Homework #3 Solutions

1. [50 pts., 5 pts. each] There may be multiple right answers for each of these, especially since there are multiple ways to write semantically equivalent sentences.

\[ \exists x \text{ Student}(x) \land \text{TakesCourse}(x,F,S) \]

b) Every student who takes French passes it.
\[ \forall x, y \text{ Student}(x) \land \text{TakesCourse}(x,F,y) \Rightarrow \text{Passes}(x,F) \]

c) Only one student took Greek in Spring 2001.
\[ \exists x \text{ Student}(x) \land \text{TakesCourse}(x,G,S) \land \neg \exists y ( \text{TakesCourse}(y,G,S) \land x \neq y ) \text{ or } \exists x \text{ Student}(x) \land \text{TakesCourse}(x,G,S) \land \forall y ( \text{TakesCourse}(y,G,S) \Rightarrow x = y ) \]

d) The best score in Greek is always higher than the best score in French.
\[ \forall t \text{ Semester}(t) \Rightarrow \exists x_g, s_g \text{ Score}(x_g,G,t,s_g) \land [ \forall x_f, s_f \text{ Score}(x_f,F,t,s_f) \Rightarrow \text{GreaterThan}(s_g,s_f) ] \]

Here we have rephrased it as “For any semester, there is a score in Greek that is better than all scores in French”. A more literal translation would be:
\[ \forall t \exists x_g, s_g \text{ Score}(x_g,G,t,s_g) \land [ \forall y_g, s_y \text{ Score}(y_g,G,t,s_y) \Rightarrow \text{GreaterThan}(s_g,s_y) ] \land [ \exists x_f, s_f \text{ Score}(x_f,F,t,s_f) \land [ \forall y_f, s_y \text{ Score}(y_f,F,t,s_y) \Rightarrow \text{GreaterThan}(s_f,s_y) ] \land \text{GreaterThan}(s_g,s_f) ] \]

e) Every person who buys a policy is smart.
\[ \forall x, y ( \text{Person}(x) \land \text{Policy}(y) \land \text{Buys}(x,y) ) \Rightarrow \text{Smart}(x) \]

f) No person buys an expensive policy.
\[ \forall x, y ( \text{Person}(x) \land \text{Policy}(y) \land \text{Expensive}(y) ) \Rightarrow \neg \text{Buys}(x,y) \]

g) There is an agent who sells policies only to people who are not insured.
\[ \exists x \text{ Agent}(x) \land ( \forall y, z \text{ Policy}(y) \land \text{Person}(z) \land \text{Sells}(x,y,z) \Rightarrow \neg \text{Insured}(z) ) \text{ or } \exists x \text{ Agent}(x) \land ( \forall y, z \text{ Policy}(y) \land \text{Sells}(x,y,z) \Rightarrow \text{Person}(z) \land \neg \text{Insured}(z) ) \]

h) There is a barber who shaves all men in town who do not shave themselves.
\[ \exists x \text{ Barber}(x) \land ( \forall \text{LocalMan}(y) \land \neg \text{Shaves}(y,y) \Rightarrow \text{Shaves}(x,y) ) \]

i) A person born in the UK, each of whose parents is a UK citizen or a UK resident, is a UK citizen by birth.
\[ \forall x \text{ Person}(x) \land \text{BornIn}(x,UK) \land ( \exists y, z \text{ Parent}(x,y) \land \text{Parent}(x,z) \land y \neq z \land ( \text{CitizenOf}(y,UK) \lor \text{ResidentOf}(y,UK) ) \land ( \text{CitizenOf}(z,UK) \lor \text{ResidentOf}(z,UK) ) ) \Rightarrow \text{CitizenByBirth}(x,UK) \]

or
\[ \forall x \text{ Person}(x) \land \text{BornIn}(x,UK) \land ( \forall y \text{ Parent}(x,y) \Rightarrow ( \text{CitizenOf}(y,UK) \lor \text{ResidentOf}(y,UK) ) ) \Rightarrow \text{CitizenByBirth}(x,UK) \]

j) A person born outside the UK, one of whose parents is a UK citizen by birth, is a UK citizen by descent.
\[ \forall x \text{ Person}(x) \land \neg \text{BornIn}(x, \text{UK}) \land (\exists y \text{ Parent}(x,y) \land \text{CitizenByBirth}(y, \text{UK})) \Rightarrow \text{CitizenByDescent}(x, \text{UK}) \]

2. \[10 \text{ pts.}\]
\[ \forall x, y \, x = y \]
This works because first-order logic assumes a non-empty domain. Thus there is at least one object. The universal statement about equality ensures that all objects must be the same. Thus exactly one.
The following also works:
\[ \exists x, \forall y \, x = y \]
A slightly more complicated way is to introduce a predicate, saying there is an object for which the predicate holds, there is no object where the predicate does not hold, and (if it does hold, the object is not different from x):
\[ \exists x \, P(x) \land \neg \exists y \, \neg P(y) \lor x \neq y \]

Grading:
- 3 pts for using equality
- 3 pts. for getting quantifiers right (either two universal, or an existential followed by a universal)
- 4 pts. for putting it all together correctly

3. \[40 \text{ pts. total}\]

a) \[25 \text{ pts.}\] 2 pts. for each rule 1/2 pt. for each fact, round up points off, 2 pts for program that load correctly

/* Homework#3 Problem3 key
   This program creates a knowledge base about the world of Harry Potter. */

/* Rules (9 total) */
/* If someone teaches at a wizard academy, then they are a wizard. */
wizard(P) :- teach(P,S), wizAcad(S).

/* Every child of a wizard is also a wizard. */
wizard(C) :- parent(P,C), wizard(P).

/* If someone is not a wizard, then they are a muggle. */
muggle(M) :- not(wizard(M)).

/* If a Hogwarts student is shifty, then they are in Slytherin house. */
member(S,slytherin) :- studentOf(S,hogwarts), shifty(S).

/* If a Hogwarts student is brave, then they are in Gryffindor house. */
member(S,gryffindor) :- studentOf(S,hogwarts), brave(S).

/* The head of a house is a member of the house. */
member(P,H) :- head(P,H).

/* The head of a house teaches at the school the house belongs too. */
teach(P,S) :- head(P,H), partof(H,S).

/* Any member of a house at a wizard academy is a wizard. */
wizard(M) :- member(M,H), partof(H,S), wizAcad(S).

/* Hermione is in the same house as Harry. */
member(hermione, H) :- member(harry,H).
/* Facts(11 total) */
/* Hogwarts is a wizard academy. */
wizAcad(hogwarts).

/* James is a wizard. */
wizard(james).

/* McGonagal is head of Gryffindor house. */
head(mcgonagal,gryffindor).

/* Snape teaches at Hogwarts. */
teach(snape,hogwarts).

/* One of Ron’s parents is Arthur. */
parent(arthur,ron).

/* One of Harry’s parents is James. */
parent(james,harry).

/* Harry and Draco are Hogwarts students. */
studentOf(harry, hogwarts).
studentOf(draco, hogwarts).

/* Harry is brave. */
brave(harry).

/* Draco is shifty. */
shifty(draco).

/* Gryffindor is part of Hogwarts. */
partOf(gryffindor,hogwarts).

b). [15 pts.] i-iii 2 pts. each, iv-vi 3 pts. each
i. => wizard(harry).
   true.
ii. => muggle(ron).
   true.
iii. => member(hermione,H).
      H = gryffindor.
iv. => teach(P,hogwarts).
    P = mcgonagal ;
    P = snape.
v. => member(harry,X), member(mcgonagal,X).
    X = gryffindor.
vi. => member(X,gryffindor).
    X = harry ;
    X = mcgonagal ;
    X = hermione.