Predicate Logic

- *Terms* represent specific objects in the *world* and can be constants, variables or functions.
- *Predicate Symbols* refer to a particular relation among objects.
- *Sentences* represent facts, and are made of of *terms*, *quantifiers* and *predicate symbols*.

Predicate Logic

- *Functions* allow us to refer to objects indirectly (via some relationship).
- *Quantifiers* and *variables* allow us to refer to a collection of objects without explicitly naming each object.

Some Examples

- Predicates: Brother, Sister, Mother , Father
- Objects: Bill, Hillary, Chelsea, Roger
- Facts expressed as atomic sentences a.k.a. literals:

Father(Bill,Chelsea)

Mother(Hillary,Chelsea)

Brother(Bill,Roger)

∅ Father(Bill,Chelsea)

Variables and Universal Quantification

- Universal Quantification allows us to make a statement about a collection of objects:

"For All"

" *x* Cat(*x*) Þ Mammel(*x*)

All cats are mammels

" *x* Father(Bill,*x*) Þ Mother(Hillary,*x*)

All of Bill's kids are also Hillary's kids.

Variables and Existential Quantification

- Existential Quantification allows us to state that an object does exist (without naming it):

$$\$x \ \text{Cat}(x) \ \text{Ù} \ \text{Mean}(x)$$

There is a mean cat.

"There exists"

$$\$ x \ \text{Father}(\text{Bill},x) \ \text{Ù} \ \text{Mother}(\text{Hillary},x)$$

There is a kid whose father is Bill and whose mother is Hillary

Nested Quantification

" *x,y* Parent(*x,y*) **Þ** Child(*y,x*)

" *x* **$** *y* Loves(*x,y*)

" *x* [Passtest(*x*) **Ú** (**$** *x* ShootDave(*x*))]

Functions

- Functions are terms - they refer to a specific object.
- We can use functions to symbolically refer to objects without naming them.
- Examples:

  fatherof(x)   *age(x)*   *times(x,y)*   *succ(x)*

Using functions

" *x* Equal(*x*,*x*)

Equal(*factorial*(0),1)

" *x* Equal(*factorial(s(x))*,
times(s(x),factorial(x)))

Representing facts with Predicate Logic - Example

- Marcus was a man
- Marcus was a Pompeian
- All Pompeians were Romans
- Caesar was a ruler.
- All Romans were either loyal to Caesar or hated him.
- Everyone is loyal to someone.
- Men only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar

Predicate Logic Knowledgebase

Man(Marcus)

Pompeian(Marcus)

" *x* Pompeian(*x*) Þ Roman(*x*)

Ruler(Caesar)

" *x* Romans*(x)* Þ Loyalto(*x*,Caesar) Ú Hate(*x*,Caesar)

" *x* $*y* Loyalto(*x*,*y*)
" *x* " *y* Man(*x*) Ù Ruler(*y*) Ù Tryassassinate(*x*,*y*) Þ
Ø Loyalto(*x*,*y*)

Tryassassinate(Marcus,Caesar)

Questions (Goals)

Was Marcus a Roman?

Was Marcus loyal to Caesar?

Who was Marcus loyal to?

Was Marcus a ruler?

Will the test be easy?

Isa and Instance relationships

- The example uses inheritance without explicitly having *isa* or *instance* predicates.
- We could rewrite the facts using isa and instance explicitly:

instance(Marcus,man)

instance(Marcus,Pompeian)

isa(Pompeian,Roman)

Quiz

Using the predicates:

Father(x,y) Mother(x,y) Brother(x,y) Sister(x,y)

Construct predicate logic facts that establish the following relationships:

- GrandParent
- GrandFather
- GrandMother
- Uncle
- Cousin

Proof procedure for Predicate Logic

- Same idea, but a few added complexities:
  - conversion to CNF is much more complex.
  - Matching of literals requires providing a matching of variables, constants and/or functions.

Ø Skates(x) **Ú** LikesHockey(x)

Ø LikesHockey(y)

We can resolve these only if we assume x and y refer to the same object.

Predicate Logic and CNF

- Converting to CNF is harder - we need to worry about variables and quantifiers.
  1. Eliminate all implications Þ
  2. Reduce the scope of all Ø to single term. *
  3. Make all variable names unique
  4. Move Quantifiers Left *
  5. Eliminate Existential Quantifiers *
  6. Eliminate Universal Quantifiers *
  7. Convert to conjunction of disjuncts
  8. Create separate clause for each conjunct.

Eliminate Existential Quantifiers

- Any variable that is existentially quantified means we are saying there is some value for that variable that makes the expression true.
- To eliminate the quantifier, we can replace the variable with a function.
- We don’t know what the function is, we just know it exists.

Skolem functions

$*y* President*(y)*

We replace y with a new function func:

President*(func())*

func is called a skolem function.

In general the function must have the same number of arguments as the number of universal quantifiers in the current scope.

Skolemization Example

" *x* $*y* Father*(y,x)*

create a new function named foo and replace y with the function.

" *x* Father*(foo(x),x)*

Predicate Logic Resolution

- We have to worry about the arguments to predicates, so it is harder to know when 2 literals match and can be used by resolution.
- For example, does the literal Father(Bill,Chelsea) match Father(x,y) ?
- The answer depends on how we substitute values for variables.

Unification

- The process of finding a substitution for predicate parameters is called *unification*.
- We need to know:
  - that 2 literals can be matched.
  - the substitution is that makes the literals identical.
- There is a simple algorithm called the *unification algorithm* that does this.

The Unification Algorithm

1. Initial predicate symbols must match.
2. For each pair of predicate arguments:
   - different constants cannot match.
   - a variable may be replaced by a constant.
   - a variable may be replaced by another variable.
   - a variable may be replaced by a function as long as the function does not contain an instance of the variable.

Unification Algorithm

- When attempting to match 2 literals, all substitutions must be made to the entire literal.
- There may be many substitutions that unify 2 literals, the *most general unifier* is always desired.

Unification Example

"substitute x for y"

- P(x) and P(y): substitution = (x/y)

y for x, then z for y

- P(x,x) and P(y,z): (z/y)(y/x)
- P(x,f(y)) and P(Joe,z): (Joe/x, f(y)/z)
- P(f(x)) and P(x): can't do it!
- P(x) **Ú** Q(Jane) and P(Bill) **Ú** Q(y):

  (Bill/x, Jane/y)

Unification & Resolution Examples

Father(Bill,Chelsea) Ø Father(Bill,x)**Ú**Mother(Hillary,x)

Man(Marcus) Ø Man(x) **Ú** Mortal(x)

Loves(*father*(a),a) Ø Loves(x,y) **Ú** Loves(y,x)

This is a function

Predicate Logic Resolution Algorithm

- While no empty clause exists and there are clauses that can be resolved:
  - select 2 clauses that can be resolved.
  - resolve the clauses (after unification), apply the unification substitution to the result and store in the knowledge base.

Example:

Ø Smart(x) Ú Ø LikesHockey(x) Ú RPI(x)

Ø Canadian(y) Ú LikesHockey(y)

Ø Skates(z) Ú LikesHockey(z)

Smart(Joe)

Skates(Joe)

Goal is to find out if RPI(Joe) is true.

- Man(Marcus)
- Pompeian(Marcus)
- ∅ Pompeian(x_1) **Ú** Roman(x_1)
- Ruler(Caesar)
- ∅ Romans(x_2) **Ú** Loyalto(x_2,Caesar) **Ú** Hate(x_2,Caesar)
- Loyalto(x_3 , $f(x_3)$)
- ∅ Man(x_4) **Ú** ∅ Ruler(y_1) **Ú** ∅ Tryassassinate(x_4,y_1) **Ú** Loyalto(x_4,y_1)

- PROVE: Tryassassinate(Marcus,Caesar)

Answering Questions

- We can also use the proof procedure to answer questions such as “who tried to assassinate Caesar” by proving:
  - Tryassassinate(y,Caesar).
  - Once the proof is complete we need to find out what was substitution was made for y.

Computation

s(x) is the integer successor function

Equal(y,y)

Equal($factorial(s(x)),times(s(x),factorial(x))$)

…assume $s(\_)$ and $times(\_,\_)$ can compute.

We can ask for 10!:

Equal($factorial(10),z$)

Test Type Question

- The members of a bridge club are Joe, Sally, Bill and Ellen.
- Joe is married to Sally.
- Bill is Ellen’s Brother.
- The spouse of every married person in the club is also in the club.
- The last meeting of the club was at Joe’s house
- Was the last meeting at Sally’s house?
- Is Ellen married?

Logic Programming - Prolog

- Prolog is a declarative programming language based on logic.
- A Prolog program is a list of facts.
- There are various predicates and functions supplied to support I/O, graphics, etc.
- Instead of CNF, prolog uses an implicative normal form: A **Ù** B **Ù** ... **Ù** C **Þ D**

Prolog Example - Towers of Hanoi

```
hanoi(N) :- move(N,left,middle,right).

move(1,A,_,C) :- inform(A,C),!.

move(N,A,B,C) :-
  N1=N-1, move(N1,A,C,B),
  inform(A,C), move(N1,B,A,C).

inform(Loc1,Loc2) :-
   write(“Move disk from”,Loc1,” to”, Loc2).


hanoi(3)
```