# CPSC 449: Prolog

Fall 2020

Consult the D2L site for the due date/time.

1. *[30%]* The Java language supports single inheritance with the class hierarchy, but multiple inheritance with its interface hierarchy.

   - A class **`extends`** its immediate superclass. Every class has at most one immediate superclass.
   - A class **`implements`** its immediate super-interfaces. Every class has zero or more immediate superinterfaces.
   - An interface **`extends`** its immediate super-interfaces. Every interface has zero or more immediate superinterfaces.

   We represent a Java classes/interfaces and their inheritance hierarchy using four Prolog predicates.

   - **`class(X)`** iff **`X`** is a class.
   - **`interface(X)`** iff **`X`** is an interface.
   - **`extends(X, Y)`** iff either (i) class **`X`** extends class **`Y`**, or (ii) interface **`X`** extends interface **`Y`**.
   - **`implements(X, Y)`** iff class **`X`** implements interface **`Y`**

   It is assumed that we are given a database populated with facts specifying the relationships between some Java classes and interfaces.

   (a) *[15%]* Define a Prolog predicate **`subclass(?X, ?Y)`**, which succeeds iff **`X`** and **`Y`** are both classes, and **`X`** is related to **`Y`** via the transitive closure of the **`extends`** relation. For example, **`subclass(a, c)`** holds if the following facts are in the database:

   ```
   class(a)
   class(b)
   class(c)
   extends(a, b)
   extends(b, c)
   ```

   In the above example, **`a`** is said to be a subclass of **`c`**, and **`c`** is said to be a superclass of **`a`**. A trivial case of subclassing is when one class extends another class.

(b) *[15%]* Define a Prolog predicate **`superinterface(?Y, ?X)`**, which succeeds iff **`Y`** is an interface, **`X`** is a class, and either **`X`** or a superclass of **`X`** implements an interface **`Z`**, such that either **`Z`** is **`Y`**, or **`Z`** is related to **`Y`** via the transitive closure of **`extends`**. For example, **`superinterface(f, a)`** holds if the following facts are in the database:

```
class(a)
class(b)
class(c)
interface(d)
interface(e)
interface(f)
extends(a, b)
extends(b, c)
implements(c, d)
extends(d, e)
extends(e, f)
```

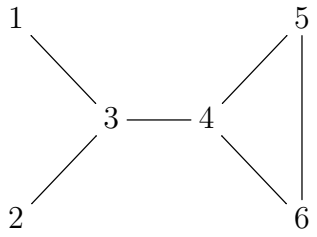
A trivial case in which **`superinterface(Y, X)`** holds is when **`X`** implements **`Y`**.

2. *[40%]* Given an undirected graph $G = (V, E)$, a subset $S \subseteq V$ of vertices is called a ***vertex cover*** of $G$ if there does not exists an edge $uv$ in $E$ such that neither $u$ nor $v$ is in $S$. People are interested in finding out if a given graph $G$ has a vertex cover of size $k$ or less, for some parameter $k$.

One can represent an undirected graph in Prolog as a list of pairs, each pair encoding an *undirected* edge. As an example, consider the following Prolog list:

```
[(1,3), (2,3), (3,4), (4,5), (4,6), (5,6)]
```

This list represents the graph below:




A vertex cover can be represented as a list of vertices. For example, the following is a vertex cover of the graph above:

```
[3, 5, 6]
```

In fact, the vertex cover above is one of the smallest vertex covers for the graph in question. The following is not a vertex cover, because the edge **`(5,6)`** is not "covered."

```
[3, 4]
```

Develop a Prolog predicate, **`vertex_cover(+Graph, ?Cover)`**, which asserts that the list **`Cover`** is a vertex cover of undirected graph **`Graph`**. Your implementation shall satisfy the following requirements:

- If **`Cover`** is not fully instantiated, then the implementation shall generate all possible vertex covers that unify with **`Cover`**.
- It is okay to end up generating the same vertex cover multiple times. (Requiring uniqueness will significantly increase the difficulty of this question.)
- The implementation shall not perform a naive generate-and-test (e.g., generate every subset of the vertex set of **`Graph`**, and then test if the generated subset is a vertex cover). Early pruning of the search space shall be attempted if possible. A naive generate-and-test solution is only worth 66% of the marks.

The predicate above can be used for checking if there exists a vertex cover for a given graph $G$ where the cover is of size no bigger than some positive integer constant $k$. For example, the following query does exactly this:

**`?- length(Cover,`** $k$**`), vertex_cover(`**$G$**`, Cover).`**

Alternatively, if $k$ is small, say 3, one can also issue the following query to check if there is a vertex cover of size 3:

**`?- vertex_cover(`**$G$**`, [V1, V2, V3]).`**

3. *[30%]* Write a Prolog program to solve the following logic puzzle. There are five houses, each of a different color and inhabited by a man of a different nationality, with a different pet, drink, and brand of cigarettes.

   (a) The Englishman lives in the red house.
   (b) The Spaniard owns the dog.
   (c) Coffee is drunk in the green house.
   (d) The Ukrainian drinks tea.
   (e) The green house is immediately to the right (your right) of the ivory house.
   (f) The winston smoker owns snails.
   (g) Kools are smoked in the yellow house.
   (h) Milk is drunk in the middle house.
   (i) The Norwegian lives in the first house on the left.
   (j) The man who smokes Chesterfields lives in the house next to the man with the fox.
   (k) Kools are smoked in the house next to the house where the horse is kept.
   (l) The Lucky Strike smoker drinks orange juice.
   (m) The Japanese smokes Parliaments.
   (n) The Norwegian lives next to the blue house.

   Who owns the Zebra? Who drinks water?

   **Hint:** A naive generate-and-test solution is not considered an acceptable solution to this question. An incremental generate-and-test that is coupled with pruning is what I am looking for. Carefully document your data structures and algorithm.