1. (a) Neither

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Smoke | Fire |  |  |  |
| T | T | T | T | T |
| T | F | F | T | T |
| F | T | T | F | F |
| F | F | T | T | T |

(b) Neither

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Smoke | Fire | Heat |  |  |  |
| T | T | T | T | T | T |
| T | T | F | T | T | T |
| T | F | T | F | F | T |
| T | F | F | F | F | T |
| F | T | T | T | T | T |
| F | T | F | T | T | T |
| F | F | T | T | F | F |
| F | F | F | T | T | T |

(c) valid

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Smoke | Fire | Heat |  |  |  |
| T | T | T | T | T | T |
| T | T | F | T | T | T |
| T | F | T | F | F | T |
| T | F | F | T | T | T |
| F | T | T | T | T | T |
| F | T | F | T | T | T |
| F | F | T | T | T | F |
| F | F | F | T | T | T |

1. 2. We are able to prove unicorn is horned and magical, but we are not able to prove the unicorn is mythical.  
      To prove horned, we add   
      Resolve S9 with S7, giving: S10:  
      Resolve S9 with S8, giving: S11:  
      Resolve S11 with S6, giving: S12:mythical  
      Resolve S12 with S1, giving: S13: immortal  
      Resolve S10 with S13, empty clause.  
        
      To prove magical, we add S9:   
      Resolve S9 with S4, giving: S10:   
      Go through above procedure to get empty clause.
2. 1. {x/A, y/B, z/B}
   2. It is not unifiable, because x cannot be substituted by both A and B
   3. {y/John, x/John}
   4. It is not unifiable
3. 1. A x (Food(x) => Likes(John, x))

Food(Apples)

Food(Chicken)   
A x A y (Eats(x, y)&-Killed(x, y) => Food(y))  
A x (E y Killed(x, y)) => -Alive(x)  
Eats(Bill, Peanuts)&Alive(Bill)  
A x Eats(Bill, x) => Eats(Sue, x)

* 1. -Food(x) | Likes(John, x) (S1)  
     Food(Apples) (S2)  
     Food(Chicken) (S3)  
     -Eats(x, y) | Killed(x, y) | Food(y) (S4)  
     -Killed(x, y) | -Alive(x) (S5)  
     Eats(Bill, Peanuts) (S6)  
     Alive(Bill) (S7)  
     -Eats(Bill, x) | Eats(Sue, x) (S8)
  2. To prove John likes peanuts, we add S9: -Likes(John, Peanuts)  
     Resolve S9 with S1 {x/Peanuts}, giving S10: -Food(Peanuts)  
     Resolve S10 with S4 {y/Peanuts}, giving S11: -Eats(x, Peanuts) | Killed(x, Peanuts)  
     Resolve S11 with S5 {y/Peanuts}, giving S12: -Eats(x, Peanuts) | -Alive(x)  
     Resolve S7 with S12 {x/Bill}, giving S13: -Eats(Bill, Peanuts)  
     Resolve S13 with S6, empty clause.
  3. To figure out “what food does Sue eat?” (E x Food(x) & Eats(Sue, x)), we add its negation S9: -Food(x)|-Eats(Sue, x)  
     Resolve S9 with S8, giving S10: -Eats(Bill, x) | -Food(x)  
     Resolve S6 with S10 {x/Peanuts}, giving S11: -Food(Peanuts)  
     Go through the procedure in (c), getting an empty clause.  
     So, Sue eats Peanuts.
  4. Now, we have:  
     -Food(x) | Likes(John, x) (S1)  
     Food(Apples) (S2)  
     Food(Chicken) (S3)  
     -Eats(x, y) | Killed(x, y) | Food(y) (S4)  
     -Killed(x, y) | -Alive(x) (S5)  
     -Eats(Bill, x) | Eats(Sue, x) (S6)  
     A x A y -Eats(x, y)=>Die(x) = Eats(x, y) | Die(x) (S7)  
     A x Die(x) => -Alive(x) = -Die(x) | -Alive(x) (S8)  
     Alive(Bill) (S9)  
       
     We add S10: -Food(z) | -Eats(Sue, z)  
     Resolve S10 with S6 {x/z}, giving S11: -Eats(Bill, z) | -Food(z)  
     Resolve S11 with S7 {x/Bill, y/z}, giving S12: Die(Bill) | -Food(z)  
     Resolve S12 with S8 {x/Bill}, giving S13: -Alive(Bill) | -Food(z)  
     Resolve S13 with S9, giving S14: -Food(z)  
     Resolve S14 with S4 {z/y}, giving S15: -Eats(x, y) | Killed(x, y)  
     Resolve S15 with S5, giving S16: -Eats(x, y) | -Alive(x)  
     Resolve S16 with S9 {x/Bill}, giving S17: -Eats(Bill, y)  
     Resolve S17 with S7 {x/Bill}, giving S18: Die(Bill)  
     Resolve S18 with S8 {x/Bill}, giving S19: -Alive(Bill)  
     Resolve S19 with S9, empty clause.

So, Sue just eats everything Bill eats.