Your Name, YourID@umbc.edu

CMSC 471 HW4

Total points: 100

1. Checking Validity (10 points, 2 each for 1.2-1.6)

2. p → p
>>> tt_true(expr('P >> P'))

3. p → (p v q)
>>> tt_true(expr('P >> (P | Q)'))

4. (p v q) → p
>>> tt_true(expr(' (P | Q) >> P'))

5. ((A ∧B) →C) ↔ (A → (B → C))
>>> tt_true(expr(' ((A & B) >> C) <=> (A >> (B >> C)) '))

6. ((a → b) → a) → a
>>> tt_true(expr(' ((A >> B) >> A) >> A '))

2. Satisfiability (9 points, 3 each)

2. ALIVE→ ¬DEAD ∧ ¬ALIVE∧ ¬DEAD
>>> dpll_satisfiable(expr('Alive>>~Dead&~Alive&~Dead'))

3. P → ¬ P v P
>>> dpll_satisfiable(expr('P>>~P|P'))

4. ~ (P v ¬ P)
>>> dpll_satisfiable(expr('~(P|~P)'))

3. Propositional Consequence (24 points, 3 each)

2. p |= p ^ q

3. p |= p v q

4. p |= ~~p

5. p-->q |= ~p-->~q

6. ~p |= p-->q

7. ~q |= p-->q

8. p, p-->q |= q

9. ~p, q-->p |= ~q

4. English to FOL (30 points, 3 each)

There are usually several reasonable ways to express natural language sentences in logic. One source of variation is what to leave implicit and what to make explicit, e.g., omitting obvious types as in rendering 'every person loves their mother' as Ax mother(x,y) => loves(y,x). Another comes from the application of standard tautologies, e.g., you can encode 'no man is an island' as ~Ex man(x) ^ island(x) or as Ax man(x) => ~island(x).

2. Everything is either dead or alive.

4. Zombies are not alive but they are animate

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

6. John has exactly two brothers.

7. No person can have two mothers.

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

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

10. The enemy of your enemy is your friend.

11. An ancestor of your ancestor is your ancestor.

5. CNF and horn clauses (27 points, 3 each for 5.2-10)

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

3. ∀x ∀y loves(x, y) ↔ loves(y, x)

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

5. ∀x ∀y loves(x, y) → ¬ hates(x, y)

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

7. ∀x ∃y knows(x, y) ^ hates(x, y)

8. ∃y ∀x knows(x, y) ^ hates(x, y)

9. ¬ (∀x loves(x, x))

10. ¬ (∃x ∀y knows(x, y))