Logic for Applications

p 165: 1

p 171: 1, 3, 5(abcdef), 8(abc), 10

Ch 1

1.

Using induction:

Basis: if G' = {~l) and C' = {l} then D' = {}, which corresponds to D as desired

Inductive hypothesis: assume D and D' correspond for all G' and C' with literals <= n

Suppose our goal clause G' has n+1 literals, with the n+1st literal x resolving against some literal y in C'. We would then get D' = G' U C' - (n+1 U l), which corresponds to D by the IH. Suppose instead that C' has n+1 literals -- we would then use the same reasoning to find the correspondence between D and D'. Last, suppose that both G' and C' have n+1 literals. In this case, we can still use the IH.

This proof is pretty awful, I know, but I'm running out of time in the quarter and will try to return to it later to do a better job.

Ch 2

1

Using an indentation tree:

~eq(c,a)

Path 1:

  ~eq(c,a)

  Path 1.1:

    ~eq(c,Y)

      failure

    ~eq(Y,a)

      failure

  failure

Path 2:

  ~eq(a,c)

    ~eq(a,Y)

      ~eq(a,b)

    ~eq(b,c)

    box!

3

Goal: ~p(X,Y)

Path 1:

  ~q(X,Y)

    ~q(a,b)

  ~p(b,Z)

    ~q(b,Y)

    failure

  Path 1.2:

  ~p(b,b) box! (X/a, Y/b)

5

a

second_element(X,[_,X|_]).

b

substitute_for_second_element(Y,[X|Xs],[X,Y|Xs]).

c

```
switch([X|Xs], [L,Zs]) :- get_last(L,Ls,Xs), append(Ls, [X], Zs).

 % first param = the last item
 % second param = given list without the last item
 % third param = given list
get_last(L,[],[L]).
get_last(L,[_|Xs],[_|Ys]) :- get_last(L,Xs,Ys).
```

d

```
?- second_element(b,[a,[b,c],d,e]).
false.
?- second_element(b,[a,b,c]).
true.
?- second_element([b,c],[a,[b,c],d,e]).
true.
?- substitute_for_second_element([a,[b,c],d],[a,b,c],[a,[b,c],d]).
false.
?- switch([a,b,c],[a,c,b]).
false.
?- switch([a,b,c],[c,b,a]).
true .
```

e

```
?- second_element(Z,[a,[b,c],d,e]).
Z = [b, c].
?- second_element(Z,[a,b,c]).
Z = b.
?- substitute_for_second_element([a,[b,c],d],[a,b,c],Z).
Z = [a, [a, [b, c], d], c].
?- substitute_for_second_element(b,[a,[a,[b,c],d],c],Z).
Z = [a, b, c].
?- switch([a,b,c],Z).
Z = [c, b, a] .
?- switch([c,b,a],Z).
Z = [a, b, c] .
?- second_element(b,Zs), substitute_for_second_element(b,Zs,[a,b,c]).
Zs = [a, b, c].
?- second_element(b,Zs), switch(Zs,[a,b,c]).
Zs = [c, b, a] .
```

f
laws:
Substituting a second element for its own second element results in the same list
?- substitute_for_second_element(b,[a,b,c],[a,b,c]).
true.
Switching a list twice yields the original list
?- switch([a,b,c],[c,b,a]), switch([c,b,a],[a,b,c]).
true .
8
a
Basis: zero added to any number is zero
IH: assume the recursive step to be true where the second parameter has length <= n.
If the second parameter has length n+1 -- that is: add(X,s(Y),s(Z)) -- then our subgoal becomes add(x,Y,Z), which we know to be addition by the IH.
b
mul(_,0,0).
mul(X,1,X).
mul(A,s(B),C) :- add(D,A,C), mul(A,B,D).
c
Basis: 0 multiplied with anything is 0. 1 multiplied with anything is that thing.
IH: assume the recursive step to be equivalent to mn=r where the second param has length <=n.
If the second param has length n+1 (mul(A,s(B),C)), then that is true if add(D,A,C) and mul(A,B,D) are true. Since x(y+1) = xy+x (or A(s(B)) = mul(A,B,C), add(C,A)), and by the inductive hypothesis we know that mul(A,B,D) <=> a*b=d, then we know mul(A,s(B),C) <=> A*s(B)=C

10.
flttn([],[]).
flttn([X|F], [X|Xs]) :- atomic(X), flttn(F, Xs).
flttn(F3, [X|Xs]) :- flttn(F1, X), flttn(F2, Xs), append(F1,F2,F3).