William Schell
COG 376
Assignment 3

1) Read the following code:

```
>>> lp = nltk.LogicParser()
>>> SnF = lp.parse('SnF')
>>> NotFnS = lp.parse('-FnS')
>>> R = lp.parse('SnF -> -FnS')
>>> prover = nltk.Prover9()
>>> prover.prove(NotFnS, [SnF, R])
```

Explain what you think it does.

- This code implements a parser named lp. It then parses three variables. After that it takes the variables in R and checks to see if the statement is true, which it will return the vale of True.

2) READ the
following code using the NLTK:

```
>>> read_expr = nltk.sem.Expression.fromstring
>>> expr = read_expr('walk(angus)', type_check=True)
>>> expr.argument
<ConstantExpression angus>
>>> expr.argument.type
e
>>> expr.function
<ConstantExpression walk>
>>> expr.function.type
<e,?>
```

Explain what you think it does.

- The code reads the statement 'walk(angus)' and it wants to determine if this is a true statement. It then takes the expressions and gives each expression a a type.

3) **READ** the following code:

```
>>> sig = {'walk': '<e, t>'}
>>> parsed = tlp.parse('walk(angus)', sig)
>>>parsed.function.type
>>> a4 = lp.parse('exists y. (woman(y) & all x. (man(x) → love(x,y)))')
>>> a5 = lp.parse('man(adam)')
>>> a6 = lp.parse('woman(eve)')
>>> g = lp.parse('love(adam,eve)')
>>> mc = nltk.MaceCommand(g, assumptions=[a4, a5, a6])
>>> mc.build_model()
```

Explain what you think it does.

- This code takes a man and tells us if he loves a woman other than the moan who is stated by printing out True of False.

4) READ
the following code:

```
>>> v = """
... bertie => b
... olive => o
... cyril => c
... boy => {b}
... girl => {o}
... dog => {c}
... walk => {o, c}
... see => {(b, o),
(c, b), (o, c)}
... """
>>> val =
nltk.parse_valuation(v)
>>> print val
{'bertie': 'b',
 'boy': set([('b',)]),
 'cyril': 'c',
 'dog': set([('c',)]),
 'girl': set([('o',)]),
 'olive': 'o',
 'see': set([('o','c'), ('c', 'b'), ('b', 'o')]),
 'walk': set([('c',),('o',)])}
```

Explain what you think it does.

- The top part is instantiating things in a way of a context free grammar. The bootm portion is applying the top part, where if we look at 'see' it is saying Olive sees Cyril, Cyril sees Bertie, and Bertie sees Olive.

5) **READ** the following code:

```
>>> lp =nltk.LogicParser()
>>> e = lp.parse(r'\x.(walk(x) & chew_gum(x))')
>>> e
<LambdaExpression \x.(walk(x) & chew_gum(x))>
>>> e.free()
set([])
>>> print lp.parse(r'\x.(walk(x) & chew_gum(y))')
\x.(walk(x) & chew_gum(y))
```

Explain what you think it does.

- You enter into the parser a statement. Then when you type in expr it tells you that it is a lambda expression. Then the last statement prints the lambda expression with a 'y' under chew_gum.