**Name:** **Entry No**

**COL 765: Introduction to Logic and Functional Programming**
**"Big Quiz" (40 minutes, 40 marks), 11.11.2024**

Name: _______ANSWERS________________________________ Entry No. ____XXX___

**Q1. First-Order Logic for Specifying Relational Properties. [2+2+2+2=8]**
Suppose we also have a *binary* predicate symbol $R^{(2)} \in \Pi$. Write First-order Logic (FOL) sentences using individual variables $x, y, z, w$, propositional connectives $\neg, \wedge, \vee, \rightarrow$, quantifiers $\forall, \exists$, and $=$ the (binary) equality relational symbol with its standard meaning, to describe the following properties:

(a) $R$ is *not transitive*: $\neg((\forall x)(\forall y)(\forall z)[(R(x,y) \wedge R(y,z)) \rightarrow R(x,z)])$
$\Leftrightarrow$ $(\exists x)(\exists y)(\exists z)[R(x,y) \wedge R(y,z) \wedge \neg R(x,z)]$ by De Morgan's Laws.

(b) $R$ is *not anti-symmetric*: $\neg((\forall x)(\forall y)[(R(x,y) \wedge R(y,x)) \rightarrow (x = y)])$
$\Leftrightarrow$ $(\exists x)(\exists y)[R(x,y) \wedge R(y,x) \wedge \neg(x = y)]$ by De Morgan's Laws.

(c) Now suppose we also have a *unary* predicate symbol $P^{(1)} \in \Pi$. We say $P$ is a logical property closed under equality if whenever an element has property $P$, then all elements equal to it also have that property. Write a FOL *sentence* in $\mathscr{L}(\mathscr{X}, \Sigma, \Pi)$ to express closure under equality of property $P$:
$(\forall x)(\forall y)[(x = y) \rightarrow (P(x) \rightarrow P(y))]$ or equivalently
$(\forall x)(\forall y)[((x = y) \wedge P(x)) \rightarrow P(y)]$

(d) Now suppose we also have a *binary* predicate symbol $\sqsubseteq^{(2)} \in \Pi$, which satisfies reflexivity, antisymmetry and transitivity properties. Write a FOL *formula* in $\mathscr{L}(\mathscr{X}, \Sigma, \Pi)$ to express that $z$ is a *least upper bound* of $x$ and $y$ with respect to partial ordering $\sqsubseteq$: $(x \sqsubseteq z) \wedge (y \sqsubseteq z) \wedge (\forall w)[((x \sqsubseteq w) \wedge (y \sqsubseteq w)) \rightarrow (z \sqsubseteq w)]$

($z$ is an upper bound of both $x$ and $y$, and *for any* upper bound $w$ of both $x$ and $y$, $z$ is less than or equal to $w$).

**Q2. First-Order Logic for Specifying Properties of Models. [2+2+1+2+3+2=12]**
First-order Logic (FOL) sentences can be used to characterise sets (which satisfy the sentence). Write *FOL sentences* using only individual variables $x, y, z$, propositional connectives $\neg, \wedge, \vee, \rightarrow$, quantifiers $\forall, \exists$, and $=$ as the *only* (binary) relational symbol (with its standard meaning) to describe the following properties of sets:

(a) There is *at most one* element in the set: [Hint: all elements, if they exist, are **equal**]
$\phi_{|\leq 1|} \equiv (\forall x)(\forall y)[x = y]$
[This is the negation of saying there are at least two different elements]

(b) There is *at least one* element in the set: [Hint: there exists an element such that **true**, i.e., **it's equal to itself**]
$\phi_{|\geq 1|} \equiv (\exists x)[x = x]$

(c) There is *exactly one* element in the set: $\phi_{|\leq 1|} \wedge \phi_{|\geq 1|}$

**Name:** **Entry No**

(d) There are *at least two* elements in the set:

$$\phi_{|\geq 2|} \equiv (\exists x)(\exists y)[\neg(x = y)] \quad \text{Note this is equivalent to}$$
$$\neg\phi_{|\leq 1|} \equiv \neg((\forall x)(\forall y)[x = y])$$

(e) There are *at most two* elements in the set:

$$\phi_{|\leq 2|} \equiv (\forall x)(\forall y)(\forall z)[(x = y) \vee (x = z) \vee (y = z)]$$

[This is the negation of saying there are at least three different elements]

(f) There are *exactly two* elements in the set:

$$\phi_{|\leq 2|} \wedge \phi_{|\geq 2|}$$

### Q3 Encodings in the Untyped $\lambda$-calculus [2+2+4=8]

Recall the encoding of the booleans in the Pure Untyped $\lambda$-calculus — constants **T** and **F** as $T \equiv \lambda x\,.\,\lambda y\,.\,x$, $F \equiv \lambda x\,.\,\lambda y\,.\,y$ and `if-then-else` as $D \equiv \lambda t\,.\,\lambda x\,.\,\lambda y\,.\,((t\ \ a)\ \ b)$

If you coded these in OCaml, giving type variables to the arguments, what are types of these terms? (Show your working by decorating the arguments of the $\lambda$-expressions.)

**(a)** Type of $T \equiv \lambda x^{\alpha}\,.\,\lambda y^{\beta}\,.\,x$ **:** $\alpha \to (\beta \to \alpha)$

**(b)** Type of $F \equiv \lambda x^{\alpha}\,.\,\lambda y^{\beta}\,.\,y$ **:** $\alpha \to (\beta \to \beta)$

**(c)** Type of $D \equiv \lambda t^{\gamma}\,.\,\lambda x^{\alpha}\,.\,\lambda y^{\beta}\,.\,((t\ \ a)^{\theta}\ \ b)^{\delta}$ **:** $(\alpha \to (\beta \to \delta)) \to (\alpha \to (\beta \to \delta))$

$\gamma = \alpha \to \theta$, for some type $\theta$ — since $t$, of type $\gamma$, is applied to $a$ of type $\alpha$
$\theta = \beta \to \delta$, for some type $\delta$ — since $(t\ \ a)$, of type $\theta$, is applied to $b$, which is of type $\beta$
Therefore, $\gamma = \alpha \to (\beta \to \delta)$

### Q4 Specifying a Type-Checker in Prolog [12]

Consider the following typing rules for a simply-typed $\lambda$-calculus:

$$(Var)\ \frac{}{\Gamma \vdash x : \tau} \quad \text{provided } (x : \tau) \in \Gamma$$

$$(App)\ \frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2} \qquad (Pair)\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$(Lam)\ \frac{\Gamma[x : \tau_1] \vdash e : \tau_2}{\Gamma \vdash \lambda x\,.\,e : \tau_1 \to \tau_2} \qquad (Proj_i)\ \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathrm{proj}_i\ e : \tau_i} \quad (i = 1 \text{ or } 2)$$

Suppose we encode type assumptions $\Gamma$ in Prolog as a *stack* (i.e., a list) of pairs of the form `(X,T)` where Prolog variable `X` represents a $\lambda$-calculus variable $x$, and Prolog variable `T`

**Name:** **Entry No**

represents a type $\tau$. We use a Prolog predicate `member(A,S)` that succeeds if `A` appears in list `S`, and fails otherwise, for looking for a type assumption on the stack.

The types in our system are represented using Prolog constructors: a Prolog name `A` is used to represent a type variable $\alpha$, and **`arrow`**`(T1, T2)` represents the function type $\tau_1 \rightarrow \tau_2$, where `T1` and `T2` are Prolog variables representing types $\tau_1$ and $\tau_2$; cartesian product of types $\tau_1 \times \tau_2$ is represented as **`cartesian`**`(T1, T2)`. $\lambda$-expressions are coded using Prolog constructors **`var`**`(X)`, **`app`**`(E1,E2)`, **`lam`**`(X,E)`, **`pair`**`(E1,E2)`, **`proj1`**`(E)` and **`proj2`**`(E)`.

The notation $\Gamma[x : \tau]$ denotes the type assumptions $\Gamma$ augmented with the (most recent) type assumption $x : \tau$. The objective is to define the type-checker as a Prolog predicate `hastype(G,E,T)` encoding the typing judgment $\Gamma \vdash e : \tau$. We show you how the rules *(Var)* and *(Lam)* respectively are encoded:

```
% Var
hastype(G,var(X),T) :- member((X,T),G), !.
% Lam
hastype(G,lam(X,E),arrow(T1,T2) ) :-  hastype([(X,T1)|G],E,T2).
```

Now complete the encoding for the other typing rules given above:

```
% App
hastype(G,app(E1,E2),T2 ) :-
                hastype(G,E1,arrow(T1,T2)), hastype(G,E2,T1).


% Pair
hastype(G,pair(E1,E2),cartesian(T1,T2)) :-
                hastype(G,E1,T1), hastype(G,E2,T2).

% Proj1
hastype(G,proj1(E),T1) :- hastype(G,E,cartesian(T1,T2)).


% Proj2
hastype(G,proj2(E),T2) :- hastype(G,E,cartesian(T1,T2)).
```