CSC208 01.31 lab

Budhil Thijm
Havin Lim * Marty Allen

January 2023

1 Equivalence Propositions

```
# One of our helper functions to get Scheme-like list behavior
# from Python lists. l[1:] returns all the elements of l,
# starting with index 1, i.e., without the head.
[[def tail(l):
    return l[1:]]]
[[def list_length(l):
    if l == []:
        return 0
    else:
        return 1 + list_length(tail(l))]]
```

Claim: (list_length [21, 7, 4]) ≡ 3.
Proof: By substituting the numbers into the left side, the left side simplifies to:
→
[[if [21, 7, 4] == []:
    return 0
else:
    return 1 + list_length(tail([21, 7, 4]))]]
→ list is not empty, so we consider the "else" case
[[else:
    return 1 + list_length(tail([21, 7, 4]))]]

→ [[return 1 + list_length(l[7, 4])]]

→ [[return 1 + [[return 1 + list_length(tail[7, 4])]]]]

→ [[return 1 + [[return1 + list_length(l[4])]]]]

→ [[return 1 + [[return 1 + [[return 1 + list_length(tail([4])]]]]]]

→ [[return 1 + [[return 1 + [[return 1 + list_length([])]]]]]]

→ [[return 1 + [[return 1 + [[return 1 + 0]]]]]]

→ [[return 1 + [[return 1 + [[return 1]]]]]]

→ [[return 1 + [[return 1 + 1]]]]

→[[return 1 + [[return 2]]]]

→ [[return 1 + 2]]

→ [[return 3]]

→ 3

2 More Equivalences

```
    # Helpers for Scheme-like manipulation of Python lists
def head(l):
    return l[0]

def cons(x, l):
    return [x] + l

def tail(l):
    return l[1:]

def list_replicate(x, n):
    if n == 0:
        return 0
    else:
        return cons(x, list_replicate(x, n-1))

def list_append(l1, l2):
    if l1 == []:
        return l2
    else:
        return cons(head(l1), append(tail(l1), l2))

define list_filter(f, l):
    if l == []:
        return []
    elif f(head(l)):
        return cons(head(l), list_filter(f, tail(l)))
    else:
```

```
        return list_filter(f, tail(l))
```

2.1 a

```
    list_replicate('!', 3) = ['!', '!', '!'].
    proof: with '!' as x, and 3 as n, the left side simplifies to:
    def list_replicate('!', 3):
        if 3 == 0:
            return 0
        else:
return cons('!', list_replicate('!', 3-1))
    --> 3 is not equal to 0, so we consider the else, and 3-1 simplifies to 2.
    [[if 3 == 0:
        return 0
    else:
        return cons('!', list_replicate('!', 2)]]
    --> then simplifying the ecxpression further, it continues until n is 0.
    [[else:
        return cons('!', cons('!', list_replicate('!', 1)]]
    -->
    [[else:
        return cons('!', cons('!', cons('!', list_replicate('!', 0))))]]
    --> list_replicate('!', 0) returns 0
    [[else:
        return cons('!', cons('!', cons('!', 0)))]]
    --> now the cons can combine lists together
    [[else:
        return cons('!', cons('!', ['!']))]]
    -->
    [[else:
        return cons('!', ['!', '!'])]]
    -->
    [[else:
        return ['!', '!', '!']]]
    -->['!', '!', '!']
```

2.2 b

```
    (list_length(list_append [1, 2] [3, 4, 5])) = list_length([1, 2]) + list_length([3, 4
    proof:
    --> first, we simplify the left side, starting with list_append
    [[(list_length([1, 2, 3, 4, 5]))]]
    --> 5
    --> Now, we simplify the right side:
    [[list_length([1, 2]) + list_length([3, 4, 5])]]
    --> [[2 + list_length([3, 4, 5])]]
```

```
--> [[2 + 3]]
--> 5
--> since both sides evaluate to 5, our claim that they are equivalent is provable
```

2.3 c

```
list_length(list_filter(lambda x: x >= 0, [1, 2, 3, 4, 5])) <= 5  True
proof: Start by simplifying the left side of the equation
--> [[list_length(list_filter(lambda x: x >= 0, [1, 2, 3, 4, 5])) <= 5]]
--> [[list_length([1, 2, 3, 4, 5]) <= 5]]
--> [[5 <= 5]]
--> True
```

3 Or Not

Consider the following definition of the Boolean xor function:

```
def xor(b1, b2):
if b1:
    return not b2
else:
    return b2
```

3.1 Claim 1

Claim 1: (Xor can return true): There exists a Boolean b such that xor(True, b) = True.
Proof: We assume that the first argument of xor is True. Since b2 is boolean, there are two cases for xor(True, b2). When b2 = True, and when b2 = False.

```
Case 1: xor(True,True)
        xor(True, True):
        if True:
            return not True
        else:
            return True
--> [[xor(True, True):
        if True:
            return not True]]
--> [[return not True]]
--> [[return False]]
--> False

Case 2 : xor(True,False)
    xor(True, False):
```

```
        if True:
            return not False
        else:
            return True

--> [[xor(True, False):
        if True:
            return not False]]
--> [[return not False]]
--> [[return True]]
--> True
```

In Case 2, since the expression xor(True, False) evaluates to True, the claim is Provable.

3.2 Claim 2

Claim 2: For any boolean b, xor(b, b) = False

Proof: There are two possible cases for this claim. xor(True, True), and xor(False, False).

```
Case 1:
def xor(True, True):
    if True:
        return not True
    else:
        return True
--> [[if True:
        return False]]
--> [[return False]]
--> False

Case 2:
def xor(False, False):
    if False:
        return not False
    else:
        return False
--> [[else:
        return False]]
--> [[return False]]
--> False
```

Both cases when b is True or False xor returns False.

3.3 Claim 3

Claim 3: For any pair of booleans b1 and b2, xor(b1, b2) = (b1 or b2) and (not (b1 and b2)).

Proof: We will start by checking the cases where b1 and b2 are equal. From Claim 2, xor(True, True) and xor(False, False) are both False, we can evaluate the left-hand-side is False. The right-hand-side evaluates as follows.

```
(True or True) and (not (True and True))
--> [[True and (not (True and True))]]
--> [[True and (not True)]]
--> [[True and False]]
--> False

(False or False) and (not (False and False))
--> [[False and (not (False and False))]]
--> [[False and (not False)]]
--> [[False and False]]
--> False
```

From Claim 1, xor(True, False) gives True. The right-hand-side evaluates as follows.

```
(True or False) and (not (True and False))
--> [[True and (not (True and False))]]
--> [[True and (not False)]]
--> [[True and True]]
--> True
```

As a result both sides are equal.
xor(False, True) evaluates as follows.

```
def xor(False, True):
    if False:
        return not True
    else:
        return True
--> [[else:
        return True]]
--> [[return True]]
--> True
```

The right-hand-side evaluates as follows.

```
(False or True) and (not (False and True))
--> [[True and (not (False and True))]]
--> [[True and (not False)]]
--> [[True and True]]
--> True
```

As a result both sides are equal.
With these cases we looked through every possible combinations of b1 and b2.
The Claim is provable.

4 Dispute