

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:

- $X = X$
- $X = Y \leftarrow Y = X$
- $X = Z \leftarrow X = Y \wedge Y = Z$

Axiom Schema:

- $ssn(X) = ssn(Y) \leftarrow X = Y$ (we wont have to use this)

Question 3

? johnsmith = johnwsmith

yes \leftarrow johnsmith = johnwsmith

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

yes $\leftarrow ssn(johnsmith) = ssn(johnwsmith)$

- **Use:** $X = Z \leftarrow X = Y \wedge Y = Z$
- **Sub:** $\{X/ssn(johnsmith), Z/johnwsmith\}$

yes $\leftarrow ssn(johnsmith) = Y1 \wedge Y1 = ssn(johnwsmith)$

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

yes $\leftarrow ssn(johnsmith) = 123456789 \wedge 123456789 = ssn(johnwsmith)$

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

yes \leftarrow 123456789 = ssn(johnwsmith)

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

yes \leftarrow ssn(johnwsmith) = 123456789

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

yes \leftarrow

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 fledged RSA, there is a way to hack your way to a paramodulation style solution in this case.

rule : ssn(johnwsmith) \Rightarrow 123456789

yes \leftarrow johnsmith = johnwsmith

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

yes \leftarrow ssn(johnsmith) = ssn(johnwsmith)

- **Rewrite:** ssn(johnwsmith) \Rightarrow 123456789

yes \leftarrow ssn(johnsmith) = 123456789

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

yes \leftarrow

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 to be grounded to a constant that will make it evaluate to false. I have run out of time to solve this problem.

```

In [2]: 1 def prove(query,kb):
2
3     vars = findvariables(query,[])
4     print("Vars in query are %s" % vars)
5     answer = [['yes']+vars]+query
6     print("Initial answer clause is %s" % prettyclause(answer))
7
8     # the initial frontier is a list whose only element is the initial
9     # answer clause
10    frontier = [answer]
11
12    while frontier:
13        # give answer clause fresh variables
14        answer = frontier[0]
15        if len(answer) == 1:
16            yesatom = answer[0]
17            str = "Proof:"
18            for var, val in zip(vars,yesatom[1:]):
19                str += " %s=%s" % (var,prettyexpr(val))
20            print(str)
21            return True
22
23        print("Trying to prove : %s" % prettyclause(answer))
24        # give it fresh variables
25        answer = freshvariables(answer)
26        print("\n after fresh vars : %s" % prettyclause(answer))
27
28        neighbors = []
29
30        i=1
31        while delay(answer[i]):
32            print(i)
33            i+=1
34            if i > 1000: break
35
36        print("\n\n EVALUATING ANSWER ATOM: ", answer[i],"\n\n")
37
38        if answer[i][0] == 'notequal':
39
40            if notequal(answer[i]):
41                print(" \n using notequal rule\n" )
42                # create answer clause
43                answercopy = answer[0:i]+answer[i+1:] # changed answer
44                # apply substitution
45                answercopy = substitute(answercopy,subs)
46                print(" result: "+prettyclause(answercopy))
47                neighbors.append(answercopy)
48            else:
49                return False
50
51        else:
52            for rule in kb:
53                subs = {}
54                if not unify(rule[0],answer[i],subs): #changed answer
55                    print('FAILED TO UNIFY ',rule[0], answer[i])
56                    continue
57                print('UNIFYing ',rule[0], answer[i])

```

```

58     print(" \n using rule %s\n" % prettyclause(rule))
59     print(" proven by %s with %s\n" % (rule[0],subs))
60     # create answer clause
61     answercopy = answer[0:i]+rule[1:]+answer[i+1:] # change
62     # apply substitution
63     answercopy = substitute(answercopy,subs)
64     print(" result: "+prettyclause(answercopy))
65     neighbors.append(answercopy)
66     break
67     frontier = neighbors+frontier[1:]
68     #print('New Front: %s' % prettyclause([frontier]))
69     print('\n\n')
70     return False
71
72 def notequal(predicate):
73     if predicate[0] == 'notequal':
74         if not unify(predicate[1],predicate[2],{}): return True
75         if predicate[1] == predicate[2]: return False
76     return True
77
78 def delay(predicate):
79     if predicate[0] == 'notequal':
80         if not unify(predicate[1],predicate[2],{}): return False
81         if predicate[1] == predicate[2]: return False
82         else: return True
83     # predicate is not 'notequal'
84     return False
85
86

```

```

In [ ]: 1 from hw4standard import *
        2
        3 prove([[ 'mct', ['s', ['s', ['s', ['s', ['s', '0']]]]], 'List']],kb)

```

```

In [4]: 1 kb = [
2         [['block', 'red1']],
3         [['block', 'red2']],
4         [['block', 'red3']],
5         [['block', 'red4']],
6         [['block', 'red5']],
7         [['block', 'red6']],
8         [['block', 'red7']],
9         [['block', 'gre1']],
10        [['block', 'gre2']],
11        [['block', 'gre3']],
12        [['block', 'blu1']],
13        [['block', 'blu2']],
14        [['block', 'yell1']],
15        [['block', 'bla1']],
16        [['color', 'red1', 'red']],
17        [['color', 'red2', 'red']],
18        [['color', 'red3', 'red']],
19        [['color', 'red4', 'red']],
20        [['color', 'red5', 'red']],
21        [['color', 'red6', 'red']],
22        [['color', 'red7', 'red']],
23        [['color', 'gre1', 'green']],
24        [['color', 'gre2', 'green']],
25        [['color', 'gre3', 'green']],
26        [['color', 'blu1', 'blue']],
27        [['color', 'blu2', 'blue']],
28        [['color', 'yell1', 'yellow']],
29        [['color', 'bla1', 'black']],
30
31
32        [['mct', ['s', '0'], ['p', 'Block', 'nil']], ['block', 'Block']],
33
34        [['mct', ['s', 'X'], 'NewTower'],
35         ['notequal', 'X', '0'],
36         ['mct', 'X', 'Tower'],
37         ['unify', 'Tower', ['p', 'Top', 'US']],
38         ['notequal', 'TopColor', 'BlockColor'],
39         ['differentfromlist', 'Block', 'Tower'],
40         ['color', 'Top', 'TopColor'],
41         ['block', 'Block'],
42         ['color', 'Block', 'BlockColor'],
43         ['unify', 'NewTower', ['p', 'Block', 'Tower']]
44     ],
45
46     [['differentfromlist', 'X', ['p', 'Y', 'nil']],
47      ['notequal', 'X', 'Y']
48 ],
49
50     [['differentfromlist', 'X', ['p', 'Top', 'Rest']],
51      ['notequal', 'X', 'Top'],
52      ['differentfromlist', 'X', 'Rest']
53 ],
54
55
56
57     [['unify', 'X', 'X']],

```

```
58  
59 ]  
60
```

Question 5 Critique

My KB is correct other than my base case for `differentfromlist`. I could not figure out how to represent an empty list and my guess was wrong. My solution is stylistically very different than the solution manual, but the operation of the code seems to be very similar. I believe I have a bug somewhere and I have not been able to find it. When running the code the operations I would expect to take place occur. I think I have some misunderstanding about the way this operation works. Also, I would like to apologize for the Python file I submitted, the code blocks are out of order as I was working on it in Jupyter where I can run block out of order. I am submitting the file that has code blocks reordered so that they can run.

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.
```

```
**/
```

Question 6 Self Assessment

While the answer I gave results in the correct solution in terms of the test cases, it has unnecessary complexity. The cases of `impliedby` are wrong. They should be consolidated into a single expression that captures all of the cases that they capture together, as is done in the solution. In the other expressions I am using the `=` expression for explicit unification where the provided solution implies unification in the head of the clauses.