Logic as a Programming Language

- Logic can be considered the oldest programming language
- Aristotle invented propositional logic over 2000 years ago in order to prove properties of formal arguments
- Propositions - simple statements that are either true or false; e.g. Betty wears a white dress. Today is Sunday.
- Propositional Logic ≡ propositions + rules of inference
- Most famous inference rule: modus ponens

HW:
Read Section 1 online tutorial available on the CSC301 Prolog page. (first tutorial)

Let A and B be propositions, then

A implies B
A is true

∴ B is true

(1) **Inference** is the act or process of drawing a conclusion based solely on what one already knows.
(2) **Rule of inference** is a scheme for constructing valid inferences.

Propositional Logic

<u>Example</u>:

If Betty wears a white dress then today is Sunday.
Betty wears a white dress.

∴ Today is Sunday.

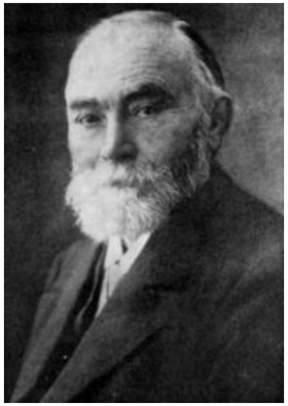
A fundamental problem with propositional logic is that it is not powerful enough to encode general knowledge - we would like to say things like:

<u>All</u> objects that are considered human are mortal.

Due to the fact that this sentence is not simple it can not be considered a proposition. But these kind of sentences are key in describing general knowledge.

Quantification

- In 1879 Gottlob Frege introduced the *predicate calculus* (‘Begriffsschrifft’)
- Today predicate calculus is more commonly known as First Order Logic.
- This logic solves the problems of propositional logic by introducing three new structures: predicates, universal quantification, and existential quantification.



Friedrich Ludwig Gottlob Frege
Philosopher and Logician

First-Order Logic

- Quantified Variables
  - Universally quantified variables

    ∀X – for All objects X

  - Existentially quantified variables

    ∃Y – there Exists an object Y

First-Order Logic

- Predicates
  - Predicates are functions that map their arguments into true/false
  - The signature of a predicate p(X) is

    p: Objects $\rightarrow$ { true, false }

  - Example: human(X)
    - human: Objects $\rightarrow$ { true, false }
    - human(tree) = false
    - human(paul) = true
  - Example: mother(X,Y)
    - mother: Objects $\times$ Objects $\rightarrow$ { true, false }
    - mother(betty,paul) = true
    - Mother(giraffe,peter) = false

First-Order Logic

- We can combine predicates and quantified variables to make statements on sets of objects
  - $\exists X[mother(X,paul)]$
    - there exists an object X such that X is the mother of Paul
  - $\forall Y[human(Y)]$
    - for all objects Y such that Y is human

First-Order Logic

- Logical Connectives: and, or, not
  - $\exists F \forall C$[parent(F,C) and male(F)]
    - There exists an object F for all object C such that F is a parent of C and F is male.
  - $\forall X$[day(X) and (wet(X) or dry(X))]
    - For all objects X such that X is a day and X is either wet or dry.

First-Order Logic

- If-then rules: $A \rightarrow B$
  - $\forall X \forall Y$[parent(X,Y) and womain(X) $\rightarrow$ mother(X,Y)]
    - For all objects X and for all objects Y such that if X is a parent of Y and X is a woman then X is a mother.
  - $\forall Q$[human(Q) $\rightarrow$ mortal(Q)]
    - For all objects Q such that if Q is human then Q is mortal.

Horn Clause Logic

In horn clause logic the form of the WFF's is restricted:

$$P_1 \land P_2 \land \ldots \land P_{n-1} \land P_n \rightarrow P_0$$

Single predicate in consequent

Conjunctions only!

Where P_0 , P_1 , P_2, … P_{n-1}, P_n are predicates.

Proving things is computation!

Use resolution to reason with horn clause expressions - resolution mimics the modus ponens using horn clause expressions.

Advantage: this can be done mechanically (Alan Robinson, 1965)

"Deduction is Computation"

J. Alan Robinson: A Machine-Oriented Logic Based on the Resolution Principle. J. ACM 12(1): 23-41 (1965)

Basic Prolog Programs

Facts - a fact constitutes a declaration of a truth; in Prolog it has to
to be a positive assertion.

Prolog Programs - a Prolog program is a collection of facts (…and rules,
as we will see later).

Example: a simple program

```
man(phil).
man(john).
woman(betty).
```

Facts, Prolog will treat these as true and enters them into its knowledgebase.

We execute Prolog programs by posing queries on its knowledgebase:

Prompt

```
?- man(phil).
   true - because Prolog can use its knowledgebase to prove true.
?- woman(phil).
   false - this fact is not in the knowledgebase.
```

Prolog ≡ Programming in Logic

Prolog - Queries & Goals

A query is a way to extract information from a logic program.

Given a query, Prolog attempts to show that the query is a logical consequence of the program; of the collection of facts.

In other words, a query is a goal that Prolog is attempting to satisfy (prove true).

When queries contain variables they are existentially quantified, consider

```
?- parent(X,liz).
```

The interpretation of this query is: prove that there is at least one object X that can be considered a parent of liz, or formally, prove that

$$\exists x[parent(x,liz)]$$

holds.

NOTE: Prolog will return all objects for which a query evaluates to true.

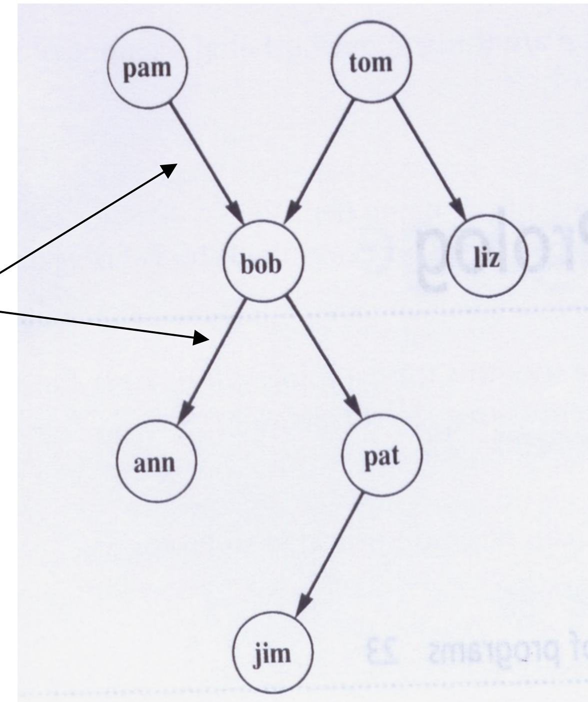
A Prolog Program

```
% a simple prolog program
woman(pam).
woman(liz).
woman(ann).
woman(pat).

man(tom).
man(bob).
man(jim).

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

Parent
Relation



A Family Tree

Example Queries:

```
?- woman(pam).
?- woman(X).        ∃X[woman(X)]?
?- parent(tom,Z).
?- father(Y).
```

Demo of ‘trace’ that demonstrates the search in Prolog

Compound Queries

A compound query is the conjunction of individual simple queries.

Stated in terms of goals: a compound goal is the conjunction of individual subgoals each of which needs to be satisfied in order for the compound goal to be satisfied. Consider:

?- parent(X,Y) , parent(Y,ann).

or formally,

$$\exists X,Y[parent(X,Y) \wedge parent(Y,ann)]$$

When Prolog tries to satisfy this compound goal, it will make sure that <u>the two Y variables always have the same values</u>.

Prolog uses <u>unification</u> and <u>backtracking</u> in order to find <u>all</u> the solutions which satisfy the compound goal.

Unification & Backtracking

- Unification is a special kind of pattern matching that instantiates variables with terms/objects.
- Backtracking allows Prolog to search for all unifications, called substitutions, that make a query true.