ASSIGNMENT 5

Problem 1

function CHECK_TRUE(S1) return true or false

symbols ← a list of proposition symbols in KB

return check_model(S1, symbols, []);

function check_model(S1, symbols, model)

if EMPTY?(symbol) then

if PL-True?(S1, model) then return true;

else

return false;

else do

P ← FIRST(symbols); rest ← REST(symbols)

return CHECK1_TRUE

return check_model(S1, rest, model ∪ {P=true})

or check_model(S1, rest, model ∪ {P=false})

Problem 2 :

function CHECK_EQUIVALENCE (KB1, KB2) return true or false

symbols ← a list of proposition symbols in KB1 & KB2

return TT_CHECK_ALL (KB1, KB2, symbols, [ ])

function TT_CHECK_ALL (KB1, KB2, symbols, model) return true or false

if Empty? (symbols) then
    if PL-True? (KB1, model) then
        if PL-True? (KB2, model)
            return true
        else
            return false
    else
        if PL_True? (KB2, model)
            return false ;
        else return true
else do
    P ← First (symbols); rest ← Rest (symbols)
return TT_CHECK_ALL (KB1, KB2, rest, Extend (P, true, model)) and
TT_CHECK_ALL (KB1, KB2, rest, Extend (P, false, model))

Problem 3

(a) $KB \models \alpha$, According to the definition of entailment knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true.

In this case, $KB \not\models S1$, because when KB is true S1 is also true.
In the truth table, row 1, KB is true where S1 is true and in row 3, KB is true & S1 is also true.

(b) From the information given, $\neg(KB)$ does not entail $\neg(S1)$, because in the truth table row 2, when $\neg KB$ is true, $\neg S1$ is False, similarly in row 4, when $\neg KB$ is true, $\neg S1$ is false. Therefore $\neg(KB)$ does not entail $\neg(S1)$.

Problem 4 :

Convert the following to CNF.

Part a: $A \Rightarrow (\text{NOT}\ (C\ \text{or}\ B))$

Soln: $\neg A \vee (\neg (C \text{ or } B))$

$\neg A \vee (\neg C \wedge \neg B)$

$(\neg A \vee \neg C) \wedge (\neg A \vee \neg B)$

Part b: $(\text{NOT}\ (C \text{ OR } B)) \Rightarrow A$

Soln: $\neg (C \vee B) \Rightarrow A$

$(\neg C \wedge \neg B) \Rightarrow A$

$\neg(\neg C \wedge \neg B) \vee A$

$C \vee B \vee A$

Problem 5

function CHECK-XOR-VALID (S1, S2) return true or false

symbols ← a list of proposition symbols in S1 and S2

return TT_CHECK_ALL (S1, S2, symbols, model[])

function TT_CHECK-ALL(S1, S2, symbols, model) return true or false.

if Empty?(Symbols) then
  if PL-True? (S1, model) then
    if PL_True?(S2, model)
      return false
    else return true;
  else
    if PL_True? (S2, model) return true
    else return false
else do
  P ← FIRST (symbols); rest ← Rest (symbols)
return TT_CHECK_ALL (S1, S2, rest, EXTEND(P, true, model)) and
  TT_CHECK_ALL (S1, S2, rest, EXTEND(P, false, model))

Problem 6 :

* dog(shadow)
  - It is a predicate, which returns true for shadow is dog.
  - shadow is constant
* gives (John, shadow, Mary)
  - gives is a predicate.
  - It returns true for john gives shadow to Mary.
  - john, shadow, Mary is constant
* male(shadow) ∧ dog(shadow) ⟹ gives (Mary, smartphone, John)
  - male, dog, gives is a predicate.
  - male (shadow) means that shadow is male,
  - dog(shadow) means that shadow is dog
  - gives (Mary, smartphone, John) means mary gives smartphone to john.
  - shadow, john, mary, smartphone - constant
* female (shadow) ∧ dog(shadow) ⟹ gives(Mary, laptop, John)
  - female, dog, gives is predicate.
  - female (shadow) means that shadow is female
  - dog(shadow) means shadow is dog.
  - gives (mary, laptop, john) means mary gives laptop to yohn.
  - shadow, mary, laptop, john is constant

* $\forall x \exists y$: gives (John, x, y) $\wedge$ people(x) $\wedge$ dog(y) $\wedge$ male(y).

- For all x as people ~~to~~, x is variable,
- There exist y dog, y is variable.
- people(x) is predicate, x is people.
- dog(y) is predicate, it returns true for y is dog
- male(y) is predicate, it returns true for y is dog.
- John is constant

* gives (mary, laptop, john)
- gives is predicate, it means that mary gave laptop to john.
- mary, john, laptop is constant.

Problem 7

1) Symbols :

taller_John_Bill

taller_John_John

taller_Bill_John

taller_Bill_Bill

It is given for every x, $taller(x, Bill) \Rightarrow tall(x)$, so x can be any person, x = 'Person'.

taller_Person_Bill

taller_Bill_Person

taller_Person_Person

tall_Person

2) taller (John, Bill)

John is than Bill

Propositional version : taller_John_Bill

For every x, $taller(x, Bill) \Rightarrow tall(x)$

Considering x as Person,

$taller\_Person\_Bill \Rightarrow tall\_Person$

$\neg(taller\_Person\_Bill) \vee tall\_Person$