Stamatios Morellas
COM S 342 - Exam 2
Section A
04/09/2020

1. Justify the answer of true, false, or unknown.
    a) **False** - A sentence is syntactically correct if it follows the grammar of the language in which the sentence is written. This does not mean that the semantics of the language can be considered ambiguous for every scenario, although there can be cases in which it could.
    b) **True** - Two different grammars (one with X amount of production rules and one with X + 1 amount of production rules) can produce the same sentence as long as the extra production rule in the respective grammar is able to hold an empty (null) value.
    c) **Unknown** - More information required.
    d) **False** - Static scoping means that a variable always refers to its top-level environment. If variables cannot be declared outside of the functions inside a program, then they will not be able to be accessed by more than one function. In the below example, x = 13 is declared before x = 20, meaning that it is "higher-level" than the latter. However, 20 is still printed, which is the innermost declaration of x, therefore, this assumption is false.

```c
#include <sdtio.h>

int main() {
    printf("%d", f()); // prints 20
    return 0;
}

int f() {
    int x = 13;
    return g();
}

int g() {
    int x = 20;
    return x;
}
```

    e) **True** - By altering the value of the parameters and
2. Consider the following grammar for expressions in first-order-logic
    a) G is ambiguous because FOL can either be **true** or ¬**false**, producing the same result (true) with two different parse trees.

b) Grammar G':

FOL → true | false | V | NOT | XOR | DA | (FOL)
NOT → ¬FOL
XOR → FOL ⊕ FOL
DA → ∃V FOL
V → x | y | z

3. Given the lambda expressions and corresponding interpretations, compute the following:

a) The function can be broken down into simpler terms:

```
⇒ (sec r)/(fst r)
⇒ λr.(r λa.λb.b)/λr.(r λa.λb.a)
⇒ λr.(r 0)/λr.(r 1)
# putting it all together...
⇒ λr.((Pair λr.(r 0)) λr.(r 1))
```

The interpretation comes out to be the rational number 0/1 which evaluates to be **zero**.

b) The function can be broken down to simpler terms:

```
(base) ⇒ λr1.λr2.((Pair ((mul (fst r1)) (sec r2))) ((mul (sec r1)) (fst r2)))
⇒ ((mul (fst r1)) (sec r2))) / ((mul (sec r1)) (fst r2)))
⇒ (((fst r1) (add (sec r2))) zero) / (((sec r1) (add (fst r2))) zero)
⇒ ((λr1.(r1 1) (add (λr2.(r2 0)))) zero) / ((λr1.(r1 0)) (add (λr2.(r2 1))))
zero)

Assume r1 = a/b and r2 = c/d

This means that the function computes (a*d)/(b*c)
```

The interpretation translates to the multiplication of r1 numerator and r2 denominator, all divided by the product of r1 denominator and r2 numerator.

c) The lambda expression for the function that takes two positive rational numbers as inputs and outputs their sum:

```
λr1.λr2.(add (Pair (mul (fst r1) (sec r2)) (mul (sec r1) (sec r2))) (Pair (mul (fst
r2) (sec r1)) (mul (sec r1) (sec r2))))
```

d) One way to represent signed rational numbers is to modify the formal parameters so that instead of inputting a natural number, you input a 2-element list containing a natural number as the first element, and an 8 or 9 as the second element to represent positive or negative sign,

respectively. Doing this would rid us of having to add an interpretation for subtraction, since each parameter now comes with a corresponding sign associated with it. We can also reuse the Pair interpretation to represent the value of a signed rational number. At this point, we would still need to add three interpretations to add, multiply, and divide two signed rational numbers; we can use add to either add or subtract based on the second list values of m and n.

4. Application of functions to output the number of students in the list with A's:

```
> (myf2 (lambda (a1)
            (lambda (a2)
                (lambda (a3)
                    (if (equal? (cddr a1) "A")
                         (+ a2 1)))))
      0
      '(students))
```

5. Evaluate the semantics of the following programs:
   a) See below:

Statement	Env	Result
`var (x 10)`	`(x 10)`	
`var (y 20)`	`(y 20) (x 10)`	
`fun ((f ()) (+ x y))`	`((f ()) (+ x y)) (y 20) (x 10)`	
`var (x 20)`	`(x 20) ((f ()) (+ x y)) (y 20) (x 10)`	
`apply ((f ()) 0)`	`((+ x y) 0) (x 20) ((f ()) (+ x y)) (y 20) (x 10)`	`40`
`apply ((f ()) 1)`	`((+ x y) 1) (x 20) ((f ()) (+ x y)) (y 20) (x 10)`	`30`
`(+ (apply (f ()) 1)` `(apply (f ()) 0)`		`70`

The program for part a evaluates to **70**.

   b) See below:

Statement	Env	Result
`var (x 20)`	`(x 20)`	

`var (y 10)`	`(y 10) (x 20)`	
`fun ((f ()) (+ x y))`	`(f … ) (y 10) (x 20)`	
`fun ((g ...) ...)`	`(g … ) (f … ) (y 10) (x 20)`	
`var (y 50)`	`(y 50) (g … ) (f … ) (y 10) (x 20)`	
`apply ((g ()) 0)`		`130`
`apply ((g ()) 1)`		`40`
		`170`

The program for part b evaluates to **170**.

6.

a) The final value of c will be zero because of the block context rules. Assigning a to 20 inside the if block will not change what b points to.