CS 560: Homework 5(b)

Eric Stevens

Question 1

If the social security number of person `X` is the same as the social security number of person `Y` then `X` and `Y` are the same person. From KB we should be able to conclude that $\phi(johnsmith)$ and $\phi(johnwsmith)$ are the same individual.

Question 2

Reflexive, Symmetric, and Transitive:

- X = X
- X = Y ← Y = X
- X = Z ← X = Y ^ Y = Z

Axiom Schema:

- ssn(X) = ssn(Y) ← X = Y (we wont have to use this)

Question 3

? johnsmith = johnwsmith

yes ← johnsmith = johnwsmith

- **Use:** X = Y ← ssn(X) = ssn(Y)
- **Sub:** {X/johnsmith, Y/johnwsmith}

yes ← ssn(johnsmith) = ssn(johnwsmith)

- **Use:** X = Z ← X = Y ^ Y = Z
- **Sub:** {X/ssn(johnsmith), Z/johnwsmith}

yes ← ssn(johnsmith) = Y1 ^ Y1 = ssn(johnwsmith)

- **Use:**
- **Sub:** {Y1/123456789}

yes ← ssn(johnsmith) = 123456789 ^ 123456789 = ssn(johnwsmith)

- **Use:** ssn(johnsmith) = 123456789
- **Sub:** {}

yes ← 123456789 = ssn(johnwsmith)

- **Use:** X = Y ← Y = X
- **Sub:** {X/123456789, Y/ssn(johnwsmith)}

yes ← ssn(johnwsmith) = 123456789

- **Use:** ssn(johnwsmith) = 123456789
- **Sub:** {}

yes ←

Question 4

The book does not clearly specify the liberties that may be taken when using paramodulation. While it may not be of utility in a fully fleged RSA, there is a way to hack your way to a paramodulation style solution in this case.

rule : ssn(johnwsmith) => 123456789

yes ← johnsmith = johnwsmith

- **Use:** X = Y ← ssn(X) = ssn(Y)
- **Sub:** {X/johnsmith, Y/johnwsmith}

yes ← ssn(johnsmith) = ssn(johnwsmith)

- **Rewrite:** ssn(johnwsmith) => 123456789

yes ← ssn(johnsmith) = 123456789

- **Use:** ssn(johnsmith) = 123456789
- **Sub:** {}

yes ←

In general, if you are allowed to choose when and where rewrites take place then you can use paramodulation to solve this problem.

Question 5

The below code does not function properly. It appears to be function properly until the point where an atom is resolved that causes one of the delayed `notequal` atoms atoms to be grounded to a constant that will make it evealuate to false. I have run out of time to solve this problem.

In [8]:

```
def prove(query,kb):

    vars = findvariables(query,[])
    print("Vars in query are %s" % vars)
    answer = [['yes']+vars]+query
    print("Initial answer clause is %s" % prettyclause(answer))

    # the initial frontier is a list whose only element is the initial
    # answer clause
    frontier = [answer]

    while frontier:
        # give answer clause fresh variables
        answer = frontier[0]
        if len(answer) == 1:
            yesatom = answer[0]
            str = "Proof:"
            for var, val in zip(vars,yesatom[1:]):
                str += " %s=%s" % (var,prettyexpr(val))
            print(str)
            return True

        print("Trying to prove : %s" % prettyclause(answer))
        # give it fresh variables
        answer = freshvariables(answer)
        print("\n after fresh vars : %s" % prettyclause(answer))

        neighbors = []

        i=1
        while delay(answer[i]):
            print(i)
            i+=1
            if i > 1000: break

        print("\n\n EVALUATING ANSWER ATOM: ", answer[i],"\n\n")

        if answer[i][0] == 'notequal':

            if notequal(answer[i]):
                print(" \n using notequal rule\n" )
                # create answer clause
                answercopy = answer[0:i]+answer[i+1:] # changed answer
                # apply substitution
                answercopy = substitute(answercopy,subs)
                print(" result: "+prettyclause(answercopy))
                neighbors.append(answercopy)
            else:
                return False

        else:
            for rule in kb:
                subs = {}
                if not unify(rule[0],answer[i],subs): #changed answer i
                    print('FAILED TO UNIFY ',rule[0], answer[i])
                    continue
                print('UNIFYing ',rule[0], answer[i])
```

```
                print(" \n using rule %s\n" % prettyclause(rule))
                print(" proven by %s with %s\n" % (rule[0],subs))
                # create answer clause
                answercopy = answer[0:i]+rule[1:]+answer[i+1:] # change
                # apply substitution
                answercopy = substitute(answercopy,subs)
                print(" result: "+prettyclause(answercopy))
                neighbors.append(answercopy)
                break
        frontier = neighbors+frontier[1:]
        #print('New Front: %s' % prettyclause([frontier]))
        print('\n\n')
    return False

def notequal(predicate):
    if predicate[0] == 'notequal':
        if not unify(predicate[1],predicate[2],{}): return True
        if predicate[1] == predicate[2]: return False
    return True

def delay(predicate):
    if predicate[0] == 'notequal':
        if not unify(predicate[1],predicate[2],{}): return False
        if predicate[1] == predicate[2]: return False
        else: return True
    # predicate is not 'notequal'
    return False


```

In [ ]:

```
from hw4standard import *

prove([['mct',['s',['s',['s',['s',['s','0']]]]],'List']],kb)
```

In [14]:

```
kb = [
    [['block', 'red1']],
    [['block', 'red2']],
    [['block', 'red3']],
    [['block', 'red4']],
    [['block', 'red5']],
    [['block', 'red6']],
    [['block', 'red7']],
    [['block', 'gre1']],
    [['block', 'gre2']],
    [['block', 'gre3']],
    [['block', 'blu1']],
    [['block', 'blu2']],
    [['block', 'yel1']],
    [['block', 'bla1']],
    [['color', 'red1', 'red']],
    [['color', 'red2', 'red']],
    [['color', 'red3', 'red']],
    [['color', 'red4', 'red']],
    [['color', 'red5', 'red']],
    [['color', 'red6', 'red']],
    [['color', 'red7', 'red']],
    [['color', 'gre1', 'green']],
    [['color', 'gre2', 'green']],
    [['color', 'gre3', 'green']],
    [['color', 'blu1', 'blue']],
    [['color', 'blu2', 'blue']],
    [['color', 'yel1', 'yellow']],
    [['color', 'bla1', 'black']],


    [['mct', ['s','0'], ['p','Block','nil']], ['block', 'Block']],

    [['mct', ['s','X'],'NewTower'],
        ['notequal', 'X','0'],
        ['mct', 'X', 'Tower'],
        ['unify', 'Tower', ['p','Top','US']],
        ['notequal', 'TopColor', 'BlockColor'],
        ['differentfromlist', 'Block', 'Tower'],
        ['color', 'Top', 'TopColor'],
        ['block', 'Block'],
        ['color', 'Block', 'BlockColor'],
        ['unify','NewTower',['p','Block','Tower']]
    ],

    [['differentfromlist', 'X', ['p','Y','nil']],
        ['notequal', 'X','Y']
    ],

    [['differentfromlist', 'X', ['p','Top','Rest']],
        ['notequal', 'X','Top'],
        ['differentfromlist','X','Rest']
    ],



    [['unify', 'X', 'X']],
```

```
58
59 ]
60
```

Question 6

```
step1(impliedby(impliedby(A,B),impliedby(C,D)),TopOutput) :-
    !,
    %write("made double implied\n"),
    step1(impliedby(A,B),OutputAB),
    step1(impliedby(C,D),OutputCD),
    TopOutput = or(OutputAB,not(OutputCD)).

step1(impliedby(X,impliedby(Y,Z)),Output) :-
    !,
    %write("made iRight\n"),
    step1(impliedby(Y,Z), Output2),
    Output = or(X,not(Output2)).

step1(impliedby(impliedby(X,Y),Z),Output) :-
    !,
    %write("made iLeft\n"),
    step1(impliedby(X,Y),Output2),
    Output = or(Output2,not(Z)).

step1(impliedby(X,Y),Output) :-
    !,
    %write("made iSingel\n"),
    Output = or(X,not(Y)).

step1(and(X,Y),TopOutput) :-
    !,
    %write("made AND\n"),
    step1(X,OutputX),
    step1(Y,OutputY),
    TopOutput = and(OutputX,OutputY).

step1(or(X,Y),TopOutput) :-
    !,
    %write("made OR\n"),
    step1(X,OutputX),
    step1(Y,OutputY),
    TopOutput = or(OutputX,OutputY).

step1(not(X),TopOutput) :-
    !,
    %write("made NOT\n"),
    step1(X,OutputX),
    TopOutput = not(OutputX).

step1(X,Output) :-
    %write("made Generalized\n"),
    Output = X.

test1 :-
```

```
    findall(X,step1(impliedby(d,e),X),L1),
    write(L1),nl,
    findall(X,step1(impliedby(d,not(e)),X),L2),
    write(L2),nl,
    findall(X,step1(impliedby(d,impliedby(e,f)),X),L3),
    write(L3),nl,
    findall(X,step1(impliedby(impliedby(a,b),impliedby(e,f)),X),L
4),
    write(L4),nl,
    findall(X,step1(and(a,(or(b,or(c,(not(impliedby(d,e)))))))),X),L
5),
    write(L5),nl.


/**
RESULTS:

?- test1.
[or(d,not(e))]
[or(d,not(not(e)))]
[or(d,not(or(e,not(f))))]
[or(or(a,not(b)),not(or(e,not(f))))]
[and(a,or(b,or(c,not(or(d,not(e))))))]
true.
**/
```