Prolog rules are Horn clauses, but they are written “backwards”, consider:

\[ \forall X, Y [ \text{female}(X) \land \text{parent}(X, Y) \rightarrow \text{mother}(X, Y) ] \]

is written in Prolog as

\[ \text{mother}(X, Y) :- \text{female}(X), \text{parent}(X, Y). \]

Implies ("think of \( \leftarrow \)"")

You can think of a rule as introducing a new “fact” (the head), but the fact is defined in terms of a compound goal (the body). That is, predicates defined as rules are only true if the associated compound goal can be shown to be true.

Prolog rules are implicitly universally quantified!
% a simple prolog program
female(pam).
female(liz).
female(ann).
female(pat).

male(tom).
male(bob).
male(jim).

parent(pam,bob).
parent(tom,bob).
parent(tom,liz).
parent(bob,ann).
parent(bob,pat).
parent(pat,jim).

mother(X,Y) :- female(X),parent(X,Y).

Queries:
?- mother(pam,bob).
?- mother(Z,jim).
?- mother(P,Q).
The same predicate name can be defined by multiple rules:

```
sibling(X,Y) :- sister(X,Y).
sibling(X,Y) :- brother(X,Y).
```
Consider the program relating humans to mortality:

\[
\text{mortal}(X) :- \text{human}(X).
\]
\[
\text{human}(<\text{Socrates}>).
\]

We can now pose the query:

\[
?- \text{mortal}(<\text{Socrates}>).
\]

True or false?
Declarative vs. Procedural Meaning

When interpreting rules purely as Horn clause logic statement ➔ declarative

When interpreting rules as “specialized queries” ➔ procedural

Observation: We design programs with declarative meaning in our minds, but the execution is performed in a procedural fashion.

Consider:

\[
\text{mother}(X,Y) :- \text{female}(X), \text{parent}(X,Y). 
\]
The unification operator: \(=/2\)

The expression \(A=B\) is true if \(A\) and \(B\) are terms and unify (look identical)

?- a = a.
  true
?- a = b.
  false
?- a = X.
  X = a
?- X = Y.
  true

Read Section 2 of Prolog Tutorial online
Lists & Pattern Matching

- Lists – a convenient way to represent abstract concepts
  - Prolog has a special notation for lists.

\[
\begin{align*}
[a] & \\
[a,b,c] & \\
[] & \\
[\text{bmw, vw, mercedes}] & \\
[\text{chicken, turkey, goose}] &
\end{align*}
\]
Lists & Pattern Matching

Pattern Matching in Lists

?- [ a, b ] = [ a, X ].
X = b

?- [ a, b ] = X.
X = [ a, b ]

But:

?- [ a, b ] = X.
X = [ a, b ]

The Head-Tail Operator: [H|T]

?- [a,b,c] = [X|Y];
X = a
Y = [b,c]

?- [a] = [Q|P];
Q = a
P = []
The predicate `first/2`: accept a list in the first argument and return the first element of the list in second argument.

```prolog
first(List,E) :- List = [H|T], E = H;
```
The predicate last/2: accept a list in the first argument and return the last element of the list in second argument.

Recursion: there are always two parts to a recursive definition; the base and the recursive step.

last([A],A).
last([A|L],E) :- last(L,E).
The append/3 predicate: accept two lists in the first two parameters, append the second list to the first and return the resulting list in the third parameter.

Hint: use recursion.

append([], List, List).
append([H|T], List, [H|Result]) :- append(T, List, Result).
Design the predicate \textit{halve/3} that takes a list as its first argument and returns two list each with half the elements of the original list (similar to the function \textit{halve} we studied in ML).
Homework

- Assignment 11: see website