- Every function symbol has an associated <u>arity</u> indicating the number of elements or terms in the domain mapped onto each element of the range.

F(x,y)	arity 2
Father (david)	arity 1(unary).

- The term may be constant, variable, or function expression.

Symbols and Terms

- 1- True and False (Truth symbols)
- 2- Constant symbols
- 3- Variable symbols
- 4- function symbols

- An atomic sentences in predicate calculus is a predicate of arity n followed by n terms enclosed in parenthesis and separated by commas. Predicate calculus sentences are delimited by a period.
- E.g.
  – Like (ahmed , ali).
  – Likes(ali , kamil).
  – .
  – .
  – Friend(bill, goerge).

- Atomic sentences are also called atomic expressions or propositions. We may combine atomic sentences using logical operator to form sentences.
- V , ^ , ¬, ==> and =
- Predicate calculus includes two symbols, the variable quantifier ∀ and ∃.
- ∃Y friends (Y, peter) . (existential quantifier)
- ∀X Likes (X,ice_cream). (Universal quantifier).

Predicate calculus sentences

- Every atomic sentence is a sentence

1- if S is a sentence then ¬S

2- if S1 and S2 are sentences then S1^S2

3- if S1 and S2 are sentences then S1 V S2

4- if S1 and S2 are sentences then S1==>S2

5- if S1 and S2 are sentences then S1=S2

6-if X is a variable and S a sentence then ∀X S is a sentence.

7-if X is a variable and S a sentence then ∃X S is a sentence.

- mother (fatema,ali).
- mother (fatema,moh).
- father (ahmed, ali).
- father (ahmed ,moh).
- $\forall X \; \forall Y$ father(X,Y) V mother (X,Y) ==> parent(X,Y).

- $\forall X \; \forall Y \; \forall Z$ parent(X,Y) ^ parent (X,Z) ==> sibling(Y,Z).

Semantics for the predicate calculus

- Predicate calculus semantics provide a formal basis for determining the truth value of well-formed expressions.

- Examples of English sentences represented in predicate calculus are :

- If it doesn’t rain tomorrow, tom will go to the mountains.

  $\neg$weather (rain , tomorrow) ==> go (tom , mountain).

- All basketball players are tall

  $\forall$X basketball-player(X) ===> tall(X).

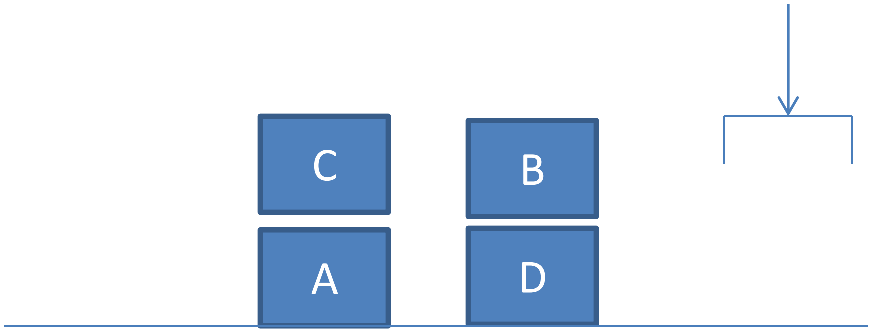
- Some people like fish

  $\exists$ X person(X) ^likes (X,fish).

- Nobody like Somali

  $\neg\exists$X likes (X, somali).

- on(c,a).
- on(b,d).
- ontable(a).
- ontable(d).
- clear(b).
- clear(c).
- hand-empty.



- ∀X(¬∃Y on(Y,X) ====> clear(X)).

- ∀X∀Y (hand-empty^ clear(X) ^clear(Y)^ pickup(X) ^ putdown(X,Y))==>stack(X,Y).

Unification

- Is an algorithm for determining the substitutions needed to make two predicate calculus expressions match.
- Suppose sentences p and q if pϛ=qϛ

p	q	ϛ
knows(john , x)	knows(john , Jane)	{ x| Jane}
knows(John , x)	knows(y , Oj)	{x|Oj, y|John}
knows(John , x)	knows(y , mother(y))	{y|John, x|mother(y)}

- Ex. foo(X, a, g(Y))

foo(fred, a, g(z))	**{ fred|X, z|Y}**
foo(w , a, g(jack))	{w|X , jack|Y}
foo(z, a, g(moo(z)))	{z|X , moo(z)|Y}

Unification algorithm

- Basic idea: can replace variables by:
- other variables
- constants
- function expressions
- High level algorithm:
- Represent the expression as a list
- Process the list one by one
  - Determine a substitution (if necessary)
  - Apply to the rest of the list before proceeding

Examples with the algorithm

- Unify p(a,X) and p(a,b)
- (p a X) (p a b)
- Unify p(a,X) and p(Y, f(Y))
- (p a X) (p Y (f Y))
- Unify parents(X, father(X), mother(bill)) and parents(bill, father(bill), Y)
- (parents X (father X) (mother bill))
- (parents bill (father bill) Y)

function unify code

```
function unify(E1, E2);
   begin
      case
         both E1 and E2 are constants or the empty list:                    %recursion stops
            if E1 = E2 then return {}
               else return FAIL;
         E1 is a variable:
            if E1 occurs in E2 then return FAIL
               else return {E2/E1};
         E2 is a variable:
            if E2 occurs in E1 then return FAIL
               else return {E1/E2}
         either E1 or E2 are empty then return FAIL         %the lists are of different sizes
         otherwise:                                                  %both E1 and E2 are lists
            begin
                  HE1 := first element of E1;
                  HE2 := first element of E2;
                  SUBS1 := unify(HE1,HE2);
                  if SUBS1 : = FAIL then return FAIL;
                  TE1 := apply(SUBS1, rest of E1);
                  TE2 : = apply (SUBS1, rest of E2);
                  SUBS2 : = unify(TE1, TE2);
                  if SUBS2 = FAIL then return FAIL;
                        else return composition(SUBS1,SUBS2)
            end
   end                                                                          %end case
end
```

Processed example

- (parents X (father X) (mother bill)), (parents bill (father bill) Y)
- parents =? Parents yes
- return nil
- (X (father X) (mother bill)), (bill (father bill) Y)
- X =? bill no, substitute
- return {bill/X}
- (bill (father bill) (mother bill)), (bill (father bill) Y)
- bill =? bill yes
- return nil

Processed example (cont'd)

- ((father bill) (mother bill)), ((father bill) Y)
- bill =? bill yes
- return nil
- (father bill), (father bill)
- father =? father yes
- return nil
- (bill) (bill)
- bill =? bill yes
- return nil

Processed example (cont'd)

- (mother bill), Y
-     (mother bill) =? Y   no, substitute
-     return {(mother bill) / Y}
- The set of unifying substitutions for
-   (parents X (father X) (mother bill)), (parents bill (father bill) Y)
- is
-   {bill / X, (mother bill) / Y}.
- The result is
-   (parents bill (father bill) (mother bill))

Inference rules

- The ability to infer new correct expressions from a set of true assertions.
- An interpretation that makes a sentence true is said to satisfy that sentence.
- An expression X logically follows from a set of predicate calculus expressions (S) if every interpretation that satisfies S also satisfies X This notion gives us a basis for verifying the correctness of rules of inference.
- If the inference rule is able to produce every expression that logically follows from S, then it is said to be complete , modus ponens.

- Ex p1=faster (bob , pat)

  P2=faster (pat, steve)

  p1^ p2 ===>q

  = faster(X, Y) ^ faster (Y,Z) ==> faster(X,Z).

  {bob|X, pat|Y, steve |Z}

  q=faster(bob,steve)

Satisfy, model, valid, inconsistent

- For a predicate calculus expression X and an interpretation I:
- If X has a value of T under I and a particular variable assignment, then I is said to *satisfy* X.
- If I satisfies X for all variable assignments, then I is a *model* of X.
- X is *satisfiable* iff there is an interpretation and variable assignment that satisfy it; otherwise it is *unsatisfiable*.

Satisfy, model, valid, inconsistent (cont'd)

- A set of expressions is *satisfiable* iff there is an interpretation and variable assignment that satisfy every element.
- If a set of expressions is not satisfiable, it is said to be *inconsistent*.
- If X has a value T for all possible interpretations, X is said to be *valid*.

Logically follows, sound, and complete

- A predicate calculus expression X *logically follows* from a set S of predicate calculus expressions if every interpretation and variable assignment that satisfies S also satisfies X.
- An inference rule is *sound* if every predicate calculus expression produced by the rule from a set S of predicate calculus expressions also logically follows from S.
- An inference rule is *complete* if, given a set S of predicate calculus expressions, the rule can infer every expression that logically follows from S.

Modus ponens and modus tollens

- If the sentences P and P $\rightarrow$ Q are known to be true, then *modus ponens* lets us infer Q.

- If the sentence P $\rightarrow$ Q is known to be true, and the sentence Q is known to be false, *modus tollens* lets us infer $\neg$P.

Example

- (S1) If John paid the electricity bill today, then he would have looked miserable when you saw him.
- (S2) John did not look miserable when you saw him.
- (S3) So John did not pay the electricity bill today.

Modus Tollens

- The form of that argument...
  - If X, then Y
  - not-Y
  - So not-X

Example

- (S1) If John paid the electricity bill today, then we do not have enough money to pay the gas bill.
- (S2) If we do not have enough money to pay the gas bill, Lisa will be angry.
- (S3) So if John paid the electricity bill today, then Lisa will be angry.

Summary

- Propositional calculus: no variables or functions
- Predicate calculus: allows quantified variables as parameters of predicates or functions
- Higher order logics: allows predicates to be variables (might be needed to describe mathematical properties such as “every proposition implies itself” or “there are decidable propositions.)