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 \(\rightarrow\)”) and “and”

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:

\[
\text{sibling}(X,Y) :- \text{sister}(X,Y) . \\
\text{sibling}(X,Y) :- \text{brother}(X,Y).
\]
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 $\rightarrow$ declarative

When interpreting rules as “specialized queries” $\rightarrow$ procedural

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

Consider:

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

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

Pattern Matching in Lists

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

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

But:

?- [a, b] = [X].
no

The Head-Tail Operator: [H|T]

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

?- [a] = [Q|P];
Q = a
P = []
Lists - the First Predicate

The predicate first/2: accept a list in the first argument and return the first element of the list in second argument.

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).
```
The halve/3 Predicate

- Design the predicate `halve/3` that takes a list as its first argument and returns two lists each with half the elements of the original list (similar to the function `halve` we studied in ML).
Homework

- Assignment 11: see website