1 First-Order Logic

Part 1: Represent the following sentences in first-order logic. You will have to define a vocabulary (which should be consistent between sentences).

2. Every student who takes French passes it. 
3. Only one student took Greek in spring 2001.
4. The best score in Greek is always higher than the best score in French.
5. There is a male barber who shaves all the men who do not shave themselves.

Part 2: Write down a first-order logic sentence such that every world in which it is true contains exactly one object.

1.1 Tutor’s Guide

Part 1: An example vocabulary might be:

- *French* - a constant denoting the subject French 
- *Greek* - a constant denoting the subject Greek 
- *student/1* - a unary relation, *student(x)* iff constant *x* is a student (unnecessary if the *took* relation is defined to only apply to students)
- *took/3* - a ternary relation, *took(x, y, t)* iff *x* took the subject *y* during time interval *t* (there are alternatives to having time as an argument to the *took* relation. You could represent this as a course event, e.g. *course(e) ∧ during(e, t) ∧ took(x, e) ∧ subject(y, e)*)
• pass/2 - a binary relation, pass(x, y) iff x passes the subject y

• Spring2001 - a constant denoting the time interval spring 2001

• bestScore/2 - a binary function which identifies the best score in a subject during a given time interval.

• greaterThan/2 - a binary relation which has the same meaning as >

• equals/2 - a binary relation which has the same meaning as = (note that it’s acceptable to assume that we are using first-order logic with equality, provided that the student knows what this means).

• numOfStudents/2 - a binary function which identifies the number of students in taking a subject during a given time interval.

• barber/1 - a unary relation, barber(x) iff x is a barber.

• shaves/2 - a binary relation, shaves(x, y) iff x shaves y.

Given this vocabulary you can represent the sentences as follows:

1. \( \exists x. \text{student}(x) \land \text{took}(x, \text{French}, \text{Spring}2001) \)

2. \( \forall x, t. \text{student}(x) \land \text{took}(x, \text{French}, t) \Rightarrow \text{pass}(x, \text{French}) \) (this is slightly vague, since a student could fail and then pass on the second attempt)

3. equals(numOfStudents(Greek, Spring2001), 1) (the challenge here is to represent the cardinality of the set of students taking Greek during spring 2001). An alternative is \( \exists x. \text{student}(x) \land \text{took}(x, \text{Greek}, \text{Spring}2001) \land (\forall y. \text{took}(y, \text{Greek}, \text{Spring}2001) \Rightarrow x = y) \). This formula asserts that there exists a student who took Greek in spring 2001 and if there is anything else which took Greek in spring 2001 then it must be this student. So it would be impossible to satisfy this sentence if less than one student took Greek in spring 2001, since we have asserted the existence of at least one student with this property. But also impossible to satisfy it if more than one student took Greek in spring 2001, since in that case there would be a \( y \) such that \( \text{took}(y, \text{Greek}, \text{Spring}2001) \land y \neq x \).

4. \( \forall t. \text{greaterThan}(\text{bestScore}(\text{Greek}, t), \text{bestScore}(\text{French}, t)) \)

5. \( \exists x. \text{barber}(x) \land \forall y. \neg \text{shaves}(y, y) \Rightarrow \text{shaves}(x, y) \) - (almost) Russell’s paradox, there is no barber with this property as if there was then it would be possible to prove that the \( \text{shaves}(\text{Barber}, \text{Barber}) \) and \( \neg \text{shaves}(\text{Barber}, \text{Barber}) \) (N.B. Russell’s paradox is that there is a barber who shaves all and only the men who do not shaves themselves - this is an equivalence, \( \Leftrightarrow \), rather than an implication)

Part 2: One possible sentence is \( \forall x. P(x) \land \neg \exists x. x \neq A \land P(A) \); which means that for all objects property \( P \) holds and there are no objects not equal to object \( A \) such that property \( P \) holds. So if this sentence is true then there can only be one object, \( A \), in the domain of interpretation. If there
were any other objects then they would have to have property \( P \) and not have property \( P \) in order to satisfy this sentence, which is impossible. A simpler alternative is \( \exists x \forall y. x = y \); which means that there exists an object such that all other objects are equivalent to this object, so there is only one unique object in the domain.

2 Most General Unifier (MGU)

The most general unifier (MGU) is the least constrained substitution that makes two clauses unify with each other. What is the MGU for each pair of clauses below? If there is no MGU, explain why.

The Unify algorithm in figure 1 (also in R&N Section 9.2, p.328.)

1. \( p(A, B, B) \) and \( p(x, y, z) \)
2. \( q(y, g(A, B)) \) and \( q(g(x, x), y) \)
3. \( \text{older}(father(y), y) \) and \( \text{older}(father(x), \text{John}) \)
4. \( \text{knows}(\text{father}(y), y) \) and \( \text{knows}(x, x) \)

Note that, constants are upper case (e.g. \( A, B \)) and variables are lower case (e.g. \( x, y, z \)).

Tutor’s Guide

1. \( x/A, y/B, z/B \)
2. Unification fails. Start with the (partial) substitution \( y/g(x,x) \), then add \( x/A \) to get \( y/g(x,x), x/A \). At this point, one clause is \( q(g(A, A), g(A, B)) \), and the other \( q(g(A, A), g(A, A)) \). Since \( q(A, A) \) cannot be unified with \( q(A,B) \) unification fails.
3. \( x/\text{John}, y/\text{John} \)
4. Unification fails. Start with (partial) substitution \( x/\text{father}(y) \). One clause is now \( \text{knows}(\text{father}(y), y) \), and the other \( \text{knows}(\text{father}(y), \text{father}(y)) \). Unification fails here because we can’t unify \( y \) and \( \text{father}(y) \), due to the occurs check.

3 Generalised Modus Ponens

Part 1: Convert the following sentences to first-order logic formulae suitable for use with Generalised Modus Ponens.
1. Horses, cows and pigs are mammals.
2. An offspring of a horse is a horse.
3. Bluebeard is a horse.
4. Bluebeard is Charlie’s parent.
5. Offspring and parent are inverse relations.

Part 2: Use the sentences to answer a query using a backward-chaining algorithm.
- Draw the proof tree generated by an exhaustive backward-chaining algorithm for the query $\text{Horse}(h)$, where clauses are matched in the order given.
- How many solutions are a logical consequence of your knowledge base?
- How could we solve this problem?

**Tutor’s Guide**

Part 1:

1. $\text{Horse}(x) \Rightarrow \text{Mammal}(x)$
   
   $\text{Cow}(x) \Rightarrow \text{Mammal}(x)$
   
   $\text{Pig}(x) \Rightarrow \text{Mammal}(x)$

2. $\text{Offspring}(y, x) \land \text{Horse}(x) \Rightarrow \text{Horse}(y)$ (y is offspring of x)

3. $\text{Horse}(\text{Bluebeard})$

4. $\text{Parent}(\text{Bluebeard}, \text{Charlie})$ (x is parent of y)

5. $\text{Offspring}(x, y) \Rightarrow \text{Parent}(y, x)$
   
   $\text{Parent}(x, y) \Rightarrow \text{Offspring}(y, x)$

Part 2:

This question deals with the problem of looping in backward-chaining proofs. The proof tree is shown in Figure 2.

The branch with $\text{Offspring}(\text{Bluebeard}, y)$ and $\text{Parent}(y, \text{Bluebeard})$ repeats indefinitely, so the rest of the proof is never reached.

We get an infinite loop because of rule 2, $\text{Offspring}(x, y) \land \text{Horse}(y) \Rightarrow \text{Horse}(x)$.

The specific loop appearing in the figure arises because of the ordering of the clauses. We could be order $\text{Horse}(\text{Bluebeard})$ before rule 2, which solve the problem of finding $\text{Horse}(\text{Bluebeard})$ as a possible answer to the query.
9 INFEREN:E IN FIRST-ORDER LOGIC

function UNIFY(x, y, θ) returns a substitution to make x and y identical
inputs:
  x, a variable, constant, list, or compound
  y, a variable, constant, list, or compound
  θ, the substitution built up so far (optional; defaults to empty)

if θ = failure then return failure
else if x = y then return θ
else if VARIABLE(x) then return UNIFYVAR(x, y, θ)
else if VARIABLE(y) then return UNIFYVAR(x, y, θ)
else if COMPOUND(x) and COMPOUND(y) then
  return UNIFY(ARGS(x), ARGS(y), UNIFY(ORDER(x), ORDER(y), θ))
else if List(x) and Last(y) then
  return UNIFY(REST(x), REST(y), UNIFY(REST(x), REST(y), θ))
else return failure

function UNIFYVAR(var, z, θ) returns a substitution
inputs:
  var, a variable
  z, an expression
  θ, the substitution built up so far

if (var/z) ∈ θ then return UNIFY(var, z, θ)
else if (y/z) ∈ θ then return UNIFY(var, z, θ)
else if OCCURS(var, z) then return failure
else return add (var/z) to θ

Figure 1: Unification Algorithm.
Figure 2: Solution to the Generalised Modus Ponens problem.