

## First Order Logic and Prolog

### Introduction to Artificial Intelligence

Henry Kautz  
Spring 2007

## First Order Logic

Sentence ::= Atom  
| (Sentence Connective Sentence)  
|  $\forall$  Variable . Sentence  
|  $\neg \exists$  Variable . Sentence  
|  $\neg$  Sentence

Atom ::= Proposition  
| Predicate(Term, ...)

Term ::= Constant  
| Variable  
| Function(Term, ...)

## First-Order Logic

### All men are mortal.

$\forall x . (\text{man}(x) \supset \text{mortal}(x))$

### No man is not mortal.

$\neg \exists x . (\text{man}(x) \wedge \neg \text{mortal}(x))$

### Everybody has somebody they lean on.

$\forall x . (\text{person}(x) \supset \exists y . (\text{person}(y) \wedge \text{leans\_on}(x,y)))$

### A number is less than it's successor.

$\forall n . (\text{number}(n) \supset \text{less\_than}(n, \text{successor}(n)))$

### Nothing is less than zero.

$\neg \exists x . \text{less\_than}(x, \text{ZERO})$

## First-Order Clausal Form

- Begin with universal quantifiers (implicit)
- Rest is a clause
- No  $\exists$ , use function symbols instead
- Variables in each clause are unique to that clause
  - "x" in clause 1 is not the same "x" as in clause 2

$\forall x . (\text{man}(x) \supset \text{mortal}(x))$

$\text{man}(x) \vee \neg \text{mortal}(x)$

## Skolem Functions

- Begin with universal quantifiers (implicit)
- Rest is a clause
- No  $\exists$ , use function symbols instead
- Variables in each clause are unique to that clause
  - "x" in clause 1 is not the same "x" as in clause 2

$\forall x . (\text{person}(x) \supset \exists y . (\text{person}(y) \wedge \text{leans\_on}(x,y)))$

$\forall x . \exists y . (\text{person}(x) \supset (\text{person}(y) \wedge \text{leans\_on}(x,y)))$

$\forall x . (\text{person}(x) \supset (\text{person}(f(x)) \wedge \text{leans\_on}(x,f(x))))$

$\forall x . \neg \text{person}(x) \vee (\text{person}(f(x)) \wedge \text{leans\_on}(x,f(x)))$

$\forall x . (\neg \text{person}(x) \vee \text{person}(f(x))) \wedge (\neg \text{person}(x) \vee \text{leans\_on}(x,f(x)))$

$\forall x . \neg \text{person}(x) \vee \text{person}(f(x))$

$\forall x . \neg \text{person}(x) \vee \text{leans\_on}(x,f(x))$

## Unification

Can resolve clauses if can unify one pair of literals

- Same predicate, one positive, one negative
- Match variable(s) to other variables, constants, or complex terms (function symbols)
- Carry bindings on variables through to all the other literals in the result

$(\text{Mortal}(\text{HENRY}))$        $(\neg \text{Mortal}(y) \vee \text{Fallible}(y))$   
  
 $(\text{Fallible}(\text{HENRY}))$

### Unification with Multiple Variables

You always hurt the ones you love.  
 Politicians love themselves.  
 Therefore, politicians hurt themselves.

$$\begin{array}{c}
 \neg \text{love}(x,y) \vee \text{hurt}(x,y) \quad \neg \text{politician}(z) \vee \text{love}(z,z) \\
 \swarrow \quad \searrow \\
 \neg \text{politician}(z) \vee \text{hurt}(z,z)
 \end{array}$$

### Unification with Multiple Variables

You always hurt the ones you love.  
 Politicians love themselves.  
 Therefore, politicians hurt themselves.

$$\begin{array}{c}
 \neg \text{love}(x,y) \vee \text{hurt}(x,y) \quad \neg \text{politician}(z) \vee \text{love}(z,z) \\
 \swarrow \quad \searrow \\
 \neg \text{politician}(w) \vee \text{hurt}(w,w)
 \end{array}$$

rename "z" as "w" so that no clauses have variables with the same name

### Why Keep Distinct Variables in Each Clause?

If you hit someone, then you hurt them.  
 If you hurt someone, then you are bad.  
 Therefore: If you hit someone, then you are bad.

$$\begin{array}{c}
 \neg \text{hit}(x,y) \vee \text{hurt}(x,y) \quad \neg \text{hurt}(y,x) \vee \text{bad}(y) \\
 \swarrow \quad \searrow \\
 \neg \text{hit}(x,x) \vee \text{bad}(x)
 \end{array}$$

if you hit yourself then you are bad?!

### Why Keep Distinct Variables in Each Clause?

If you hit someone, then you hurt them.  
 If you hurt someone, then you are bad.  
 Therefore: If you hit someone, then you are bad.

$$\begin{array}{c}
 \neg \text{hit}(x,y) \vee \text{hurt}(x,y) \quad \neg \text{hurt}(w,z) \vee \text{bad}(w) \\
 \swarrow \quad \searrow \\
 \neg \text{hit}(x,z) \vee \text{bad}(x)
 \end{array}$$

### Unification with Function Symbols

A number is less than its successor  
 "Less than" is transitive  
 A number is less than the successor of its successor

$$\begin{array}{c}
 (\text{Less}(a, \text{suc}(a))) \quad (\neg \text{Less}(b,c) \vee \neg \text{Less}(c,d) \vee \text{Less}(b,d)) \\
 \swarrow \quad \searrow \\
 \{c/a, d/\text{suc}(a)\} \\
 (\neg \text{Less}(b,a) \vee \text{Less}(b, \text{suc}(a))) \\
 \text{rename variables:} \\
 \{e/a, f/\text{suc}(a)\} \\
 (\neg \text{Less}(e,f) \vee \text{Less}(e, \text{suc}(f))) \\
 \swarrow \quad \searrow \\
 \text{Less}(a, \text{suc}(\text{suc}(a)))
 \end{array}$$

### Making FOL Practical

Barriers to using FOL:

- Choice of clauses to resolve
- Huge amount of memory to store DAG
- Getting useful answers to queries (not just "yes" or "no")

PROLOG's answers:

- Simple backward-chaining resolution strategy – left/right, first to last clause
- Tree-shaped proofs – no need to store entire proof in memory at one time
- Extract answers to queries by returning variable bindings

## happy.pl

```
happy(X) :- rich(X).
happy(X) :- loves(X,Y),happy(Y).
loves(X,Y) :- spouse(X,Y).
loves(X,Y) :- mother(X,Y).
```

QUERIES:

```
?- happy(bill).           YES
?- happy(henry).         NO
?- happy(Z).
```

rich(bill).  
spouse(melinda,bill).  
mother(elaine,melinda).  
mother(mary,bill).  
rich(paul).  
mother(barbara,henry).

## Prolog Interpreter

```
binding_list disprove(literal neglit){
  choose (clause c) such that
    (binding = unify(head(c),neglit))
  if (no choice possible){
    backtrack to last choice;}
  for (each lit in body(c)){
    binding = binding U
      disprove(substitute(lit,binding));
  }
  return binding;
}
```

## Exercise: The Mini Zebra Puzzle

There are three houses in a row on street. Each house is inhabited by a man of a different nationality, who has a different pet, and drinks a different beverage.

The Spaniard own a dog.

The Ukranian drinks tea.

The man in the third house drinks milk.

The Norwegian lives next to the tea drinker.

The juice drinker owns a fox.

The fox is next door to the dog.

Question: Who owns the zebra?

## Prolog Limitations

- Only handles definite clauses (exactly one positive literal per clause)
  - Cannot express e.g.  
happy(bill) v happy(henry)
- Tree-shaped proofs means some sub-steps may be repeatedly derived
  - DATALOG: does forward-chaining inference and caches derived unit clauses
- Interpreter can get into an infinite loop if care is not taken in form & order of clauses