<u>Predicate Calculus *or* First-Order Logic</u>

Propositional Logic is very restrictive.
It can't even describe the basic rules of arithmetic.
That's how it manages to be both complete and sound.

We will depart here from the book's way of saying things so that we don't get confused later when we get to Logic programming.

Predicate calculus adds quite a few things on top of propositional logic

- Names beginning with little letters represent constants or functions or predicates
  - You can mostly tell which because a function name is always followed by (...), but functions look just like predicates. They just appear in different contexts.
- Constants are the most basic things
  - Constants do not have values, they stand for themselves
  - x means x and nothing more, it is one of the values we work with.
  - But they can represent things:
    - tom is just tom,
    - but we can use tom to represent a particular cartoon cat.
- Functions look like functions in programming languages, but they are not
  - Here's some functions being used: f(x), father(tom), g(x, y)
    - Functions are not called,
    - they do not have definitions,
    - and they don't return anything.
  - father(tom) = father(tom) and nothing else
  - but father(tom) can be used to represent the father of that cat
  - constants and functions are both kinds of Terms.
- Predicates are what do all the work,
  - They are sort-of called, and
  - they sort-of have values. Every predicate is either true or false
  - prime(X) might be used to find out whether X is prime or not.
  - likes(X, toast) can be used to state that X likes toast.
- Variables begin with capital letters
  - Variables can take on any value: constant or function.
- Equality: term = term is also a predicate
  - Terms are equal if they look the same.
- Universal quantification:
  - $\forall$ X, p(X) means that p(X) is true for all possible values of X
- Existential qualification:
  - $\exists$ X: p(X) means that there is at least one possible value of X that makes p(X) true.
- Here is a small but well-known example:
    - man(socrates)
    - $\forall$ X, man(X) $\Rightarrow$ mortal(X)
  - from which we can deduce:
    - mortal(socrates)

And there are some fairly obvious equivalences, for example:

- $\neg \exists$ X: *something* $\equiv \forall$ X, $\neg$ *something*
- $\neg \forall$ X, *something* $\equiv \exists$ X: $\neg$ *something*
- $\forall$ X: *something* $\equiv \neg \exists$ X: $\neg$ *something*
- $\exists$ X: *something* $\equiv \neg \forall$ X: $\neg$ *something*

An example, Kinship

- We have a collection of very basic facts:
  female(marge)
  female(lisa)
  male(homer)
  parent(marge, lisa)
  spouse(marge, homer)
  and so on
- And some other axioms:
  $\forall$ M, C, mother(M, C) $\Leftrightarrow$ female(M) $\wedge$ parent(M, C)
  $\forall$ H, W, husband(H, W) $\Leftrightarrow$ male(H) $\wedge$ spouse(H, W)
  $\forall$ A, B, spouse(A, B) $\Leftrightarrow$ spouse(B, A)
  $\forall$ P, C, parent(P, C) $\Leftrightarrow$ child(C, P)
  $\forall$ G, C, grandparent(G, C) $\Leftrightarrow \exists$ P: parent(G, P) $\wedge$ parent(P, C)
  $\forall$ A, B, sibling(A, B) $\Leftrightarrow$ A$\neq$B $\wedge \exists$ P: parent(P, A) $\wedge$ parent(P, B)
- From these, we should, in principle, be able to prove some Theorems:
  $\forall$ A, B, sibling(A, B) $\Leftrightarrow$ sibling(B, A) for example
- But how? Just another search?
  Gödel again

First-Order Definite Clauses

- Can always be written in the form Antecedents $\Rightarrow$ Consequence
- Antecedents are just ands of predicates
- Consequence is just a predicate
  king(X) $\wedge$ greedy(X) $\Rightarrow$ evil(X)
- When something is always true the antecedents and the arrow are left out
- A Datalog knowledge base: It is a crime for an American to sell weapons to a hostile nation; North Korea is an enemy of America; North Korea has some missiles; all of its missiles were sold to it by Colonel West; Colonel West is an American.
  american(X) $\wedge$ weapon(Y) $\wedge$ hostile(Z) $\wedge$ sold(X, Y, Z) $\Rightarrow$ criminal(X)
  missile(X) $\Rightarrow$ weapon(X)
  enemy(northkorea, america)
  enemy(X, america) $\Rightarrow$ hostile(X)
  owns(northkorea, m1)
  missile(m1)
  (m1 is a Skolem Constant, introduced because of $\exists$)
  missile(X) $\wedge$ owns(northkorea, X) $\Rightarrow$ sold(colonelwest, X, northkorea)

american(colonelwest)

Unifying and Substituting

- Function symbols are not used, just variables, constants, and predicates

  so Unifying is very simple:

  Unify(p(tom, X), p(Y, jerry)) = { X→jerry, Y→tom }
  Unify(p(tom, X), p(tom, Y)) = { X→Y } or maybe { Y→X }
  Unify(p(tom), p(tom)) = { }
  Unify(p(tom), p(jerry)) = fail
- The result of unify is a Substitution or Unifier
- Subst rewrites a formula F according to a substitution θ:

  Subst({ X→jerry, Y→tom }, hates(X, Y)) = hates(jerry, tom)
  Subst({ X→joe }, human(X) $\wedge$ wrong(Y)) = human(joe) $\wedge$ wrong(Y)
- StandardiseVariables(F) renames all of F's variables with totally new ones

  StandardiseVariables(human(X) $\wedge$ hates(Y, X)) =
  human(V179) $\wedge$ hates(V180, V179)

A simple Forward Chaining Inference Procedure

- KB is the Knowledge Base, all those facts above
- Question is a predicate, we want to know if it's true

```
while True:
  inferences = { }
  for rule in KB:
    StandardiseVariables(rule) to get (p1 ∧ p2 ∧ p3 ∧ … ∧ pn) ⇒ q
    find some facts r1, r2, r3, …, rn in KB
      such that Unify(p1 ∧ p2 ∧ p3 ∧ … ∧ pn, r1 ∧ r2 ∧ r3 ∧ … ∧ rn) = θ
      and θ ≠ fail
    s = Subst(θ, q)
    if s can not unify with anything in KB or inferences:
      answer = Unify(s, Question)
      if answer ≠ Fail:
        return answer
      add s to inferences
  if inferences = { }:
    return False
  add inferences to KB
```

- Implied inner inner loop for the "find some …"
- Very inefficient indeed
- Can be improved a bit:

  Find a fact r_1 that can be unified with p_1 to give θ
  Then find a fact r_2 that unifies with p_2 in a way consistent with θ
  Continue doing that until all the p_i's are covered

Cheapest First Heuristic

- Find a carpenter whose father is a senator
- carpenter(X) $\wedge$ father(Y, X) $\wedge$ senator(Y)

Predicate	Number of Bindings
carpenter(X)	100,000
senator(X)	100
father(Y, X)	100,000,000
father(Y, someconstant)	2.4
father(someconstant, X)	1

We can also define how numbers work:

- number(z)
  z, which stands for zero here, is a number
- $\forall$ X, number(X) $\Rightarrow$ number(s(X))
  here, s stands for "successor": one after X
- $\forall$ X, add(z, X, Y) $\Leftrightarrow$ X = Y
  if it is true that adding z and X gives Y then X and Y are the same
- Zermelo-Fraenkel-Peano axioms
- $\forall$ X, add(s(A), X, s(Y)) $\Leftrightarrow$ add(A, X, Y)
- and so on. We can do something like this to define everything
  Principia Mathematica, Russell and Whitehead
- Because numbers *can* be represented, they are usually just taken for granted. We allow ourselves to write 7 instead of s(s(s(s(s(s(s(z))))))) and to use +, ×, and all the rest

Syntactic Sugar

- adds nothing real to the language
- but makes things much easier to read
- [a, b, c] can be used to represent a list containing a, b, and c
- but we don't *need* it
- lists can also be defined, linked-list style, without adding anything
- cons(a, cons(b, cons(c, nil)))
- many other examples, most of them are completely obvious

More on Unification

- A kind of pattern matching, in a way
- To unify two terms, you find values for all of their variables that make those terms equal
- Unifying cat(tom) with cat(X) tells us that X = tom
- Unifying add(s(A), X, s(Y)) with add(s(s(s(s(z)))), s(z), s(s(s(s(s(z))))))
  tells us that A is s(s(s(z))), X is s(z), and Y is s(s(s(s(z))))
- Unifying cat(Tom) with mortal(Tom) fails
- We unify terms that are somehow *supposed* to be equal
  To find out if they really can be equal (it doesn't fail)
  And what is required to make them equal:
  An Instantiation of their variables
- Instantiations can still have variables in them:

Unifying p(s(A)) with p(s(B)) tells us only that A = B
Unifying p(s(A)) with p(A) tells us that A = s(A), it *should* fail
    The Occurs rule