![\texttt{(lambda I } E\texttt{)}]\!] &= \texttt{(lambda I } \mathcal{T}[\![\texttt{E}]\!]\texttt{)} \\
\mathcal{T}[\![\texttt{(I := } E\texttt{)}]\!] &= \underline{\quad\quad} \\
\mathcal{T}[\![\texttt{(call } E_1 \ \underline{\quad\quad}\texttt{)}]\!] &= \texttt{(call } \mathcal{T}[\![E_1]\!] \ \underline{\quad\quad}\texttt{)}
\end{aligned}$$

(b) (2 points) Evaluate the above code using FLR! with static scoping, where we have modified $\mathcal{T}$ so that

$$\mathcal{T}[\![\texttt{(call } E_1 \texttt{ I)}]\!] = \texttt{(call } \mathcal{T}[\![E_1]\!] \texttt{ (new } \mathcal{T}[\![\texttt{I}]\!]\texttt{) )}$$

6. (6 points) We will extend FL to a language FLOP which will support object-oriented programming.

$$\begin{aligned}
E &::= \ \ldots \\
&\quad\mid \ \texttt{(object } M^*\texttt{)} \\
&\quad\mid \ \texttt{(}E_1\texttt{.N } E^*\texttt{)} \\
M &::= \ \texttt{(method N (I}^*\texttt{) } E_b\texttt{)}
\end{aligned}$$

Here `N` is the name of a method. Using our experience with FLK to guide you, write down denotational semantics for each production rule above, i.e., complete the following equations:

$$\begin{aligned}
\mathcal{E}[\![\texttt{(object)}]\!] &= \lambda u\,.\,\underline{\quad\quad} \\
\mathcal{E}[\![\texttt{(object } M\texttt{)}]\!] &= \lambda u\,.\,\underline{\quad\quad} \\
\mathcal{E}[\![\texttt{(object } M_1 \ M_2\texttt{)}]\!] &= \lambda u\,.\,\underline{\quad\quad} \\
\mathcal{E}[\![\texttt{(method N (I}^*\texttt{) } E_b\texttt{)}]\!] &= \lambda u\,.\,\underline{\quad\quad} \\
\mathcal{E}[\![\texttt{(}E_1\texttt{.N } E^*\texttt{)}]\!] &= \lambda u\,.\,\underline{\quad\quad}
\end{aligned}$$

As in class, assume that the leftmost definition of a method name has priority during method lookup.

**Total points: 30**