Dean Fleming

CMSC 471 – 01

HW #4

1.
   a. >>> tt_true(expr('P | ~P'))
      True
   b. >>> tt_true(expr('P >> P'))
      True
   c. >>> tt_true(expr('P >> (P | Q)'))
      True
   d. >>> tt_true(expr('(P | Q) >> P'))
      False
   e. >>> tt_true(expr('((A & B) >> C) <=> (A >> (B >> C))'))
      True
   f. >>> tt_true(expr('((A >> B) >> A) >> A'))
      True
2.
   a. >>> dpll_satisfiable(expr('P & Q'))
      {P: True, Q: True}
   b. >>> dpll_satisfiable(expr('A >> ~D & ~A & ~D'))
      {A: False, D: False}
   c. >>> dpll_satisfiable(expr('P >> ~P | P'))
      {P: True}
   d. >>> dpll_satisfiable(expr('~(P | ~P)'))
      False
3.
   a. P ^ Q ⊨ P
      true
   b. P ⊨ P ∧ Q
      false
   c. P ⊨ P v Q
      true
   d. P ⊨ ¬ ¬ P
      true
   e. P → Q ⊨ ¬ P → ¬ Q
      false
   f. ¬ P ⊨ P → Q
      true
   g. ¬ Q ⊨ P → Q
      false

h. P ∧ (P → Q) ⊨ Q
   true
i. (¬ P) ∧ (Q → P) ⊨ ¬ Q
   true

4.

a. There is no largest prime number.
   ~(Ex number(x) ^ prime(x) ^ (Ay number(y) ^ prime(y) -
   > x>=y))

b. Everything is either dead or alive.
   Ax dead(x) xor alive(x)

c. Dead things are not animate.
   Ax dead(x) => ~animate(x)

d. Zombies are not alive but they are animate.
   Az ~alive(z) ^ animate(z)

e. Good food is not cheap and cheap food is not good.
   Af good(f) => ~cheap(f) ^ cheap(f) => ~good(f)

f. John has exactly two brothers.
   Ej => (brothers(j) == 2)

g. No person can have two mothers.
   Ap ~Ep => (mothers(p) == 2)

h. If John has a sister, she is smart.
   Ej sister(j) => smart(j)

i. Every person is either male or female and no person
   can be both male and female.
   Ap male(p) xor female(p)

j. The enemy of your enemy is your friend.
   Ap (Ee ^ Ef f => enemy(e) ^ e => enemy(p)) => (f =>
   friend(p))

k. An ancestor of your ancestor is your ancestor.
   Ap (Ea ^ Eb b => ancestor(a) ^ a => ancestor(p)) => (b
   => ancestor(p))

5.

a. ∀x knows(x, x) ∧ likes(x, x)

   i. Everyone knows and likes himself

   ii. This can be rewritten as a horn clause

   iii. Set of clauses: [knows(x, x), likes(x, x)]

b. ∀x ∀y married(x, y) → loves(x, y) v hates(x, y)

   i. All couples who are married have one spouse (x) who loves and/or hates the other spouse (y).

   ii. Horn clause-able

   iii. [married(x, y), loves(x, y), hates(x, y)]

c. ∀x ∀y loves(x, y) ↔ loves(y, x)

   i. For all couples, partner x loves partner y if and only if y loves x.

   ii. Horn clause-able

   iii. [loves(x, y), loves(y, x)]

d. ∀x ∀y dating(x, y) v engaged(x, y) → knows(x, y) ∧ likes(x, y)

   i. For all couples who are dating or engaged, partner x knows and likes partner y.

   ii. Not Horn clause-able

   iii. [dating(x, y), engaged(x, y), knows(x, y), likes(x, y)]

e. ∀x ∀y loves(x, y) → ¬ hates(x, y)

   i. For all couples who are in love, partner x does not hate partner y.

   ii. Horn clause-able

iii. [loves(x, y), hates(x, y)]

f. ∀x ∀y ¬ knows(x, y) → ¬ likes(x, y)

i. For all couples where partner x does not know partner y, x does not like y.

ii. Horn clause-able

iii. [knows(x, y), likes(x, y)]

g. ∀x ∃y knows(x, y) ∧ hates(x, y)

i. For all x there exists a y where x knows y and x hates y.

ii. Horn clause-able

iii. [knows(x, y), hates(x, y)]

h. ∃y ∀x knows(x, y) ∧ hates(x, y)

i. There exists a y for all x where x knows y and x hates y.

ii. Horn clause-able

iii. [knows(x, y), hates(x, y)]

i. ¬ (∀x loves(x, x))

i. For no x does x love their self.

ii. Not Horn clause-able

iii. [loves(x, x)]

j. ¬ (∃x ∀y knows(x, y))

i. There is no x for all y where x knows y.

ii. Not Horn clause-able

iii. [knows(x, y)]