Gabrielle Boehmer, boehmer1@umbc.edu

Homework Four

Check Validity

1. P v ¬P
   >>> tt_true(expr('P | ~P'))
   True
2. P → P

   >>>tt_true(expr('P >> P'))

   True
3. P→ (P v Q)

   >>> tt_true(expr('P >> (P | Q)'))

   True
4. (P v Q) → P

   >>>tt_true(expr('(P | Q) >> P'))

   False
5. ((A ∧B) →C) ↔ (A → (B → C))

   >>>tt_true(expr('((A & B) >> C) % (A >> (B >> C))'))

   True
6. ((A →B) →A) →A

   >>>tt_true(expr('((A >> B) >> A) >> A'))

   True

Satisfiability

1. P ∧ Q
   >>> dpll_satisfiable(expr('P & Q'))
   {P: True, Q: True}

2. ALIVE→ ¬DEAD ∧ ¬ALIVE ∧ ¬DEAD

   >>>dpll_satisfiable(expr('P >> ~Q & ~P & ~Q'))

   {P: False, Q: False}

3. P → ¬ P v P

   >>>dpll_satisfiable(expr('P >> ~P | P'))

   {P: True}

4. ~ (P v ¬ P)

   >>>dpll_satisfiable(expr('~(P | ~P)'))

   False

Propositional Consequence

1. P ∧ Q ⊨ P
   true

2. P ⊨ P ∧ Q

   >>>tt_entails(expr('P'), expr('P & Q'))

   False

3. P ⊨ P v Q

   >>>tt_entails(expr('P'), expr('P | Q'))

   True

4. P ⊨ ¬ ¬ P

   >>>tt_entails(expr('P'), expr('~~P'))

   True

5. P → Q ⊨ ¬ P → ¬ Q

   >>>tt_entails(expr('P >> Q'), expr('~P >> ~Q'))

False

6. $\neg P \models P \rightarrow Q$

   >>>tt_entails(expr(‘~P’), expr(‘P >> Q’))

   True

7. $\neg Q \models P \rightarrow Q$

   >>>tt_entails(expr(‘~Q’), expr(‘P >> Q’))

   False

8. $P \wedge (P \rightarrow Q) \models Q$

   >>>tt_entails(expr(‘P & (P >> Q)’), expr(‘Q’))

   True

9. $(\neg P) \wedge (Q \rightarrow P) \models \neg Q$

   >>>tt_entails(expr(‘(~P) & (Q >> P)’), expr(‘~Q’))

   True

English to FOL

1. There is no largest prime number.

   ~(Ex number(x) ^ prime(x) ^ (Ay number(y) ^ prime(y) -> x >= y))
   It is not true that there is a prime number and it is greater than or equal to all prime numbers

2. Everything is either dead or alive.

   (Ax) dead(x) v alive(x)

   For all things, it is either dead or alive

3. Dead things are not animate.

   (Ax) dead(x) => ~animate(x)

   All dead things are not animate.

4. Zombies are not alive but they are animate.

   (Ax) zombies(x) => ~alive(x) ^ animate(x)

   All zombies are not alive and are animate.

5. Good food is not cheap and cheap food is not good.

   (Ax) (goodFood(x) => ~cheap(x)) ^ (cheapFood(x) => ~good(x))

   All good food is not cheap and all cheap food is not good.

6. John has exactly two brothers.

   (Ex)(Ey) brother_of(x, John) ^ brother_of(y, John) ^ ~(x = y) ^ (Az) (brother_of(z, John) => ((x = z) v (y = z))

   There exists one person who is a brother of John and another person who is a brother of John, and these are not the same people, and for all other brothers of John, they must be either the first or second brother.

7. No person can have two mothers.

   ~((Ax)(Ay)(Az) person(x) ^ mother_of(y, x) ^ mother_of(z, x) ^ ~(y = z))

   There does not exist a mother of someone and another mother of the same someone and the mothers are not the same person

8. If John has a sister, she is smart.

   (Ax) sister_of(x, John) => smart(x)

   For all things, if it is a sister of John, then it is smart.

9. Every person is either male or female and no person can be both male and female.

   (Ax) (person(x) => male(x) v female(x)) ^ ~(person(x) => male(x) ^ female(x))

   Every person is either a male or a female and it is not true that a person is male and female.

10. The enemy of your enemy is your friend.

    (Ay)(Ae)(Af) ((is_against(y, e) ^ is_against(e, f)) => is_for(y, f))

    You are against your enemy and your enemy is against your friend; therefor, you are for your friend.

11. An ancestor of your ancestor is your ancestor.

    (Ay)(An)(Ac) ancestor_of(n, y) ^ ancestor_of(c,n) => ancestor_of(c,y)

    If n is the ancestor of you and c is the ancestor or your ancestor, then the ancestor of your ancestor is your ancestor.

CNF and horn clauses

1. ∀x knows(x, x) ∧ likes(x, x)
   (a) Everyone knows and likes himself;
   (b) this can be rewritten as a horn clause;
   (c) set of clauses: [knows(x, x), likes(x, x)]

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

   (a) Everyone married either loves or hates their spouse;
   (b) This cannot be rewritten as a horn clause (two non-negatives);
   (c) ~married(x,y) | loves(x,y) | hates(x,y)
       Set of Clauses: [loves(x,y) | ~married(x, y) | hates(x, y)]

3. ∀x ∀y loves(x, y) ↔ loves(y, x)

   (a) Everyone loves someone if and only if they are loved by that someone;
   (b) This can be rewritten as a horn clause, because the clauses are separated;
   (c) (Loves(x,y) v ~loves(y,x)) & (~loves(x,y) v loves(y,x))
       Sets of clauses: [loves(x, y) v ~loves(y, x), loves(y, x) v ~loves(x, y)]

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

   (a) Everyone who is either dating or engaged knows and likes the person they are dating or engaged to;

   (b) This cannot be rewritten as a horn clause

   (c) ((~dating(x,y) | knows(x,y)) & (~engaged(x, y) | knows(x,y)) & (~dating(x, y) | likes(x, y)) & (~engaged(x, y) | likes(x, y)))

       Sets of clauses: [~dating(x, y) v knows(x, y) , ~engaged(x, y) v knows(x, y) , likes(x, y)]

5. ∀x ∀y loves(x, y) → ¬ hates(x, y)

   (a) Everyone who loves someone does not hate that person;

   (b) This can be rewritten as a horn clause (no non-negatives).

(c) ~hates(x,y) v ~loves(x,y)

Set of clauses: [~hates(x, y) v ~loves(x, y)]

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

(a) Everyone who does not know someone does not like that someone;

(b) This can be rewritten as a horn clause (only one nonnegative)

(c) ~likes(x,y) v knows(x,y)

Sets of clauses: [~likes(x, y) v knows(x, y)]

7. ∀x ∃y knows(x, y) ∧ hates(x, y)

(a) Everyone knows and hates someone;

(b) This can be rewritten as a horn clause (clauses can be separated);

(c) Knows(x,f(x)) & hates(x,f(x))

Sets of clauses[knows(x, f(x)), hates(x, f(x))]

8. ∃y ∀x knows(x, y) ∧ hates(x, y)

(a) There exists someone that everyone knows and everyone hates;

(b) This can be rewritten as a horn clause (clauses can be separated);

(c) Knows(x,P) & hates(x,P)

Sets of clauses[knows(x, P), hates(x, P)]

9. ¬ (∀x loves(x, x))

(a) It is not true that everyone loves themselves

(b) This can be rewritten as a horn clause (no non-negatives)

(c) Same as Ex ~loves(x,x) (There exists someone who does not love himself.)

~loves(P, P)

10. ¬ (∃x ∀y knows(x, y))

(a) It is not true that there exists someone that knows everyone.

(b) This can be rewritten as a horn clause (no non-negatives)

(c) Same as AxEy ~knows(x,y) (Everyone does not know someone)

~knows(x, f(x))

Extra Credit

Which answer in this list below is the correct answer to this question?

1. All of the below.
2. None of the below.
3. All of the above.
4. One of the above.
5. None of the above.
6. None of the above.

Explain your reasoning by (a) mapping the problem into propositional logic and (b) showing how the AIMA code can be used to solve this problem.

5 if the correct answer to the question.

(a)

If 1 is true:

1 => 2 & 3 & 4 & 5 & 6

But 2 & 3 contradict each other so 1 cannot be true

If 2 is true:

2 => ~3 & ~4 & ~5 & ~6

But this does not hold with 4 so 2 cannot be true

If 3 is true:

3 => 1 & 2

But 1 and 2 do not line up with this so 3 cannot be true

If 4 is true:

4 => 1 v 2 v 3

But we already determined that 1,2, and 3 are not correct, so 4 cannot be correct.

If 5 is true

5 => ~1 & ~2 & ~3 & ~4

Since we already determined that 1,2,3, and 4 are not correct then the negation of them has to be correct. So, 5 is correct.

If 6 is true

6 => ~1 & ~2 & ~3 & ~4 & ~5

But since we determined 5 to be true therefore ~5 is not correct. So, 6 is not correct

This knowledge can be formalized to show the problem in propositional logic of better syntax. To do this, let a, b, c, d, e, f represents each answer respectively. Then:

1. a ⇔ b ^ c ^ d ^ e ^ f
2. b ⇔ ~c ^ ~ d ^ ~e ^ ~f
3. c ⇔ a ^ b
4. d ⇔ (a ^ ~b ^ ~c) v (~a ^ b ^ ~c) v (~a ^ ~b ^ c)
5. e ⇔ ~a ^ ~b ^ ~c ^ ~d
6. f ⇔ ~a ^ ~b ^ ~c ^ ~d ^ ~e

Prolog needs to be used in aima in order to solve this problem. This is because prolog explores the possible proofs independently. So, it is able to find a solution and not just accept negation as failure.