TD 1 : Introduction à la Logique du Premier Ordre et Datalog

Exercice 1. (Modélisation)
Préciser à l'aide d'un quantificateur le sens de "un" dans les phrases suivantes et formaliser les en logique des prédicats :

1. Jean suit un cours.
   Solution: on introduit deux prédicats binaires suit et cours, et une constante Jean.
   $\exists c, cours(c) \wedge suit(Jean, c)$
2. Un logicien a été champion du monde de cyclisme.
   Solution: on introduit trois symboles de relation d'arité 1 compétition, champion, logicien.
   $\exists l, comptition(l) \wedge logicien(l) \wedge champion(l)$
3. Un entier naturel est pair ou impair.
   Solution: ici 3 symboles de relation d'arité 1 : naturel, pair, impair.
   $\forall n(naturel(n) \rightarrow (pair(n) \vee impair(n)))$
4. Un enseignant-chercheur a toujours un nouveau sujet à étudier.
   Solution: deux symboles de relation d'aritée 1 enseignantc, nouveau et un symbole de relation d'arité 2 etudie.
   $\forall x(enseignantc(x) \rightarrow (\exists y(nouveau(y) \wedge etudie(x, y))))$
5. Dans un triangle isocèle il existe une médiane qui est également hauteur.
   Solution: on introduit deux symboles de relation unaire triangle, isocele et deux symboles de relation binaire mediane (m est une médiane de t) et hauteur.
   $\forall t((triangle(t) \wedge isocele(t)) \rightarrow (\exists m(mediane(m, t) \wedge hauteur(m, t))))$
6. Dans un triangle équilatéral toute médiane est également hauteur.
   Solution: on introduit deux symboles de relation unaire triangle, equilateral et deux symboles de relation binaire mediane (m est une médiane de t) et hauteur.
   $\forall t((triangle(t) \wedge equilateral(t)) \rightarrow (\forall m(mediane(m, t) \rightarrow hauteur(m, t))))$

Exercice 2. (Modélisation 2)
Formaliser les phrases suivantes en logique des prédicats :
Hint: en analysant les phrases, on dégage différentes notions et propriétés : être un cheval, être un lapin, être un lévrier, être plus rapide que et deux indicidus Harry et Ralph. On va modéliser cela à l'aide d'un langage contenant deux constantes (symbole de fonction d'arité 0) H (pour Harry) et Ralph (pour Ralph), 4 symboles de relation d'aritée 1 chien, cheval, lapin et levrier et un symbole de relation binaire plus (pour plus rapide que).

1. Un cheval est plus rapide qu'un chien.
   Solution: $\forall x \forall y(cheval(x) \wedge chien(y) \rightarrow plus(x, y))$
2. Il existe un lévrier qui est plus rapide que tout lapin.
   Solution: $\exists l_0(levrier(l_0) \wedge (\forall y(lapin(y) \rightarrow plus(l_0, y))))$
3. Tout lévrier est un chien.
   Solution: $\forall x(levrier(x) \rightarrow chien(x))$
4. Harry est un cheval.
   Solution: $cheval(H)$
5. Ralph est un lapin.
   Solution: $lapin(R)$
6. La relation être plus rapide que est transitive.
   Solution: $\forall x \forall y \forall z(plus(x, y) \wedge plus(y, z) \rightarrow plus(x, z)$

$entier(x)$	"est un entier naturel"
$successeur(x, y)$	"est successeur de y"
$inf(x, y)$	"est inférieur ou égal à y"

Exercice 3. (Modélisation 3)

Représenter la phrase "Tout nombre entier naturel x a un successeur qui est inférieur ou égal à tout entier strictement supérieur à x" par une formule logique en utilisant les prédicats suivants :

Solution: $\forall n(entier(n) \rightarrow (\exists m(entier(m) \wedge successeur(m, n) \wedge (\forall r((entier(r) \wedge \neg(inf(r, n))) \rightarrow inf(m, r))))))$

Exercice 4. (SQL et Datalog)

Considérez la relation Flight :

`Flights`(fino : integer, from : string, to : string, distance : integer, departs : time, arrives : time)

Question : Donnez la requête en Datalog et SQL pour les phrases suivantes :

1. Trouvez le *fino* de tous les vols au départ de Madison.

(DL)
```
Ans(fino):- Flights(fino, "Madison", _, _, _, _)
```

(SQL)
```
SELECT fino
FROM Flights
WHERE Flights.from='Madison';
```

2. Trouvez le *fino* de tous les vols qui quittent Chicago après que le vol 101 arrive à Chicago au plus tard une heure après.

(DL)
```
Ans(fino):- Flights(101, _, "Chicago", _ , _, carrives), Flights(fino, "Chicago", _, depart, _
  ), depart > carrives, carrives + 1hr < depart
```

(SQL)
```
SELECT F2.fino
  FROM Flights F1, Flights F2
WHERE F1.to='Chicago' AND
    F1.fino=101 AND
    F2.from='Chicago' AND
    F2.depart > F1.carrives AND
    F2.depart < F1.carrives + 1;
```

3. Trouvez le *fino* de tous les vols qui ne partent pas de Madison.

(DL)
```
Ans(fino):-Flights(fino, _, _, _, _, _), not Flights(fino, "Madison", _, _,_, _)
```

(SQL)
```
SELECT fino
FROM Flights
WHERE Flights.from <> 'Madison';
```

4. Trouver toutes les villes accessibles de Madison à travers une série d'un ou plusieurs vols de correspondance.

(DL)
```
MadCity(to):-Flights(_,"Madison",to,_, _, _)
MadCity(to):-MadCity(from), Flights(_, from, to, _, _, _)
```

(SQL)
```
WITH RECURSIVE MadCity (to) AS
(SELECT F1.to -- Initial Subquery
 from Flights F1
 WHERE F1.from='Madison'
  UNION ALL
 SELECT F2.to FROM MadCity, Flights F2 -- Recursive Subquery
 WHERE F2.from=Madcity.to)
 SELECT * FROM MadCity;
```

5. Trouver toutes les villes accessibles à partir de Madison à travers une chaîne d'un ou plusieurs vols de correspondance, avec pas plus d'une heure passée sur n'importe quelle correspondance. (C'est-à-dire, chaque vol de correspondance doit partir au plus tard une heure après l'arrivée du vol précédent dans la chaîne.)

(DL)
```
MadCity(to,arrive):-Flights(_,"Madison",to,_, _, arrive)
MadCity(to, arrives):-MadCity(from,prev_arrives), Flights(_, from, to, _, departs,arrives),
  prev_arrives < departs, prev_arrives + 1hr < departs
```

(SQL)
```
WITH RECURSIVE MadCity (to, arrive) AS
(SELECT F1.to, F1.arrive -- Initial Subquery
from Flights F1
WHERE F1.from='Madison'
UNION ALL
SELECT F2.to, F2.arrives FROM MadCity, Flights F2 -- Recursive Subquery
WHERE F2.from=Madcity.to AND
      MadCity.arrive < F2.departs AND
      F2.departs < MadCity.arrive +1)
SELECT * FROM MadCity;
```

6. Trouver le *fino* de tous les vols qui ne partent pas de Madison ou une ville qui est accessible de Madison à travers une chaîne de vols.

(DL)
```
MadCity(to):-Flights(_,"Madison",to,_, _, _)
MadCity(to):-MadCity(from), Flights(_, from, to, _,_, _)
Ans(fino):-Flights(fino,from,to, _, _, _), not MadCity(from), not Flights(fino,'Madison',to,_,
  _,_)
```

(SQL)
```
WITH RECURSIVE MadCity (to) AS
(SELECT F1.to -- Initial Subquery
from Flights F1
WHERE F1.from='Madison'
UNION ALL
SELECT F2.to FROM MadCity, Flights F2 -- Recursive Subquery
WHERE F2.from=Madcity.to)
SELECT * FROM MadCity;

SELECT F1.fino
from Flights F1, MadCity MC
WHERE F1.from <> MC.from AND
      (F1.fino, F1.to) NOT IN (SELECT F2.fino, F2.to
                               FROM Flights F2
                               WHERE F2.from='Madison');
```