CSCE 420, Assignment A5

due: Tues, Apr 26, 3:55pm – THIS IS A HARD DEADLINE – NO GRACE PERIOD - LATE ASSIGNMENTS TURNED IN AFTER Tues, Apr 26, 3:55pm WILL BE WORTH 0.

**turn-in answers as a Word document (A5.docx) and commit/push it to your TAMU GitHub**

1. Translate the following sentences into First-Order Logic. Remember to break things down to simple concepts (with short predicate and function names) and make use of quantifiers. For example, don’t say “tasteDelicious(someRedTomatos)”, but rather: “x tomato(x)^red(x)^ taste(x, delicious)”. See the lecture slides for more examples and guidance.

Tomatoes are either a fruit or vegetable.

Because Tomato can either be fruit or vegetable but not both!

Some mushrooms are poisonous.

No student at TAMU likes every class.

For every X, student(X), tamu(X) -> Some Y, classes(Y) ^ dislikes(X,Y)

Every king has a crown and some subjects. (How many is not specified)

For every X, king(X), has\_crown(X) -> For some Y, subject(Y)

An isosceles triangle is a triangle with 2 sides of equal length (but not 3).

For all X, Y, Z, side(X), side(Y), side(Z), triangle(side(X), side(Y), side(Z)) -> (equals(X, Y) ^ ~equals(X, Z)) v (equals(Y, Z) ^ ~equals(X, Y)) v (equals(X, Z) ^ ~equals(X, Y))

The left front tire of Henry’s car is flat.

For some t, c, Car(c) ^ owner(Henry, c) ^ Tire(t) ^ count(Tire(t)) = 4 ^ flat(t) ^ partOf(t, leftfront(c))

All disk drives are electronic; they are either mechanical (HDDs) or solid-state (SSDs). Solid-state drives are faster than mechanical drives, in terms of read-access time (e.g., ms per byte), but SSDs are also more expensive (in dollars per Gb).

For all X, DiskDrive(x) ^ Electronic(x) -> Mechanical(x) V SolidStateDrive(x)

For all X,Y SSD(X), MechanicalDrive(Y)-> ReadSpeed (X) > ReadSpeed (Y)

For all X,Y SSD(X), MechanicalDrive(Y) –> cost(X) > cost(Y)

All laptops sold by Dell in 2022 have at least 4GB of RAM and cost at least $1000.

For all X, laptop(X), sellingYEAR(2022), sold(DELL,X) -> ramInGB(X) >= 4 ^ costInDollar(X) > 1000

John’s favorite Sci-Fi movie is Star Wars (hint: instead of saying *favoriteSciFiMovie(John,StarWars)* which is too complex, break it down into simpler concepts such as predicate ‘likes(P,M)’, and function ‘score(P,M)’, which denotes P’s internal rating system for things (e.g., 1-100). Ensure you are comparing SciFi movies, because a person might use a different rating system for other items, such as toasters.

For all X, movie(X) ^ sciFi(X) -> score(John,X) <= score(John,starwars)

2. Determine whether or not the following pairs of predicates are unifiable. If they are, give the most-general unifier and show the result of applying the substitution to each predicate. If they are not unifiable, indicate why. Terms that are variables are in capital letters; constants and function names are lowercase. For example, ‘loves(A, hay)’ and ‘loves(horse, hay)’ are unifiable, the unifier is u={A/horse}, and the unified expression is ‘loves(horse, hay)’ for both.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0) | loves(A, hay) | loves(horse, hay) | yes | u={A/horse} | loves(horse, hay) |
| 1) | p(a, X) | p(b, X) | no | reason: regardless of X, a and b conflict |  |
| 2) | p(a, X, f(g(Y))) | p(Z, f(Z), f(U)) | |  |  |  | | --- | --- | --- | | yes | u={Z/a, X/f(a), U/g(Y)} | p(a, f(a), f(g(Y))) | | no | Because if we unify the Y to f(a), we cannot unify it to g(X). |  | | yes | u={Z/f(Y), Y/f(a), X/f(V)} | r(f(f(a)), f(a)), f(V) | | yes | u={Z/a, Y/f(b), X/b} | p(a, f(b), f(b)) | | yes | u={Y/g(f(a)), X/f(a), Z/f(W)} | q(g(f(a)), g(f(a)), f(W)) | | no | Because X/b then X/a is not unifiable |  | | yes | u={X/f(a, a), V/f(f(a, a),f(a, a)), Y/Z} | q( f(a, a),  f( f(a, a), f(a, a) ),  Y) | | | |
| 3) | q(f(a), g(X)) | q(Y, Y) |
| 4) | r(f(Y), Y,X) | r(Z, f(a), f(V)) |
| 5) | p(a, Y, f(X)) | p(Z, f(b), f(b)) |
| 6) | q(g(f(a)), g(X), Z) | q(Y, Y, f(W)) |
| 7) | p(x, f(X) ,X) | p(Y, f(a),b) |
| 8) | q(f(a, a), V ,Z) | q(X, f(X,X), Y) |

3. Caesar

Consider the following situation:

*Marcus is a Pompeian.*

All Pompeiians are Romans.

*Caesar is a ruler.*

*All Romans are either loyal to Caesar or hate Caesar (but not both).*

*Everyone is loyal to someone.*

*People only try to assassinate rulers they are not loyal to.*

*Marcus tries to assassinate Caesar.*

Translate these sentences to First-Order Logic.

1. pompeiian(marcus)
2. For all x, pompeiian(X) -> roman(X)
3. ruler(caesar)
4. For all x, roman(X)->[loyal(X,caesar)^~hate(X,caesar)] V [~loyal(X,caesar)^hate(X,caesar)]
5. For all x, Some y, people(X), people(Y) -> loyal(X, Y)
6. For all x,y, people(X), ruler(Y), tryToAssassinate(X, Y) -> ~loyal(X, Y)
7. tryToAssassinate(marcus, caesar)

Prove that ***Marcus hates Caesar*** using Natural Deduction. Label all derived sentences with the ROI and prior sentences and unifier used.

Implicitly We Know

1. Some x, marcus ^ people(X)
2. Some x, caesar ^ people(X)

*Marcus is a roman because {Marcus is Pompeian and Pompeiians are Romans}*

1. roman(marcus) MP on 1 and 2

u={X/marcus} pompeiian(marcus) -> roman(marcus)

*Marcus is not loyal to Caesar because {People only try to assassinate rulers they are not loyal to and Marcus tries to assassinate Caesar}*

1. people(marcus) AE on 8

u={X/marcus}

1. ~loyal(marcus, caesar) MP on 11, 7, 3 on 6

u={X/marcus, Y/caesar}

people(marcus), ruler(marcus), tryToAssassinate(marcus, caesar)

*Marcus hates Caesar because {Marcus is a roman and Marcus is not loyal to Caesar}*

1. ~loyal(marcus, caesar) ^ hates(marcus, caesar) MP 10, 4

u={X/marcus}

1. hates(marcus, caesar) AE on 13

Convert all the sentences into CNF

1. pompeiian(marcus)
2. ~pompeiian(x) v roman(x)
3. ruler(caesar)

~roman(x) v { [loyal(x,caesar) ^ ~hate(x,caesar)] v [~loyal(x,caesar) ^ hate(x,caesar)] }

//Now distribute the ~roman(X) inside the big { } bracket!

[~roman(x) v loyal(x,caesar) ^ ~roman(x) v ~hate(x,caesar) ] v [~loyal(x,caesar) ^ hate(x,caesar)]

//Applying distributivity and removing the unnecessary ones

~roman(x) v { [~loyal(X, caesar) v loyal(X, caesar)] ^ [~loyal(X,caesar) v ~hate(X, caesar)] ^ [hate(x, caesar) v loyal(x, caesar)] ^ [hate(x, caesar) v ~ hate(x, caesar)] }

1. ~roman(x) v { hate(x, caesar) v loyal(x, caesar) }
2. ~people(X) v ~people(F(x)) v loyal(X, F(x))
3. ~people(X) ^ ~ruler(Y) ^ ~tryToAssassinate(X, Y) v ~loyal(X, Y)
4. tryToAssassinate(marcus, caesar)

Prove that ***Marcus hates Caesar*** using Resolution Refutation.

1. ~hates(marcus, caesar)

Distribution on 4

1. [~roman(marcus) v hate(marcus, caesar)] v [~roman(marcus) v loyal(marcus, caesar)]

Unification on 9 using 8 AND Resolution Refutation on 9

1. ~roman(marcus) v loyal(marcus, caesar)

Resolution Refutation from 10 onto 2

1. ~pompeiian(marcus)

//11 contradicts with statement 1 and therefore they cancel out by Resolution Refutation

1. {null-set}

4. Writing KBs in FOL

Write a KB (rules/axioms) in FOL for…

**Map-colouring** – every state must be exactly 1 colour, and adjacent states must be different colours. Assume possible colours are states are defined using unary predicate like colour(red) or state(WA). To say a state has a colour, use a binary predicate, e.g. ‘colour(WA, red)’.

state(WA)

state(NT)

state(SA)

state(QU)

state(NSW)

state(VI)

state(TA)

neighbor(WA,NT)

neighbor(WA,SA)

neighbor(NT,SA)

neighbor(NT,QU)

neighbor(QU,SA)

neighbor(QU,NSW)

neighbor(SA,NSW)

neighbor(SA,VI)

neighbor(NSW,VI)

For every X, state(X) -> colour(X, red) v colour(X, green) v colour(X, blue)

For every X, Y, P, Q state(X) ^ state(Y) ^ neighbor(X,Y) ^ color(X, P) ^ color(Y, Q) -> P != Q

**Sammy’s Sport Shop** – include implications of facts like obs(1,W) or label(2,B), as well as constraints about the boxes and colours. Use predicate ‘cont(x, q)’ to represent that box x contains tennis balls of colour q (where q could be W, Y, or B).

box(1)

box(2)

box(3)

color(white)

color(yellow)

color(both)

observe(1, yellow)

observe(2, white)

observe(3, yellow)

label(1, white)

label(3, both)

label(1, white)

//If you observe a ball of that color drawn, then it contains either that color alone or both

For every X,Y, box(X), color(Y), observe(Y) -> contains(X, Y) v contains(X, B)

//If the box is labeled that color, then it does not contain that color

For every Y, X, box(Y), color(X), label(Y, X) -> ~contains(Y, X)

//Each box is of different color

For every Y, contains (box(Y), color(white)) -> contains (box(Y), ~color(yellow)) ^ tains (box(Y), ~color(both))

For every Y, contains (box(Y), color(yellow)) -> contains (box(Y), ~color(white)) ^ tains (box(Y), ~color(both))

For every Y, contains (box(Y), color(both)) -> contains (box(Y), ~color(yellow)) ^ tains (box(Y), ~color(white))

//Each box contains one of the three colors

For every X, Y, box(Y) ^ color(X)

**Wumpus World** - (hint start by defining a helper concept ‘adjacent(x,y,p,q)’ which defines when a room at coordinates (x,y) is adjacent to another room at (p,q). Don’t forget rules for ‘stench’, ‘breezy’, and ‘safe’.

//Define the starting square

For all X,Y starting\_square(X,Y) -> X = 1 ^ Y = 1

//Declare the adjacent squares for any given coordinates X, Y in the Wumpus world!

For all X, Y, adjacent(X, Y, P, Q) <-> [P = X+1 ^ Q = Y] v [P = X-1 ^ Q = Y] v

[P = X ^ Q = Y+1] v [P = X ^ Q = Y-1]

//Rooms next to rooms with Wumpus have stench

For all X, Y, P, Q adjacent(X, Y, P, Q) ^ wumpus(P,Q) -> stench(X, Y)

//Rooms next to rooms with Breeze have Pit

For all X, Y, P, Q adjacent(X, Y, P, Q) ^ breeze(P,Q) -> pit(X, Y)

//Rooms next to rooms with Breeze have Pit

For all P, Q ~pit(P, Q) ^ ~Wumpus(P, Q) -> safe(P, Q)

**4-Queens** – assume row(1)…row(4) and col(1)…col(4) are facts; write rules that describe configurations of 4 queens such that none can attack each other, using ‘queen(r,c)’ to represent that there is a queen in row r and col c.

row(0)

row(1)

row(2)

row(3)

col(0)

col(1)

col(2)

col(3)

//No Queen in both the left diagonal

For all X, Y, queen(row(X), col(Y)), X >= 0, X <= 3, Y >= 0, Y <= 3 -> ~queen(row(X+1), col(Y+1)) ^ ~queen(row(X-1), col(Y-1)) ^ ~queen(row(X+2), col(Y+2)) ^ ~queen(row(X-2), col(Y-2)) ^ ~queen(row(X+3), col(Y+3)) ^ ~queen(row(X-3), col(Y-3))

//No Queen in both the right diagonal

For all X, Y, queen(row(X), col(Y)), X >= 0, X <= 3, Y >= 0, Y <= 3 -> ~queen(row(X+1), col(Y-1)) ^ ~queen(row(X-1), col(Y+1)) ^ ~queen(row(X+2), col(Y-2)) ^ ~queen(row(X-2), col(Y+2)) ^ ~queen(row(X+3), col(Y-3)) ^ ~queen(row(X-3), col(Y+3))

//No Queen in the top or bottom col

For all X, Y, queen(row(X), col(Y)), X >= 0, X <= 3, Y >= 0, Y <= 3 -> ~queen(row(X+1), col(Y)) ^ ~queen(row(X-1), col(Y)) ^ ~queen(row(X+2), col(Y)) ^ ~queen(row(X-2), col(Y)) ^ ~queen(row(X+3), col(Y)) ^ ~queen(row(X-3), col(Y))

//No Queen in the left or right row

For all X, Y, queen(row(X), col(Y)), X >= 0, X <= 3, Y >= 0, Y <= 3 -> ~queen(row(X), col(Y+1)) ^ ~queen(row(X), col(Y-1)) ^ ~queen(row(X), col(Y+2)) ^ ~queen(row(X), col(Y-2)) ^ ~queen(row(X), col(Y+3)) ^ ~queen(row(X), col(Y-3))