Homework 2 solutions

1. Determine whether the following statements are well-formed formulae in Predicate Logic.
   (a) $P \rightarrow Q \vee R$
      a. Well-formed (propositional logic is a subset of predicate logic)
   (b) $\forall x\, P(x) \rightarrow \exists y$
      a. Not well-formed (variable y has no corresponding sentence)
   (c) $\exists x\, P(x) \rightarrow \forall x P(x)$
      a. Well-formed (the two x's are not related to each other)

2. Use the following predicates to write first-order Predicate Logic formulas:
   P(x) = x is a programmer
   S(x) = x is smart

   (a) There are programmers. $\exists x P(x)$
   (b) Everyone is a programmer. $\forall x P(x)$
   (c) Everyone is not a programmer (i.e., every person is not a programmer) $\forall x {\sim} P(x)$
   (d) Not everyone is a programmer. ${\sim}\forall x P(x) \ \equiv \exists x {\sim} P(x)$
   (e) Someone is not a programmer. $\exists x {\sim} P(x)$
   (f) All programmers are smart. $\forall x P(x) \rightarrow S(x)$
   (g) There are some programmers who are smart. $\exists x P(x)\ \wedge\ S(x)$
   (h) No programmer is smart. ${\sim}\exists x P(x)\ \wedge\ S(x)\ \equiv\ \forall x {\sim} P(x)\ \vee\ {\sim}S(x)\ \equiv\ \forall x P(x) \rightarrow {\sim}S(x)$
   (i) Some programmers are not smart. $\exists x P(x)\ \wedge\ {\sim}S(x)$

3. What is the result of unifying the following pairs of predicate logic expressions? (Determine if unification fails, or if it succeeds give the substitution). Note: variables start with a small letter, Constants start with a Capital letter.
   a) Mother(x) and Parent(x)
      ○ Fail
   b) Parent(Mary, John) and Parent(x, y)
      ○ x <- Mary, y <- John; Parent(Mary,John)
   c) Parent(Mary, z) and Parent(y, John)
      ○ y <- Mary, z <- John; Parent(Mary,John)
   d) Parent(x, z) and Parent(y, John)
      ○ x <- y, z <- John; Parent(y,John)
   e) Parent(x, x) and Parent(y, John)
      ○ x <- y, y <- John; Parent(y,John)
   f) Parent(x, x) and Parent(y, Father(y))
      ○ Fail
   g) Parent(x, y) and Parent(y, Father(y))
      ○ To reduce confusion, renaming variables: Parent(x1, y1) and Parent(y, Father(y))

    - x1 <- y, y1 <- Father(y); Parent(y,Father(y))

h) Parent(x, z) and Parent(y, Father(y))
    - x <- y, z <- Father(y); Parent(y, Father(y))

4. The attached **`family.clp`** CLIPS program describes a set of parent-child pairs and rules to identify siblings. Remember to (reset) to load the facts.

    a. Add one new rule to family.clp to print a list of all people who are parents. Do not add/remove facts. [Show rule and output of the program]

```
    (defrule parent-print
      (parent ? ?x)
    =>
      (printout t ?x " is a parent" crlf )
    )
CLIPS> (reset)
CLIPS> (run)
Sanjay is a parent
Priyanka and Rahul are siblings
Sonia is a parent
Sonia is a parent
Rajeev is a parent
Rajeev is a parent
Sanjay and Rajeev are siblings
Indira is a parent
Indira is a parent
CLIPS>
```

    b. Add 1-2 new rules to family.clp to print a list of all pairs of persons who are cousins *and* assert new facts of the form (cousin Varun Rahul). Two persons are cousins if their parents are siblings. Do not print or assert duplicate pairs. [Show rule(s) and output of the program]

```
(defrule cousin
  (sibling ?x ?y)
  (parent ?a ?x)
  (parent ?b ?y)
=>
  (assert (cousin ?a ?b))
  (printout t ?a " and " ?b " are cousins" crlf )
)

CLIPS> (reset)
CLIPS> (run)
Sanjay is a parent
```

```
Priyanka and Rahul are siblings
Sonia is a parent
Sonia is a parent
Rajeev is a parent
Rajeev is a parent
Sanjay and Rajeev are siblings
Varun and Priyanka are cousins
Varun and Rahul are cousins
Indira is a parent
Indira is a parent
CLIPS>
```

5. The **`family.pl`** Prolog program on Canvas describes a set of facts and rules to identify some family relationships. `parent(pam, bob)` means that pam is the parent of bob.

   a. Add one new rule to family.pl that will enable querying all `(aunt, nephew/niece)` pairs. [Show rule and output of the program]

   ```
   aunt(Y,X) :- parent(Xp,X), parent(Z,Y), parent(Z,Xp), female(Y), Y
   \== Xp.
   ?- aunt(X,Y).
   X = liz,
   Y = ann ;
   X = liz,
   Y = pat ;
   X = ann,
   Y = jim ;
   false.
   ```

   b. Add a new procedure to family.clp that will enable querying all `(maternal_ancestor, person)` pairs. A maternal ancestor of a person is an ancestor via the person's mother's side, e.g., mother, mother's father, mother's grandfathers, mother's grandmothers, ... [Show rules and output of the program]

   ```
   maternal_ancestor(X,Z) :- mother( Y, Z), ancestor( X, Y).
   maternal_ancestor( X, Z):- mother( X, Z).
   ?- maternal_ancestor(X,Y).
   X = pam,
   Y = jim ;
   X = tom,
   Y = jim ;
   X = bob,
   Y = jim ;
   X = pam,
   Y = bob ;
   ```

```
X = pat,
Y = jim.
```

6. Represent the following scenario using **Propositional Logic.**

KB: "CPSC375 has CPSC131 and MATH338 as prerequisites. CPSC481 has CPSC335 and MATH338 as prerequisites. CPSC483 has CPSC375 as a prerequisite. A course can only be taken if all the prerequisites have been taken. If a course has been taken, then all of its prerequisites have also been taken. A course cannot be taken again."

a. List the propositional symbols you will use and their meaning.

   i. Taken375: CPSC 375 has been taken
   ii. Taken131: CPSC 131 has been taken
   iii. Taken338: MATH 338 has been taken
   iv. Taken481: CPSC 481 has been taken
   v. Taken335: CPSC 335 has been taken
   vi. Taken483: CPSC 483 has been taken
   vii. Cantake375: Can take CPSC 375
   viii. Cantake131: Can take CPSC 131
   ix. Cantake338: Can take MATH 338
   x. Cantake481: Can take CPSC 481
   xi. Cantake335: Can take CPSC 335
   xii. Cantake483: Can take CPSC 483

b. Represent the KB as a set of Propositional Logic sentences.

   i. Taken131 ∧ Taken338 → Cantake375
   ii. Taken335 ∧ Taken338 → Cantake481
   iii. Taken375 → Cantake483
   iv. Taken375 → Taken131 ∧ Taken338
   v. Taken481 → Taken335 ∧ Taken338
   vi. Taken483 → Taken375
   vii. Taken375 → ~Cantake375
   viii. Taken131 → ~Cantake131
   ix. Taken338 → ~Cantake338
   x. Taken481 → ~Cantake481

xi. Taken335 → ~Cantake335

xii. Taken483 → ~Cantake483

c. A student has taken CPSC481. Represent the following queries in Propositional Logic. What will be the result of proving each of these queries? [Do not give a proof. Only give the query and whether you expect the result to be Valid/Invalid.]

Taken481

i. Can this student take CPSC375?
   - Query: Cantake375. Invalid.

ii. Can this student take CPSC481?
   - Query: Cantake481. Invalid.
   - Note: Query: ~Cantake481. Valid.

iii. Can this student take CPSC483?
   - Query: Cantake483. Invalid.

iv. Has this student taken CPSC 375?
   - Query: Taken375. Invalid.

d. A student has taken CPSC131 and CPSC481. Represent the following queries in Propositional Logic. What will be the result of proving each of these queries? [Do not give a proof. Only give the query and whether you expect the result to be Valid/Invalid.]

Taken131

Taken481

i. Can this student take CPSC375?
   - Query: Cantake375. Valid.

ii. Can this student take CPSC481?
   - Query: Cantake481. Invalid.
   - Note: Query: ~Cantake481. Valid.

iii. Can this student take CPSC483?
   - Query: Cantake483. Invalid.

iv. Has this student taken CPSC 375?

- Query: Taken375. Invalid.

7. Answer all parts of the previous question but use **Predicate Logic**. For (a), give the predicates you will use and their meaning.
   a. List the predicates and constants you will use and their meaning.
      i. Taken(x): Course x has been taken
      ii. CanTake(x): Course x can be taken
      iii. Prereq(c,p): Course c has a prerequisite course p

   b. Represent the KB as a set of Predicate Logic sentences.
      i. Prereq(CPSC375, CPSC131)
      ii. Prereq(CPSC375, MATH338)
      iii. Prereq(CPSC481, CPSC335)
      iv. Prereq(CPSC375, MATH338)
      v. Prereq(CPSC483, CPSC375)
      vi. $\forall c$ ($\forall p$ Prereq(c,p) → Taken(p)) ∧ $\sim Taken(c)$ → CanTake(c)
      vii. $\forall c$ $\forall p$ Taken(c) ∧ Prereq(c,p) → Taken(c)

   c. A student has taken CPSC481. Represent the following queries in Propositional Logic. What will be the result of proving each of these queries? [Do not give a proof. Only give the query and whether you expect the result to be Valid/Invalid.]

      Taken(CPSC481)

      i. Can this student take CPSC375?
         - Query: CanTake(CPSC375). Invalid.
      ii. Can this student take CPSC481?
         - Query: CanTake(CPSC481). Invalid.
         - Note: Query: ~CanTake(CPSC481). Valid.
      iii. Can this student take CPSC483?
         - Query: CanTake(CPSC483). Invalid.
      iv. Has this student taken CPSC 375?
         - Query: CanTake(CPSC375). Invalid.

d. A student has taken CPSC131 and CPSC481. Represent the following queries in Propositional Logic. What will be the result of proving each of these queries? [Do not give a proof. Only give the query and whether you expect the result to be Valid/Invalid.]

   Taken(CPSC131)

   Taken(CPSC481)

   i. Can this student take CPSC375?
      - Query: CanTake(CPSC375). Valid.

   ii. Can this student take CPSC481?
      - Query: CanTake(CPSC481). Invalid.
      - Note: Query: ~CanTake(CPSC481). Valid.

   iii. Can this student take CPSC483?
      - Query: CanTake(CPSC483). Invalid.

   iv. Has this student taken CPSC 375?
      - Query: CanTake(CPSC375). Invalid.

8. Write a CLIPS program that will answer the queries to the situations above.
   a. Give the CLIPS program which will include:
      i. Facts. **Already given to you in `courses.clp` on Canvas**
      ii. Rules that will print a list of courses that can be taken and have already been taken. To be added.
      iii. To run the program, you will load the program (and reset), assert the user-specific facts (e.g., the student has taken CPSC481), and then call run.

```
(deffacts courses "some courses and their prerequisites"
  (course CPSC131)
  (course CPSC335)
  (course CPSC375)
  (course CPSC481)
  (course CPSC483)
  (course MATH338)
  (prereq CPSC375 CPSC131)
  (prereq CPSC375 MATH338)
  (prereq CPSC481 CPSC335)
  (prereq CPSC481 MATH338)
  (prereq CPSC483 CPSC375)
)
```

```
(defrule cantake
  (course ?c)
  (not (and (prereq ?c ?p) (not (hastaken ?p))))
  (not (hastaken ?c))
=>
  (printout t "Can take course " ?c crlf)
  (assert (cantake ?c))
)

(defrule hastaken
  (course ?c)
  (hastaken ?c)
  (prereq ?c ?p)
=>
  (printout t "Has taken course " ?p crlf)
  (assert (hastaken ?p))
)
```

```
CLIPS> (load "courses.clp")
CLIPS> (reset)
CLIPS> (assert (hastaken CPSC131))
<Fact-12>
CLIPS> (run)
Can take course MATH338
Can take course CPSC335
CLIPS> (reset)
CLIPS> (assert (hastaken CPSC131))
<Fact-12>
CLIPS> (assert (hastaken CPSC481))
<Fact-13>
CLIPS> (run)
Has taken course MATH338
Can take course CPSC375
Has taken course CPSC335
CLIPS>
```

b. If new courses were added along with their prerequisites, what changes would your program need to answer similar queries? Answer in 3-5 sentences.

Only some new facts would have to be added. Specifically, these would be one (course) fact for each course, and a set of (prereq) facts for each prerequisite. The rules do not have to be changed.

9. Write a Prolog program that will answer the queries to the situations above.

a. Give the Prolog program which will include:

   i. Facts. **Already given to you in `courses.pl` on Canvas**

   ii. Rules that can be used to answer the queries: (1) all courses that can be taken, and (2) all courses that have already been taken. To be added.

   iii. To run the program, you will add the user-specific facts (e.g., the student has taken CPSC481) to the program file[1], consult the program file, and then call the query.

```
course(cpsc131).
course(cpsc335).
course(cpsc375).
course(cpsc481).
course(cpsc483).
course(math338).
prereq(cpsc375, cpsc131).
prereq(cpsc375, math338).
prereq(cpsc481, cpsc335).
prereq(cpsc481, math338).
prereq(cpsc483, cpsc375).

hastaken(cpsc481).
hastaken(cpsc131).
hastaken(P) :- course(C), course(P), prereq(C,P), hastaken(C).

has_missing_prereq(C) :- course(C), prereq(C,P), not(hastaken(P)).

cantake(C) :- course(C), not(hastaken(C)), not(has_missing_prereq(C)).

?- hastaken(X).
X = cpsc481 ;
X = cpsc131 ;
X = cpsc335 ;
X = math338 ;
false.

?- cantake(X).
X = cpsc375 ;
false.
```

b. If new courses were added along with their prerequisites, what changes would your program need to answer similar queries? Answer in 3-5 sentences.

Only some new facts would have to be added. Specifically, these would be one course() fact for each course, and a set of prereq() facts for each prerequisite. The rules do not have to be changed.

[1] You can also assert user-specific facts from the console using `assertz()` similar to CLIPS. E.g., `assertz(hastaken(cpsc481))`