CSE 5360 003 – Fall 2018
Exam 2, Monday 10/29/2018

Name: SOLUTION

Student ID:

(Not providing this information: -10 Points)
(Name missing in Individual pages: -5 Points)

Name: ___________________________

Total Exam Points: 100

Score

Question	Points	Max Points
1		15
2		8
3		10
4		15
5		12
6		10
7		15
8		15
9 (Opt.)		10*
Total		**100 (110*)**

*: Extra Credit Points

Name: ______________________

Question 1 – 15 points

Convert the following statements into conjunctive normal form:

(a) **[5 points]**:

(B OR (NOT C)) ⇒ A

Remove ⇒

NOT(B OR(NOT C)) OR A

DeMorgans

(NOT B AND NOT(NOT C)) OR A

Double Negation

(NOT B AND C) OR A

Dist. Prop.

(NOT B OR A) AND (C OR A)

(b) **[5 points]**:

(A ⇒ B) ⇔ C

Remove ⇔

((A ⇒ B) ⇒ C) AND (C ⇒ (A ⇒ B))

Remove ⇒

(NOT(NOT A OR B) OR C) AND (NOT C OR(NOT A OR B))

DeMorgans

((NOT (NOT A) AND NOT B) OR C) AND (NOT C OR(NOT A OR B))

DOUBLE NEGATION

((A AND NOT B) OR C) AND (NOT C OR (NOT A OR B))

Dist. Prop.

((A OR C) AND (NOT B OR C)) AND (NOT C OR (NOT A OR B))

FLATTEN

(A OR C) AND (NOT B OR C) AND (NOT C OR NOT A OR B)

(c) **[5 points]**:
A OR (NOT (B ⇒ C))

Remove ⇒

A OR (NOT (NOT B OR C))

DeMorgans

A OR (NOT(NOT B) AND ~~N~~NOT C)

Double Negation

A OR (B AND NOT C)

Dist Prop.

(A OR B) AND (A OR NOT C)

Name: ______________________

Question 2 – 8 points

For each pair of statements, determine if you can apply resolution to those statements. If you can apply resolution, write at least one statement that can be derived by applying resolution to the given pair of statements. If you can apply resolution, apply it blindly; **do NOT simplify the resulting sentences**. If you cannot apply resolution, just write "cannot apply".

(a) **[2 points]**:
Statement 1: A OR (~~NOT~~ E)
Statement 2: B OR D OR ~~E~~

A OR B OR D

(b) **[2 points]**:
Statement 1: A OR (~~NOT~~ B) OR C
Statement 2: ~~B~~ OR (NOT C) OR D

A OR C OR (NOT C) OR D.

(c) **[2 points]**:
Statement 1: A OR B OR (NOT D)
Statement 2: A OR C OR (NOT D)

Cannot apply

(d) **[2 points]**:
Statement 1: A OR (~~NOT~~ B)
Statement 2: A OR ~~B~~ OR (NOT C)

A OR A OR (NOT C)

Question 3 – 10 points

You have a KB and two sentences S1 and S2 that are defined in a domain containing 5 symbols. You check the following using TT-ENTAILS:

KB ⊨ S1, KB ⊨ S2, KB ⊨ (S1 ∨ S2) and KB ⊨ (S1 ∧ S2)

How many calls to TT-CHECK-ALL are required in total.

If KB ⊨ S1 is true and KB ⊨ S2 is also true does that guarantee that KB ⊨ (S1 ∨ S2) and KB ⊨ (S1 ∧ S2) are also true? Justify your answer.

Each call to TT-ENTAILS requires $(2^n + 1)$ calls to TT-CHECKS ALL,

Here $n = 5$ so in total we need.

$4 \cdot \times (2^5 + 1) = 4 \times 33 = 132$ calls

If KB ⊨ S1 is true then whenever KB is true then S1 is true. Similarly whenever KB is true, S2 is true. So whenever KB is true, S1 ∨ S2 & S1 ∧ S2 are also true.

So, KB ⊨ S1 ∨ S2 & KB ⊨ S1 ∧ S2

Name: ____________________

Question 4 – 15 points

John, Harry and Bob sign the following contract on December 31, 2018.

- *Bob will give John \$500 if and only if John gets a 4.0 GPA in Spring 2019*
- *If Bob gives John \$500, John will give Harry a guitar*
- *If John gave Harry a guitar, Harry will play at the event*

The all-inclusive sequence of events at the end of Spring 2019 are the following:

John got a 4.0 GPA. Bob gave John \$500. Harry played at the event.

(**HINT:** All-inclusive means anything not mentioned here did not happen).

(a) **[8 points]**: Describe the contract using first-order logic. You also need to list the constants, predicates used and the semantics [*father(x,y): x is father of y*].

Predicates.

Give (x, y, z) : x gives y to z

GetA (x) : x gets 4.0 GPA in Sprig 2019.

PlayE (x) : x plays at event

Constants: Bob, John, 500, Guitar

Variables: x, y, z

Contract

$Give(Bob, 500, John) \Leftrightarrow GetA(John)$

$Give(Bob, 500, John) \Rightarrow Give(John, Guitar, Harry)$

$Give(John, Guitar, Harry) \Rightarrow PlayE(Harry)$

(b) **[7 points]**: Describe what happened using first order logic and using notation consistent with your answer in Question 4(a).

$$GetA(John) \wedge Give(Bob, 500, John) \wedge [not(Give(John, Guitar, Harry))] \wedge PlayE(Harry)$$

(c) **[5 points]**: Was the contract violated or not? Justify your answer.

Yes. While Bob gave John \$500, John did not give a Guitar to Harry. This is a violation of the second part of the contract.

Name: ____________________

Question 5 – 12 points

Give the unifier for the following predicates (if possible). If no unifier possible, Justify.

(a) **[3 points]**: Father(x, Bob), Son(Bob, John)

None Possible. [Father and Son are different predicates]

(b) **[3 points]**: Gave(x, Laptop, Mary), Gave(Mother(Mary), Laptop, y)

{x/ Mother(Mary), y/ Mary}

(c) **[3 points]**: TallerThan(x, Barry), TallerThan(Duncan, y)

{x / Duncan, y/Barry)

(d) **[3 points]**: Grouped(Tom, Richard, Harry), Grouped(x, Son(y), z)

None Possible. No assignment available for y.

Question 6 – 10 points

Suppose you have a domain in First-Order Logic that has the following:

- 10 constants
- 1 predicate of 3 arguments
- 2 predicates of 2 arguments
- 3 predicates of 1 argument

How many symbols do you need to convert any KB in this domain into a propositional logic KB. Justify your answer.

Possible assignents

1 predicate with 10^3 assignents.

2 predicates with 10^2 assignents.

3 predicates with 10^1 assignents.

Total

$10^3 + 2 \times 10^2 + 3 \times 10^1 = 1000 + 200 + 30$

$= 1230$ assignents.

So ~~we~~ need 1230 symbols.

Question 7 – 15 points

Consider this first-order logic knowledge base:

big(car) ∧ small(bike)
∀x [small(x) ⇔ fast(x)]
fast(bike) ⇒ slower(car, bike)

big, small, fast, slower.

In this first-order logic knowledge base, ~~fast, slower, slow~~ are predicates, and car, bike are constants. Convert this first-order logic knowledge base into a propositional logic knowledge base, by performing the following two steps:

(a) Define symbols for the propositional-logic version of the knowledge base and specify what their equivalents are in the original first-order logic knowledge base.
(b) Define the statements that should be stored in the propositional-logic version of the knowledge base.

The symbols you define should be comprehensive enough to allow us to translate any well-defined inference problem in the original knowledge base to an equivalent problem for the propositional knowledge base. Anything that we can infer from the original first-order logic knowledge base we should also be able to infer from the propositionalized knowledge base, and vice versa.

(a) big(car) : big_car   big(bike) : big_car
small(car) : small_car   small(bike) : small-bike
fast(car) : fast_car   fast(bike) : fast_bike
slower(car, car) : slower_car_car
slower(car, bike) : slower_car_bike.
slower(bike, bike) : slower_bike_bike
slower(bike, car) : slower_bike_car

(b) Universally Instantiation.

big(car) ∧ Small(bike)

Small(bike) ⟺ fast(bike)

Small(Car) ⟺ fast(car)

fast(bike) ⟹ slower(Car, bike).

Converting to Propositional Logic:

big_car ∧ small_bike.

small_bike ⟺ fast_bike.

Small_car ⟺ fast_car

fast_bike ⟹ slower_car_bike.

Name: ____________________

Question 8 – 15 points

Is the propositional logic KB obtained in Question 7(b) in Horn Form? If it is not can it be converted into Horn Form. If it can be, use the Horn form representation and Backward Chaining to show that his knowledge base entails slower(car, bike) [Note: Use the symbols obtained in Question 7(a) to convert this statement into propositional logic too. ~~Also Modus Ponens can also be used with~~

Yes it can be written in Horn form:

big_car

Small_bike.

Small_bike ⇒ fast_bike.

fast_bike ⇒ Small_bike.

Small_car ⇒ fast_car.

fast_car ⇒ Small_car

fast_bike ⇒ slower_car_bike.

Using Backward chaining:

| slower_car_bike |

fast_bike ⇒ slower_car_bike

add fast_bike

fast_bike.
slower_car_bike

Small_bike ⇒ fast_bike.

add small_bike.

Small_bike is already true,

So you can use small_bike ⇒ fast_bike &
Small_bike to show fast_bike. by Modus Ponens

Now use fast_bike ⇒ slower_car_bike
& fast_bike to show slower_car_bike
by Modus Ponens

So slower_car_bike is true.

KB ⊨ slower_car_bike.

Name: ______________________

Question 9 (OPTIONAL) – 10 points Extra Credit

```
function PL-RESOLUTION(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
      for each Ci, Cj in clauses do
         resolvents ← PL-RESOLVE(Ci, Cj)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

The box above shows the PL-RESOLUTION function as defined in the textbook.

(a) **[5 points]**:
This function refers to the empty clause. Specify what this empty clause is, in the context of PL-RESOLUTION. Remember, a propositional logic sentence is defined by defining its symbol (if any), its connective (if any) and its sub expressions (if any). Thus, fully specify what values the empty clause will contain for the symbol, connective, and sub expressions.

Symbol : None.
Connective : None
Sub-expressions : None.

(b) **[5 points]**:
What can you say about the truth value of the empty clause referred to in the PL-RESOLUTION pseudo code? Is it always true, always false? Or sometimes true and sometimes false? Justify your answer.

The empty clause is inherently false.

Name: ____________________________

SCRATCH