CMPSC 461: Programming Language Concepts
Midterm 1 Solution

Lambda Calculus

Problem 1 [10pt] Consider a λ-term

$$\lambda x.\ \lambda y.\ x\ z\ \lambda x.\ x$$

1. (6pt) Compute the set of free variables in the term. **Show the detailed derivation in your answer**.

   only z is free

2. (4pt) Based on the results above, connect all bound variables to their definitions with lines. For example, the answer for $\lambda x.\ x\ x\ y$ should be written as $\lambda\, \underbrace{x.\ \underbrace{x}\ x}\ \ y$.

   $\lambda\, \underbrace{x.\ \lambda y.\ x}\ z\ \lambda\, \underbrace{x.\ x}$

Problem 2 [10pt] Fully ... ed derivation in your answer.

$$
\begin{aligned}
&(\lambda x\ y\ z.\ x\ z)\ (\lambda x \\
=&(\lambda x.\ \lambda y.\ \lambda z.\ x\ z)\ (\lambda x.\ x)\ v \\
=&(\lambda y.\ (\lambda z.\ ((\lambda x.\ x)\ z)))\ v \\
=&\lambda z.\ ((\lambda x.\ x)\ z) \\
=&\lambda z.\ z
\end{aligned}
$$

Problem 3 [12pt] Recall the following encodings defined in lectures:
TRUE $\triangleq \lambda x\ y.\ x$
FALSE $\triangleq \lambda x\ y.\ y$
One way of encoding the logical "and" in λ-calculus is as follows:

$$\texttt{AND} \triangleq \lambda x\ y.\ x\ y\ \texttt{FALSE}$$

1. (6pt) Write down an encoding of the logical "or" in λ-calculus so that it computes on TRUE and FALSE (e.g., OR FALSE TRUE is TRUE; OR FALSE FALSE is FALSE).

$$\texttt{OR} \triangleq \lambda x\ y.\ x\ \texttt{TRUE}\ y$$

2. (6pt) Based on your encoding, show that `OR FALSE FALSE` evaluates to `FALSE`. **Show the detailed derivation in your answer**.

$$
\begin{aligned}
&\texttt{OR FALSE FALSE}\\
=&(\lambda x\ y.\ x\ \texttt{TRUE}\ y)\ \texttt{FALSE FALSE}\\
=&\texttt{FALSE TRUE FALSE}\\
=&(\lambda x\ y.\ y)\ \texttt{TRUE FALSE}\\
=&\texttt{FALSE}
\end{aligned}
$$

Scheme

Problem 4 [30pt]

1. (10pt) Consider the following Scheme program:


```
(define (foo l)  (map (lambda (x) (/ x 2)) l))
```

- What is the re
- In general, g rn in one sentence.

(1 2 3 4)
In general, this function returns a list where each element is d

2. (10pt) Consider the following Scheme program:

```
(define (goo f l)
  (cond ((null? l) '())
        (else (let ((result (goo f (cdr l))))
                (append (f (car l)) result)))))
```

- What is the result of `(goo (lambda (x) (list x x)) '(1 4 9 16))`?
- In general, given parameters `f l`, describe what does this function return in one sentence.

(1 1 4 4 9 9 16 16)
In general, this function returns a list where elements are the concatenation of the results of applying `f` to elements in `l`, in the same order.

3. (10pt) Use the `foldl` function in Scheme to implement a function `newfilter`, which has the same functionality as the `filter` function in Scheme.

```
(define (newfilter f l)
  (foldl (lambda (x c) (if (f x) (append c (list x))
                            c)) '() l))
```

Syntax

Problem 5 [16pt]

1. (8pt) Write down a regular expression that captures all numbers from 1 to 33.

   [1-9] | [12] [0-9] | 3 [0-3]

2. (8pt) Write down a regular expression that captures nonempty binaries (strings with 0's and 1's) that start and end with **different digits** and with **at least three digits**. For example, 011 and 1100 are such binaries, while 1 and 010 are not.

   1(0|1)+0 | 0(0|1)+1

Problem 6 [6pt] For the f us. If it is ambiguous, give an equi precedences and associativities enforced by the grammar. Assu nal that generates identifiers.

$E ::= E + E \mid F$
$F ::= G * F \mid G$
$G ::= Id \mid (E)$

Ambiguous. It is equivalent to the grammar below:

$E ::= E + F \mid F$
$F ::= G * F \mid G$
$G ::= Id \mid (E)$

Problem 7 [16pt] Consider the context free grammar for (simplified) λ-calculus:

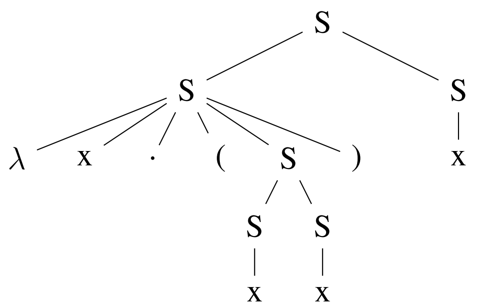
$$S ::= \mathtt{x} \mid \lambda\mathtt{x}.\ (S) \mid S\ S$$

where x is a terminal for letter x, symbols 'λ', '.', '(', ')' are terminals.

1. (8pt) Write down the **leftmost derivation** for the string "λx.(x x) x". **Show the detailed derivation in your answer**.

$S \Rightarrow S\ S \Rightarrow \lambda x.(S)\ S \Rightarrow \lambda x.\ (S\ S)\ S \Rightarrow \lambda x.\ (x\ S)\ S \Rightarrow \lambda x.\ (x\ x)\ S \Rightarrow \lambda x.\ (x\ x)\ x$

2. (8pt) Draw the *concrete* parse tree for the string "λx.(x x) x".



Bonus Questions

Bonus Problem 1 [5pt] Implement a function `findMax` **using foldl** in Scheme to get the largest element from a list of numbers. If the list is empty, your implementation should return `−inf.0`, which represents negative infinity in Scheme. Here are a couple of examples:

```
(findMax  '())              ; returns -inf.0
(findMax  '(1 -3 9 13 -2))  ; returns 13
(findMax  '(-4 -5 -1))      ; returns -1
```

```
(define (findMax lst)
   (foldl (lambda (x c) (if (> x c) x c)) -inf.0 lst))
```

Bonus Problem 2 [5pt] A list can be encoded in the λ-calculus by its `fold` function (the `foldr` function in Scheme). For example, a list with x,y,z can be encoded as $\lambda c.\ \lambda n.\ (c\ x\ (c\ y\ (c\ z\ n)))$. Moreover, an empty list `nil` is simply encoded as $\texttt{nil} \triangleq \lambda c.\ \lambda n.\ n$.

Under such encoding, define a function `cons` (the `cons` function in Scheme) that takes an element h, a list t and returns a lis

$\texttt{cons} \triangleq (\lambda h. \lambda t. (\lambda c\ n\ .$