Lauren Fromm
404751250
CS 161

HW 6

1.

a) {x/A, y/B, z/C}
b) No unifier exists: {y/G(x,x)} -> x/A and x/B ?
c) {x/A, y/A}
d) {x/John, y/John}
e) No unifier exists: {x/y} -> x!=father(x)

2.

a)

Basic functions: food(a), likes(liker, likee), eats(eater, food), killed(killed, killer), alive(b)

John likes all kinds of food
(A x) (food(x) => likes(john,x))

Apples are food
food(apples)

Chicken is food
food(chicken)

Anything someone eats and isn’t killed by is food
(A x) ((E y) ((eats(y,x) & ~killed(y, x) => food(x)))

If you are killed by something, you are not alive
(A y, x) ((killed(y, x) => ~alive(y)))

Bill eats peanuts and is still alive
(eats(bill, peanuts) & alive(bill))

Sue eats everything Bill eats
(A x) (eats(bill, x) => eats(Sue, x))

b)

(A x) (food(x) => likes(john,x))
(~food(x) | likes(john,x))

food(apples)

food(chicken)

(A x, y) (eats(y,x) & ~killed(y) => food(x))
~(eats(y,x) & ~killed(y)) | food(x)
(~eats(y,x) | killed(y,x) | food(x))

(A y) (killed(y) => ~alive(y))
(~killedy(y,x) | ~alive(y))

(eats(bill, peanuts) & alive(bill))
eats(bill,peanuts)
alive(bill)

(A x) (eats(bill, x) => eats(Sue, x))
(~eats(bill,x) | eats(Sue,x))

1.(~food(x) | likes(john,x))
2. food(apples)
3. food(chicken)
4. (~eats(y,x) | killed(y,x) | food(x))
5. (~killed(y, x) | ~alive(y))
6. eats(bill,peanuts)
7. alive(bill)
8. (~eats(bill,x) | eats(Sue,x))

C) John likes peanuts (likes(john,peanuts))

0. ~likes(john,peanuts)
9. ~food(peanuts) 0,1 {x/peanuts}
10. ~eats(y, peanuts) | killed(y, peanuts) 4,9 {x/peanuts}
11. killed(bill, peanuts) 6, 11 {y/bill}
12. ~alive(bill) 5, 11 {y/Bill, x/Peanuts}
13. alive(bill) 7
14. Contradiction 12,13

Therefore we have proven John likes Peanuts through resolution.

D) What food does Sue eat?
Assume Sue does not eat any food
0. ~eats(Sue,x)
9. ~eats(bill,x) 0, 8
10. ~eats (bill, peanuts) 9 {x/peanuts}
11. Contradiction 10, 6

Therefore we have proven Sue eats all food.

E)
If you don’t eat, you die.
If you die, you are not alive.
Bill is alive.

(A x,y) (~eat(y, x) => dead(y))
(A y) (dead(y) => ~alive(y))
alive(Bill)

CNF:
(eat(y,x) | dead(y))
(~dead(y) | ~alive(y))
alive(Bill)

Given:

1.(~food(x) | likes(john,x))
2. food(apples)
3. food(chicken)
4. (~eats(y,x) | killed(y,x) | food(x))
5. (~killed(y, x) | ~alive(y))
6. (eat(y,x) | dead(y))
7. (~dead(y) | ~alive(y))
8. alive(Bill)
9. (~eats(bill,x) | eats(Sue,x))

What food does Sue eat?

0. ~eats(sue,x)	
10. ~eats(bill,x)	0,9
11. dead(bill)	6,10 {y/bill}
12. ~alive(bill)	7, 11
13. Contradiction	8, 12

Therefore we can still prove that Sue eats everything.