Logic Programming
Tutorial 1: Prolog Warm-up

You should be able to do all of the unstarred exercises before the tutorial session. You should also be able to do exercises with a single star (*) but these may require more thought. Exercises with two stars (**) are harder and may involve material we haven't covered; try them if you have time.

1. **Unification**

   Without using a computer, find the most general unifiers of the following unification problems (or determine that they are not unifiable).

   (a) `X = f(Y,Y), Y = f(Z,Z), Z = f(W,W)`

   (b) `X = f(g(Y,Z), W), f(Y,W) = Z`

   (c) `X = f(g(Y,Z), W), f(X,Z) = W`

   (d) `X = f(Y), Y = f(Z), f(X,Y) = f(g(Z),W)`

   Compare your answers to Prolog's behavior. Are there any surprises?

2. **Basic queries**

   Start Sicstus and load the data in `simpsons.pl`, which defines some basic facts about the charcters in the Simpsons television program.

   Define predicates with the following behavior:

   (a) `person(X)`, which holds if `X` is a person. (It is safe to assume that all people in this example are either male or female.)

   (b) `employer(X)`, which says that `X` is an employer (at least one person works for `X`).

   (c) `parent(X,Y)`, which says that `X` is one of `Y`'s parents.

   (d) `grandparent(X,Y)`, which says that `X` is one of `Y`'s grandparents.

   (e) `sibling(X,Y)`, which holds if `X` and `Y` have a parent in common. (Technically, this should probably be called "half-sibling".)

   (f) `classmate(X,Y)`, which says that `X` and `Y` are taught by the same person.

   The predicates should have at most two rules and their bodies should be conjunctions of atomic formulas.

   Finally, write a goal that returns Bart's sibling's teacher's employer.

3. **Disjunction**

   The `parent` query in part (2) above involves *disjunction* (logical OR). In Prolog, disjunction can be represented in two ways. First, we can define multiple clauses that derive the same conclusion from different subgoals:

```
p_or_q :- p.
p_or_q :- q.
```

Alternatively, Prolog provides syntax for using disjunction within goals. Specifically, if `G1` and `G2` are goals, we may write `G1;G2` for the disjunction of the two goals.

```
p_or_q :- p ; q.
```

Prolog will try to solve the first goal and if that fails (or if we backtrack) it will then try the second goal. This syntax is helpful because sometimes we can use this to avoid writing helper predicates to deal with disjunction.

Define predicate called `parent_disj/2` whose meaning is the same as `parent/2` but uses disjunction. Compare its behavior (with and without tracing) to the two-clause version.

4. **Symmetry**

   The predicates `friend/2` and `neighbor/2` are not symmetric and they probably should be: if `X` is a friend or neighbor of `Y` then `Y` should be a friend or neighbor of `X` as well.

   One way to do this is simply to add rules such as

   ```
   friend(X,Y) :- friend(Y,X).
   neighbor(X,Y) :- neighbor(Y,X).
   ```

   Add these rules, and write a query that searches for the catchphrases of Milhouse's friend's parent's neighbor. Try to find all solutions. Is there a potential problem with this approach? How might we avoid this problem?

5. (*) **Goal ordering**

   Define predicates `aunt/2` and `uncle/2` in terms of `male`, `female`, `parent` and `sibling`. Observe the behavior of `aunt(X,bart)`, with or without tracing.

   Experiment with different goal orderings. Which goal ordering seems to be the most efficient for the above query?

6. (**) **Negation and inequality**

   The `classmate` and `sibling` predicates we defined above may give undesired answers. Try solving `classmate(X,X)` or `sibling(X,X)` to see examples of this problem.

   To avoid this problem, we want to eliminate the pairs `(X,X)` from the `classmate` relation to make it *irreflexive*. More generally, we often want to use *negation* to make exceptions to rules.

   (a) Prolog has a built-in *non-unifiability* predicate, written `t \= u`, that succeeds if `t` and `u` do *not* unify. Use `\=` to implement a version of `classmate` that is irreflexive. Do the same for `sibling`.

   What happens now in response to queries `?- classmate(X,X)` or `?- classmate(X,Y)`?

   (b) Prolog also has a *negation-as-failure* primitive, written `\+(G)` that tries to solve a goal `G`, and succeeds if `G` fails. Use negation-as-failure to define a predicate `people_with_no_catchphrase/1` that holds of all people with no catchphrase. Also, define a predicate `adults_with_no_catchphrase/1` that holds of all people who are not children and have no catchphrase.

   This should be possible with a rule with only two goals (one negated). Experiment with goal ordering — which ordering seems to have better behavior?