CS 560: Homework 6 Critique

Eric Stevens

November 18, 2018

OVERVIEW: Please see initial submission for verbose Python output.

Question 1:

$KB =$
$\quad \neg b \vee \neg c$
$\quad b \vee \neg a$
$\quad c \vee \neg a$

Question 2:

a

Question 3:

$KB =$
$\quad \neg b \vee \neg c$
$\quad b \vee \neg a$
$\quad c \vee \neg a$
$\quad a$

Question 4:

Yes, none of the conjuncts have more than a single positive literal.

Question 5:

$C =$
$\quad \neg b \vee \neg c$
$\quad b \vee \neg a$
$\quad c \vee \neg a$
$\quad a$

Resolve: $b \vee \neg a$ and a
Result: b

$C =$
$\neg b \vee \neg c$
$b \vee \neg a$
$c \vee \neg a$
a
b

Remove: $b \vee \neg a$
Because: $b \subset b \vee \neg a$

$C =$
$\neg b \vee \neg c$
$c \vee \neg a$
a
b

Resolve: $c \vee \neg a$ and a
Result: c

$C =$
$\neg b \vee \neg c$
$c \vee \neg a$
a
b
c

Remove: $b \vee \neg a$
Because: $c \subset c \vee \neg a$

$C =$
$\neg b \vee \neg c$
a
b
c

Resolve: $\neg b \vee \neg c$
and c
Result: $\neg b$

$C =$
$\neg b \vee \neg c$
a
b
c
$\neg b$

Since our consequence set contains both b and $\neg b$ our knowledgebase is false because there is no interpretation of b that will model KB. Therefore KB' is false and therefore $KB \models \neg a$.

Question 6

Interpretations:

a	b	c	$a \lor b \lor c$	$\neg a \lor \neg b$
0	0	0	0	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Models:

a	b	c	$a \lor b \lor c$	$\neg a \lor \neg b$
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1

Question 7:

No, coming to g through a resoltion of two clauses does not imply that g is true in every model of KB. Rather, it implies that there is a model of KB in which g is true. In other words, $KB \vdash g$. It is possible that a different resolution rule could have been applied that resulted in the opposite, or both could be done over the course of a proof procedure.

1. $b \lor \neg b \lor c$

a	b	c	$a \lor b \lor c$	$\neg a \lor \neg b$
0	0	0	0	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1

a	b	c	$a \lor b \lor c$	$\neg a \lor \neg b$
1	1	0	1	0
1	1	1	1	0

Since both b and $\neg b$ are in the rule it is true for all interpretations and thus true in all models.

2. $a \lor \neg a \lor c$

a	b	c	$a \lor b \lor c$	$\neg a \lor \neg b$
0	0	0	0	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Since both a and $\neg a$ are in the rule it is true for all interpretations and thus true in all models.

3. c

a	b	c	$a \lor b \lor c$	$\neg a \lor \neg b$
0	0	1	1	1
0	1	1	1	1
1	0	1	1	1
1	1	1	1	0

No, there are several models of KB in which c evaluates to false.

Question 7 Critique:

This solution is correct other than the first paragraph. I actually initally thought that $KB \vDash g$ but the phrasing of the question made be second guess myself. I thought 'since we are working with negative knowledge, there might be a chance might be a chance for other models to exist that dont require g. The truth tables and explinations are correct.

Question 8:

1.

$a(X, c) \vee b(X, d)$ with $\neg b(e, Y)$

Resolve: $b(e, d) \vee \neg b(e, d)$
Sub: $\{X/e, Y/d\}$
Result: $a(e, c)$

2.
$a(X, c) \vee b(X, d) \vee f$ with $\neg a(e, Z) \vee b(e, Y)$

Resolve: $a(e, c) \vee \neg a(e, c)$
Sub: $\{X/e, Z/c\}$
Result: $b(e, d) \vee f \vee b(e, Y)$

Resolve: $b(e, d) \vee b(e, d)$
Sub: $\{X/e, Y/d\}$
Result: $a(e, c) \vee f \neg a(e, Z)$

Question 9:

The first part of the question above is an example of a **unit resolution** since one of the resolvants was done with a unit literal.

Question 10:

$\neg mortal \longleftarrow mythical$
$mortal \wedge mammal \longleftarrow \neg mythical$
$horned \longleftarrow \neg mortal \vee mammal$
$magical \longleftarrow horned$

Question 11:

$\neg mortal \vee \neg mythical$
$mortal \vee mythical$
$mammal \vee mythical$
$horned \vee mortal$
$horned \vee \neg mammal$
$magical \vee \neg horned$

Question 12:

The set of clauses is not Horne or Datalog because there is more than one positive literal in multiple clauses.

Question 13:

$daughter(Daughter, Mother) \lor male(Daughter) \lor \neg mother(Mother, Daughter)$
$mother(mary, nancy)$
$\neg male(nancy)$

Question 14:

The above is neither Horne nor Datalog because the first statement has more than one positive literal.

Question 15:

Initialize C with KB:

$C =$
  $daughter(Daughter, Mother) \lor male(Daughter) \lor \neg mother(Mother, Daughter)$
  $mother(mary, nancy)$
  $\neg male(nancy)$

Resolve: $mother(mary, nancy) \lor \neg mother(Mother, Daugher)$
Sub: $\{Mother/mary, Daughter/nancy\}$
Result: $daughter(nancy, mary) \lor male(nancy)$

$C =$
  $daughter(Daughter, Mother) \lor male(Daughter) \lor \neg mother(Mother, Daughter)$
  $mother(mary, nancy)$
  $\neg male(nancy)$
  $daughter(nancy, mary) \lor male(nancy)$

Prune:
$daughter(nancy, mary) \lor male(nancy) \subset daughter(Daughter, Mother) \lor male(Daughter) \lor$

$C =$
  $mother(mary, nancy)$
  $\neg male(nancy)$
  $daughter(nancy, mary) \lor male(nancy)$

Resolve: $\neg male(nancy) \lor male(nancy)$
Sub: $\{\}$
Result: $daughter(nancy, mary)$

$C =$
  $mother(mary, nancy)$
  $\neg male(nancy)$
  $daughter(nancy, mary) \lor male(nancy)$
  $daughter(nancy, mary)$

Prune: $daughter(nancy, mary) \subset daughter(nancy, mary) \lor male(nancy)$

$C =$
$mother(mary, nancy)$
$\neg male(nancy)$
$daughter(nancy, mary)$

$daughter(nancy, mary)$ is in the concequence set.

Question 16:

$yes \longleftarrow daughter(nancy, mary)$

$yes \lor \neg daughter(nancy, mary)$

Use: $daughter(Daughter, Mother) \lor male(Daughter) \lor \neg mother(Mother, Daughter)$
Subs: $\{Daughter/nancy, Mother/mary\}$

$yes \lor male(nancy) \lor \neg mother(mary, nancy)$

Use: $male(nancy)$
Subs: $\{\}$

$yes \lor \neg mother(mary, nancy)$

Use: $mother(mary, nancy)$
Subs: $\{\}$

yes

Question 17:

Rewrite KB:

$poor(X) \lor \neg student(X)$
$student(john) \lor student(tim)$

$?poor(X)$

$yes(X) \lor \neg poor(X)$

Use: $poor(X) \lor \neg student(X)$
Subs: $\{\}$

$yes(X) \lor \neg student(X)$

Use: $student(john) \lor student(tim)$
Subs: $\{X/tim\}$

$yes(tim) \lor student(john)$

Use: $\neg student(X1)$
Subs: $\{X1/john\}$

$yes(tim)$

Question 17 Critique:

Here I was not sure whether we had to prove both students or only get to the first yes we could. I could have easily backtracked here and gotten yes(john) as well, and was wondering if I should have done so while I was doing the problem.

Question 18:

No, it is not possible to prove that 'tweety' can fly without the CKA. This is because we need to able reason about $abnormal(tweety)$. The falure to resolve the top down proof follows:

$?fly(tweety)$

$yes \longleftarrow fly(tweety)$

Use: $fly(X) \longleftarrow bird(X) \land \neg abnormal(X)$
Subs: $\{X/tweety\}$

$yes \longleftarrow bird(tweety) \land \neg abnormal(tweety)$

Use: $bird(tweety)$
Subs: $\{\}$

$yes \longleftarrow \neg abnormal(tweety)$

This is where the proof without CKA gets stuck. If we were able to assume that any knowledge that can not be derived from the knowledgebase is false we would be able to conclude that $abnormal(tweety)$ would evealuate to false. This would allow us to complete the proof.

Question 19:

$yes \longleftarrow fly(tweety)$

Use: $fly(X) \longleftarrow bird(X) \land \neg abnormal(X)$
Subs: $\{X/tweety\}$

$yes \longleftarrow bird(tweety) \land \neg abnormal(tweety)$

Use: $bird(tweety)$
Subs: $\{\}$

$yes \longleftarrow \neg abnormal(tweety)$

Recursion to prove $\sim abnormal(tweety)$ with NaF

$yes \longleftarrow abnormal(tweety)$

Use: $abnormal(X) \longleftarrow toy(X)$
Subs: $\{X/tweety\}$

$yes \longleftarrow toy(tweety)$

FAIL

$yes \longleftarrow abnormal(tweety)$

Use: $abnormal(X) \longleftarrow dead(X)$
Subs: $\{X/tweety\}$

$yes \longleftarrow dead(tweety)$

FAIL

$yes \longleftarrow abnormal(tweety)$

FAIL

Add $\sim abnormal(tweety)$ to C

$yes \longleftarrow \neg abnormal(tweety)$

Use: $\sim abnormal(tweety)$
Subs: $\{\}$

yes

Question 20:

$abnormal(X) \longleftarrow toy(X) \vee dead(X)$
$toy(X) \longleftarrow X = gun$
$dead(X) \longleftarrow X = elvis$

Question 21:

$abnormal(X) \longleftrightarrow toy(X) \vee dead(X)$
$toy(X) \longleftrightarrow X = gun$
$dead(X) \longleftrightarrow X = elvis$

Question 22:

$toy(X) \vee dead(X) \longleftarrow abnormal(X)$
$X = gun \longleftarrow toy(X)$
$X = elvis \longleftarrow dead(X)$

CNF

$toy(X) \vee dead(X) \vee \neg abnormal(X)$
$X = gun \vee \neg toy(X)$
$X = elvis \vee \neg dead(X)$

Question 22 Critique:

I can see that I was expected to replace my gun and elvis constants with what I guess are fresh constants but I dont exactly understand how we can pull out fresh constants that we havent seen before.

Question 23:

$KB =$
$bird(tweety)$
$fly(X) \vee \neg bird(X) \vee abnormal(X)$
$abnormal(X) \vee \neg toy(X)$
$abnormal(X) \vee \neg dead(X)$
$toy(gun)$
$dead(elvis)$
$toy(X) \vee dead(X) \vee \neg abnormal(X)$
$X = gun \vee \neg toy(X)$
$X = elvis \vee \neg dead(X)$

$?fly(tweety)$

$yes \vee \neg fly(tweety)$

Resolve: $fly(X) \vee \neg bird(X) \vee abnormal(X)$
Sub: $\{X/tweety\}$
Result: $\neg bird(tweety) \vee abnormal(tweety)$

$yes \vee \neg bird(tweety) \vee abnormal(tweety)$

Resolve: $bird(tweety)$
Sub: $\{\}$
Result: $abnormal(tweety)$

$yes \vee abnormal(tweety)$

Resolve: $toy(X) \vee dead(X) \vee \neg abnormal(X)$
Sub: $\{X/tweety\}$
Result: $toy(tweety) \vee dead(tweety)$

$yes \vee toy(tweety) \vee dead(tweety)$

Resolve: $X = gun \vee \neg toy(X)$
Sub: $\{X/tweety\}$
Result: $tweety = gun \vee dead(tweety)$

$yes \vee tweety = gun \vee dead(tweety)$

Resolve: $X = elvis \vee \neg dead(X)$
Sub: $\{X/tweety\}$
Result: $tweety = gun \vee tweety = elvis$

Resolve:
Sub: $\{\}$
Result:

Not sure how this is supposed to work

Question 23 Critique:

Ah-ha! The unique name assumption!

$yes \lor tweety = gun \lor tweety = elvis$

Resolve: $tweety = gun \lor \neg(tweety = gun)$ With UNA
Sub: $\{\}$
Result: $tweety = elvis$

$yes \lor tweety = elvis$

Resolve: $tweety = elvis \lor \neg(tweety = elvis)$ With UNA
Sub: $\{\}$
Result:

yes

QED

Question 24:

Helper Functions

In [ ]:

```
from hw4standard import *

def PrettyCNF(expr):
    #print(expr,"\n\n")
    s = ""
    for literal in expr:
        if s != "": s += " v "
        #print(type(literal))
        if type(literal) is list:
            if literal[0] == "not":
                s += "!"
                atom = literal[1]
            else:
                atom = literal
        s += prettyexpr(atom)
    return s

def PrettyConsequence(consequence):
    '''prints an entire consequenct set'''
    for i in consequence:
        print(PrettyCNF(i))

def PrettyResolution(unit, clause, result):
    print("Resolving", PrettyCNF(clause), "with", PrettyCNF(unit))
    print("Result:", PrettyCNF(result))

def sameclause(a,b):
    subs = {}
    ret = unify(a,b,subs)
    if ret == False or subs == {}:
        return ret
    # same clause if bottom just has variables and each is different
    seen = []
    for bottom in ret.values:
        if not isVar(bottom):
            return False
        if bottom in seen:
            return False
        seen.append(bottom)
    return True


def negate(literal):
    if type(literal) is list and literal[0] == "not":
        return literal[1]
    return ['not',literal]


def resolve(unit, clause):
    '''Takes in a unit clause and a disjunct clause and returns a reso

    # negate the unit clause for resolution
    neg_unit = negate(unit)

    # look for matching disjunct in clause
    for disjunct in clause:

        subs = {}
        # find one and get subs
        if unify(neg_unit, disjunct, subs):
            sub_clause = substitute(clause, subs)
            sub_neg_unit = substitute(neg_unit, subs)

            # remove instance that was resolved
            sub_clause.remove(sub_neg_unit)

            # return new clause
            return sub_clause
# TEST
#print(resolve(['canary','tweety'],[['normal','X'],['not',['canary','X

def can_resolve(unit, clause):
    '''Checks to see if a resolution is possible'''
    neg_unit = negate(unit)
    for disjunct in clause:
        if unify(neg_unit,disjunct,{}):
            return True
    return False
# TEST
#can_resolve(['canary','tweety'],[['normal','X'],['not',['canary','X']


def already_in_c_set(resolution, c_set):
    ''' See if a clause is already in a set'''
    for disjunct in c_set:
        if sameclause(resolution, disjunct):
            return True
    return False
#TEST
#print(alread_in_c_set([['canary','tweety']],kb))
#print(alread_in_c_set([['canary','SHOULD FAIL']],kb))

def merge_sets(s1, s2):
    '''Merges to sets without overlap'''
    new_set = s1.copy()

    for expression in s2:
        if not already_in_c_set(expression, new_set):
            new_set.append(expression)
    return new_set
# TEST
#print(merge_sets(kb[:3],kb[1:5]))
```

Prove Function

In [ ]:

```
def prove(kb, query):

    # declare consequence set
    consequence = kb.copy()

    # negate query
    neg_query = negate(query)

    # add query negation to consequence set
    consequence.append([neg_query])

    #PrettyConsequence(consequence)

    # loop until the consequence set is empty
    while 1:

        temp_set = [] # new resolutions before adding to consequence

        # look for units and try to resolve
        for unit in consequence:
            if len(unit) == 1:

                # unit clause found, give fresh variables
                unit_clause = freshvariables(unit[0])

                # look at every expression in the current consequence s
                for expression in consequence:

                    # if it can be resolved do so and add the result to
                    if can_resolve(unit_clause, expression):
                        new_clause = resolve(unit_clause, expression)
                        PrettyResolution([unit_clause], expression, new
                        print("\n")
                        # success if empty disjunct
                        if new_clause == []:
                            return True

                        # otherwise check to see if resolution is in te
                        # add to temp set if it is not.
                        else:
                            if not already_in_c_set(new_clause, temp_se
                                temp_set.append(new_clause)
                                PrettyConsequence(merge_sets(consequenc
                                print("\n")

        # if no further possible resolutions, fail
        if temp_set == []:
            return False
        # otherwise, merge temp and consequence and run again
        else:
            consequence = merge_sets(consequence, temp_set)

```

Demo 1

In [ ]:

```
kb = [[['ostrich','sam']],
[['canary','tweety']],
[['bird','X'],['not',['ostrich','X']]],
[['bird','X'],['not',['canary','X']]],
[['fly','X'],['not',['bird','X']],['not',['normal','X']]],
[['not',['normal','X']],['not',['ostrich','X']]],
[['normal','X'],['not',['canary','X']]]]


print(prove(kb,['fly','tweety']))
```

Demo 2

In [ ]:

```
print(prove(kb,['fly','sam']))
```

Demo 3

In [ ]:

```
print(prove([[['boy',['goo','X','Y']], ['boy',['foo','X','Y']]]], ['boy
```

Question 25:

If any of the following is true the entire disjunct is true:

- a is true
- b is true
- c is false
- d is true

Question 26:

```
disjunction([p(true)|_]).
disjunction([n(false)|_]).
disjunction([_|Rest]) :-
        disjunction(Rest).


?- disjunction([p(A),p(B),n(C)]).
A = true ;
B = true ;
C = false ;
false.

?- disjunction([p(A),p(true),n(C)]).
A = true ;
true ;
C = false ;
false.

?-  disjunction([p(A),p(false),n(C)]).
A = true ;
C = false ;
false.
```

Question 27:

```
conjunction([X]) :- disjunction(X).
conjunction([X|Rest]) :-
        conjunction(Rest),
        disjunction(X).
```

Question 28:

```
?- conjunction([[p(A),p(B),n(C)],[p(A),n(B)],[n(A)]]).
A = B, B = C, C = false ;
false.

?- conjunction([[p(A),p(B),n(C),p(D)],[p(A),n(B)],[n(A)],[n(D),n
(A)]]).
A = B, B = C, C = D, D = false ;
A = B, B = C, C = false ;
A = B, B = false,
D = true ;
false.
```