CSE 4308/5360 001 – Fall 2018
Exam 2, Monday 10/29/2018

Name: SOLUTION

Student ID:

Course:

- ❑ **4308 – 001**
- ❑ **5360 – 001**

(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) **[6 points]**:
(NOT (C OR B)) ⇔ A

Remove ⇔

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

Remove ⇒

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

Double Negation.

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

De Morgans

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

Dist. Prod.

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

Flattening

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

(NOT A OR NOT B)

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

A OR (NOT (B ⇒ C))

Remove ⇒

A OR (NOT (NOT B OR C))

DeMorgans

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

DN

A OR (B AND NOT C)

Dist. Prop.

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

(c) **[4 points]**:

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

DEMORGANS.

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

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

DN.

(NOT A OR B OR C) AND

(NOT A OR NOT B OR 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 Resolve.

(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

Let S1 and S2 be two logical statements in propositional logic. Write pseudocode that determines if the statement (S1 XOR S2) is valid (a statement is valid if it is true in all models). The XOR of the two statements S1 and S2 is true in two cases:

1) If S1 is true, then S2 is false
2) If S1 is false, then S2 is true.

Your pseudocode must use TT-ENTAILS.

(**HINT**: TT-ENTAILS (A, B) checks if whenever A is true then is B true)

```
XOR(S1, S2)
{
    return TT-ENTAILS(S1, not(S2))
                AND TT-ENTAILS(not(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. John gave Harry a guitar.

(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 Spring 2019

PlayE (x) : x plays at event.

Constants: Bob, John, 500, Guitar.

Variables : x, y, z

Contract

Give (Bob, 500, John) ⇔ GetA (John)

Give(Bob, 500, John) ⇒ Give (John, Guitar, Harry)

Give(John, Guitar, Harry) ⇒ 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) ∧ Give(Bob, 500, John) ∧ Gave(John, Guitar, Harry) ∧ [not (PlayE(Harry))]

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

Yes. While John gave a Guitar to Harry, Harry did not play at the event. So Third part of contract was violated.

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. Cannot find value for y.

Question 6 – 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.

For each call to TT-ENTAILS you would need 2^5+1 calls to TT-CHECK-ALL.

So in total you would need $4 \times (2^5+1)$ calls.

KB ⊨ S1 & KB ⊨ S2 means that for all rows when KB is true, S1 & S2 are true. This means that S1 ∧ S2 and S1 ∨ S2 are also true. So KB entails those scentences too.

Name: __________________________

Question 7 – 15 points

Consider this first-order logic knowledge base:

$$\text{big(car)} \land \text{small(bike)}$$
$$\forall x\ [\ \text{small}(x) \Leftrightarrow \text{fast}(x)]$$
$$\text{fast(bike)} \Rightarrow \text{slower(car, bike)}$$

In this first-order logic knowledge base, big, small, fast, slower. ~~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.

$big(car) \wedge Small(bike)$

$Small(bike) \Leftrightarrow fast(bike)$

$Small(Car) \Leftrightarrow fast(car)$

$fast(bike) \Rightarrow slower(Car, bike)$

Converting to Propositional Logic:

big_car $\wedge$ small_bike.

small_bike $\Leftrightarrow$ fast_bike.

Small_car $\Leftrightarrow$ fast_car

fast_bike $\Rightarrow$ slower_car_bike.

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 $\models$ 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