CSCE 420 – Fall 2025

Homework 4 (HW4)

due: Tues, Nov 11, 2025, 11:59 pm

Turn-in answers as a PDF document (HW4.pdf) and commit/push it to your class github repo (in your HW4/ directory).

All homeworks must be typed, *not* hand-written and scanned as a photo.

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.

- **bowling balls are sporting equipment**

$\forall x[\ bowlingBall(x) \rightarrow sportEquip(x)\ ]$

bowlingBall(x) - x is a bowling ball
sportEquip(x) - x is sporting equipment

- **horses are faster than frogs (there are many ways to say this in FOL; try expressing it this way: "all horses have a higher speed than any frog")**

horse(x) - x is a horse

frog(y) - y is a frog

$\forall x \forall y[\ horse(x) \wedge frog(y) \rightarrow speed(x) > speed(y)\ ]$

- **all domesticated horses have an owner**

horse(x) - x is a horse

domesticated(x) - x is domesticated

person(y) - y is a person (optional, for clarity)

owns(y,x) - y owns x

$\forall x[\ horses(x) \wedge domesticated(x) \rightarrow \exists y\ (person(y) \wedge owns(y,x))\ ]$

- **the rider of a horse can be different than the owner**

horse(h) - h is a horse

rider(h) - function returning the rider of h

owner(h) - function returning the owner of h

$\exists h\ [\ rider(h) \wedge horse(h) \neq owner(h)\ ]$

- **a finger is any digit on a hand other than the thumb**

finger(x) - x is a finger

digit(x) - x is a digit

on(x,h) - x is on h

hand(h) - h is a hand

thumb(x) - x is a thumb

$\forall x[finger(x) \leftrightarrow (digit(x) \wedge \exists h\ [hand(h) \wedge on(x,h)] \wedge \neg thumb(x))]$

- **an isosceles triangle is defined as a polygon with 3 edges connected at 3 vertices, where 2 (but not 3) edges have the same length**

isoTri(x) - x is an isosceles triangle

polygon(x) - x is a polygon

edge(e,x) - e is an edge of polygon x

vertex(v,x) - v is a vertex of polygon x

len(e) - the length of edge e

connectedAt(e_1,e_2,v) - edges e_1 and e_2 meet at vertex v

$\forall x$ [isoTri(x) $\leftrightarrow$ (polygon(x) $\wedge$ $\exists e_1,e_2,e_3,v_1,v_2,v_3$ (edge(e_1, x) $\wedge$ edge(e_2, x) $\wedge$ edge(e_3, x) $\wedge$ vertex(v_1, x) $\wedge$ vertex(v_2, x) $\wedge$ vertex(v_3, x) $\wedge$ connectedAt(e_1, e_2, v_1) $\wedge$connectedAt(e_2, e_3, v_2) $\wedge$ connectedAt(e_3, e_1, v_3) $\wedge$ len(e_1) = len(e_2) $\wedge$ len(e_1) $\neq$ len(e_3) $\wedge$ len(e_2) $\neq$ len(e_3)))))]

2. Convert the following first-order logic sentence into CNF:

" x person(x) ^ [$z petOf(x,z) ^ ["y petOf(x,y)®dog(y)]] ® doglover(x)

$\forall$ x person(x) ^ [$\exists$z petOf(x,z) ^ [$\forall$y petOf(x,y)$\rightarrow$dog(y)]] $\rightarrow$ doglover(x)

Keep in mind that the quantifiers have *lower* precedence than all the other operators in FOL sentences.

$\forall$ x person(x) ^ [$\exists$ z petOf(x,z) ^ [$\forall$ y petOf(x,y) -> dog(y)]] -> doglover(x)
Law of implication
¬ (person(x) ^ [$\exists$ z petOf(x,z) ^ [$\forall$ y petOf(x,y) -> dog(y)]] v doglover(x))
(¬ person(x) v [¬ $\exists$ z petOf(x,z) v ¬ $\forall$ y petOf(x,y) -> dog(y))] v doglover(x)
Updating qualifiers negation
(¬ person(x) v [$\forall$ z ¬petOf(x,z) v $\exists$ y ¬petOf(x,y) -> dog(y))] v doglover(x)
petOf(x,z) -> dog(y) = ¬petOf(x,z) v dog(y)
¬(petOf(x,z) -> dog(y)) = ¬(petOf(x,z) v dog(y)) = petOf(x,z) v ¬dog(y)
Sub back in
(¬ person(x) v [$\forall$ z ¬petOf(x,z) v $\exists$ y(petOf(x,z) v ¬dog(y))] ^ doglover(x)
$\forall$ x[$\forall$ z $\exists$ y (¬ person(x) v ¬petOf(x,z) v (petOf(x,z) ^ ¬dog(y)) v doglover(x))]
$\forall$ x $\forall$ z (¬ person(x) v ¬petOf(x,z) v (petOf(x,z) ^ ¬dog(x,z)) v doglover(x))
Distribute v and ^
A = (¬ person(x) v ¬petOf(x,z) v doglover(x))
(B ^ C) = (petOf(x,f(x,z)) ^ ¬dog(f(x,z)))
(¬ person(x) v ¬petOf(x,z) v (petOf(x,f(x,z)) v doglover(x)) ^ (¬ person(x) v ¬ petOf(x,z) v ¬dog(f(x,z)) v doglover(x))
Hence final ans,
$\forall$ x $\forall$ z (¬ person(x) v ¬petOf(x,z) v (petOf(x,f(x,z)) v doglover(x)) ^ (¬ person(x) v ¬ petOf(x,z) v ¬dog(f(x,z)) v doglover(x))

3. 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. Capital letters represent variables; 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.

- **owes(owner(X),citibank,cost(X)) owes(owner(ferrari),Z,cost(Y))**

Comparing arguments:

owner(X) = owner(ferrari) => {X/ferrari}

citibank = Z => {Z/citibank}

cost(X) = cost(y) => {Y/ferrari}

Hence, u = { X/ferrari, Y/ferrari, Z/citibank }

Applying the substitution

1. owes(owner(X),citibank,cost(X)) = owes(owner(ferrari),citibank,cost(ferrari))
2. owes(owner(ferrari),Z,cost(Y)) = owes(owner(ferrari),citibank,cost(ferrari))

Hence yes unifiable

- **gives(bill, jerry, book21) gives(X,brother(X),Z)**

Comparing arguments:

bill = x => {X/bill}

jerry = brother(X) => substitute previous binding X/bill → becomes brother(bill);

compare jerry vs brother(bill) → error different constant, hence, cannot unify

book21 = Z => {Z/book21}

Hence, fails cuz jerry is a constant, brother(bill) is a function term. These can never match. Once X is bound to bill there is a mismatch argument.

- **opened(X,result(open(X),s0))) opened(toolbox,Z)**

Comparing arguments:

Toolbox = X -> u1 = {X/toolbox}

result(open(X),s0)) = Z -> u2 = { X/toolbox, Z/result(open(toolbox), s0)}

Substituting

opened(X,result(open(X),s0))) = opened(toolbox, result(open(toolbox), s0))

opened(toolbox,Z) = opened(toolbox, result(open(toolbox), s0))

Thus unifiable

4. Consider the following situation:

Marcus is a Pompeian.
All Pompeians are Romans.
Ceasar 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.

a) Translate these sentences to First-Order Logic.

Pompeian(x) - x is a Pompeian

Roman(x) - x is a Roman

Ruler(x) - x is a ruler

LoyalTo(x,y) - x is loyal to y

Hates(x,y) - x hates y

TryAssassinate(x,y) - x tries to assassinate y

Marcus is a Pompeian.
Pompeian(marcus)
All Pompeians are Romans.
$\forall$ x(Pompeian(x) -> Roman(x))
Ceasar is a ruler.

Ruler(Ceasar)

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

∀ x[Roman(x) -> ((LoyalTo(x, Caesar) v Hates(x, Caesar)) ^ ¬ (LoyalTo(x, Caesar) ^ Hates(x, caesar)))]

Everyone is loyal to someone.

∀ x ∃ y LoyalTo(x,y)

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

∀ x ∀ y [TryAssassinate(x,y) -> Ruler(y) ^ ¬LoyalTo(x,y))]

Marcus tries to assassinate Caesar.

TryAssassinate(Marcus, Caesar)

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

Goal is Hates(Marcus/Caesar)

Pompeian(Marcus) -> Roman(Marcus)	UI (Universal Instantiation)	{x/Marcus}
Roman(Marcus)	MP (Modus Ponens)	
TryAssassinate(Marcus, Caesar) -> (Ruler(Caesar) ^ ¬ LoyalTo(Marcus, Caesar))	UI	{x/Marcus, y/Caesar}
Ruler(Caesar) ^ ¬ LoyalTo(Marcus, Caesar)	MP	
¬ LoyalTo(Marcus, Caesar)	⋀-Elim	
Roman(Marcus) -> ((LoyalTo(Marcus, Caesar) v Hates(Marcus, Caesar)) ^ ¬(LoyalTo(Marcus, Caesar) ^ Hates(Marcus, Caesar)))	UI	{x/Marcus}
(LoyalTo(Marcus, Caesar) v Hates(Marcus, Caesar)) ^ ¬(LoyalTo(Marcus, Caesar) ^ Hates(Marcus, Caesar))	MP	
LoyalTo(Marcus, Caesar) v Hates(Marcus, Caesar)	Elim	
¬(LoyalTo(Marcus, Caesar) ^ Hates(Marcus, Caesar))	Elim	

LoyalTo(Marcus, Caesar) is false hence, must have Hates(Marcus, Caesar).		

c) Convert all the sentences into CNF

Marcus is a Pompeian.
Pompeian(marcus)
All Pompeians are Romans.
∀ x(Pompeian(x) -> Roman(x))
¬Pompeian(x) ^ Roman(x)
Ceasar is a ruler.
Ruler(Ceasar)
All Romans are either loyal to Caesar or hate Caesar (but not both).
∀ x[Roman(x) -> ((LoyalTo(x, Caesar) v Hates(x, Caesar)) ^ ¬ (LoyalTo(x, Caesar) ^ Hates(x, caesar)))]
∀ x[Roman(x) v ((LoyalTo(x, Caesar) v Hates(x, Caesar)) ^ (LoyalTo(x, Caesar) ^ Hates(x, caesar)))]
eliminate
Distribute v over ^
∀ x[(Roman(x) v (LoyalTo(x, Caesar) v Hates(x, Caesar)) ^ (Roman(x) v LoyalTo(x, Caesar) ^ Hates(x, caesar)))]
De morgans law to second part
LoyalTo(x, Caesar) ^ Hates(x, Caesar))(LoyalTo(x, Caesar) v Hates(x, Caesar)
Sub back in
Roman(x) v LoyalTo(x, Caesar) v Hates(x, Caesar)
Everyone is loyal to someone.
∀ x ∃ y LoyalTo(x,y)
People only try to assassinate rulers they are not loyal to.
∀ x ∀ y [TryAssassinate(x,y) -> Ruler(y) ^ ¬LoyalTo(x,y))]
Elimination
∀ x ∀ y [TryAssassinate(x,y)(Ruler(y)LoyalTo(x,y))]
Distribute ^ and v
∀ x ∀ y [(TryAssassinate(x,y)(Ruler(y))(TryAssassinate(x,y)LoyalTo(x,y))]
Remove qualifiers
(TryAssassinate(x,y)(Ruler(y))(TryAssassinate(x,y)LoyalTo(x,y))
Marcus tries to assassinate Caesar.
TryAssassinate(Marcus, Caesar)

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

Goal: Hates(Marcus,Caesar)

Negated goal: ¬Hates(Marcus,Caesar)

Clauses	Derived Clause
Pompeian(Marcus) and ¬Pompeian(x) ∨ Roman(x) {x/Marcus}	Roman(Marcus)
¬TryAssassinate(x,y) ∨ ¬LoyalTo(x,y) and TryAssassinate(Marcus,Caesar) {x/Marcus, y/Caesar}	¬LoyalTo(Marcus,Caesar)
Ruler(Caesar)	
¬Roman(x) ∨ LoyalTo(x,Caesar) ∨ Hates(x,Caesar) {x/Marcus} Roman(Marcus)	LoyalTo(Marcus,Caesar) ∨ Hates(Marcus,Caesar)
LoyalTo(Marcus,Caesar) ∨ Hates(Marcus,Caesar) and ¬LoyalTo(Marcus,Caesar)	Hates(Marcus,Caesar)
Hates(Marcus,Caesar) is opposite of ¬Hates(Marcus,Caesar)	

Hence, since ¬Hates(Marcus,Caesar) leads to a contradiction the knowledge base has Hates(Marcus,Caesar).

5. Write a KB in First-Order Logic with rules/axioms for…

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

Predicate	Meaning
state(s)	s is a state
color(c)	c is a color
adjacent(s_1, s_2)	states s_1 and s_2 are adjacent
hasColor(s, c)	state s has color c

KB

Each state has at least 1 color

$\forall s\,[\,state(s) \rightarrow \exists c\,(color(c) \wedge hasColor(s, c))\,]$

Each state has at most 1 color

$\forall s \forall c1 \forall c2\,[(state(s) \wedge color(c1) \wedge color(c2) \wedge hasColor(s, c1) \wedge hasColor(s, c2)) \rightarrow (c1 = c2)]$

Adjacent states must have different colors

$\forall s1 \forall s2 \forall c\,[(state(s1) \wedge state(s2) \wedge adjacent(s1, s2) \wedge hasColor(s1, c)) \rightarrow \neg hasColor(s2, c)]$

b. **Sammy's Sport Shop** – include implications of facts like obs(1,W) or label(2,B), as well as constraints about the boxes and colors. Use predicate '<u>cont(x,q)</u>' to represent that box x contains tennis balls of color q (where q could be W, Y, or B).

Predicate	Meaning
obs(x, q)	observation that box x has a label or sign q
label(x, q)	box x is labeled with color q
cont(x, q)	box x contains tennis balls of color q

(q can be W = white, Y = yellow, or B = blue)

1. If an observation says a box has a certain label, then the box actually has that label.

$\forall x \forall q$ [obs(x, q) $\rightarrow$ label(x, q)]

2. Each box contains exactly one color of tennis balls.

$\forall x \exists q$ [cont(x, q) $\wedge$ color(q)]

$\forall x \forall q1 \forall q2$ [(cont(x, q1) $\wedge$ cont(x, q2)) $\rightarrow$ (q1 = q2)]

3. A box label may be incorrect

$\forall x \forall q$ [label(x, q) $\rightarrow \neg$cont(x, q)]

4. If a box is labeled incorrectly, then its actual contents must be one of the *other* colors.
$\forall x \forall q \forall r$ [(label(x, q) $\wedge$ color(r) $\wedge$ r $\neq$ q) $\rightarrow$ cont(x, r)]

5. There are only three possible colors for tennis balls.

color(W)

color(Y)

color(B)

c. **Wumpus World** - (hint start by defining a helper concept '<u>adjacent(x,y,p,q)</u>' 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'.

Predicate	Meaning
room(x, y)	there is a room at coordinates (x, y)
adjacent(x, y, p, q)	room (x, y) is adjacent to room (p, q)
stench(x, y)	there is a stench in room (x, y)
breeze(x, y)	there is a breeze in room (x, y)

wumpus(x, y)	the Wumpus is in room (x, y)
pit(x, y)	there is a pit in room (x, y)
safe(x, y)	room (x, y) is safe

1. rooms are adjacent if they share a side

$\forall x \forall y \forall p \forall q$ [adjacent(x, y, p, q) $\leftrightarrow$((x = p $\wedge$ (y = q+1 $\vee$ y = q−1)) $\vee$ (y = q $\wedge$ (x = p+1 $\vee$ x = p−1)))]

2. a stench exists in a room if the Wumpus is in an adjacent room

$\forall x \forall y$ [stench(x, y) $\leftrightarrow$ $\exists p \exists q$ (adjacent(x, y, p, q) $\wedge$ wumpus(p, q))]

3. a breeze exists in a room if a pit is in an adjacent room

$\forall x \forall y$ [breeze(x, y) $\leftrightarrow$ $\exists p \exists q$ (adjacent(x, y, p, q) $\wedge$ pit(p, q))]

4. a room is safe if it has no Wumpus and no pit

$\forall x \forall y$ [safe(x, y) $\leftrightarrow$ ¬wumpus(x, y) $\wedge$ ¬pit(x, y)]

5. If there is no stench and no breeze, then all adjacent rooms are safe

$\forall x \forall y$ [(¬stench(x, y) $\wedge$ ¬breeze(x, y)) $\rightarrow$ $\forall p \forall q$ (adjacent(x, y, p, q) $\rightarrow$ safe(p, q))]

d. **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 '<u>queen(r,c)</u>' to represent that there is a queen in row r and col c.

Don't forget to quantify all your variables.

Predicate	Meaning

row(r)	r is a row (1…4)
col(c)	c is a column (1…4)
queen(r, c)	There is a queen in row r and column c
attack(r1, c1, r2, c2)	The queen at (r1, c1) can attack the queen at (r2, c2)
safe(r1, c1, r2, c2)	The two queens are safe from each other (cannot attack each other)

Facts

row(1) row(2) row(3) row(4)

col(1) col(2) col(3) col(4)

1. Attack condition – two queens attack each other if they share a row, a column, or a diagonal.

$\forall r1 \forall c1 \forall r2 \forall c2$ [attack(r1, c1, r2, c2) $\leftrightarrow$ ((r1 = r2) $\vee$ (c1 = c2) $\vee$ (|r1 − r2| = |c1 − c2|))]

2. Safety condition – two queens are safe if they do not attack each other.

$\forall r1 \forall c1 \forall r2 \forall c2$ [safe(r1, c1, r2, c2) $\leftrightarrow \neg$attack(r1, c1, r2, c2)]

3. Constraint – no two queens can attack each other.

$\forall r1 \forall c1 \forall r2 \forall c2$ [(queen(r1, c1) $\wedge$ queen(r2, c2) $\wedge$ (r1 $\neq$ r2 $\vee$ c1 $\neq$ c2)) $\rightarrow \neg$attack(r1, c1, r2, c2)]

4. Constraint – exactly one queen per row.

$\forall r$ [$\exists c$ queen(r, c) $\wedge$ $\forall c1 \forall c2$ ((queen(r, c1) $\wedge$ queen(r, c2)) $\rightarrow$ (c1 = c2))]

5. Constraint – exactly one queen per column.

$\forall c$ [$\exists r$ queen(r, c) $\wedge$ $\forall r1 \forall r2$ ((queen(r1, c) $\wedge$ queen(r2, c)) $\rightarrow$ (r1 = r2))]