
Ch 7.1, 9.1-9.6,9.8

Predicates are more expressive than propositions.

Predicates express relationships between 0 or more objects.

The price: more expensive inference and a weaker form of completeness, formally it is semidecidable.

Notation:
SUBST(x/Sam,y/Pam,Likes(x,y))=Likes(Sam,Pam).

- Examples:
  - Rather than representing "Socrates is mortal" as the proposition "socrates-mortal", we write "Mortal(Socrates)".
  - Married(Bill,Jane) would assert that Bill is married to Jane. This allows us to write generally true statements such as

    $$Married(x, y) \rightarrow Married(y, x)$$

    , which was unexpressible in propositional logic.
  - Note: You are free to choose whatever predicates you want.
- The use of **complex objects**, which appear as functions of zero or more arguments, e.g
  Table(Legs(4),Height(28in)).

- The ability to use **variables**. For example, Rare(x) → Expensive(x) would assert that if something is rare, then it is expensive. (Note: Prolog reverses the logicians' standard practice of using lower case for variables). A variable can stand for any object but not for a predicate.
- An ability to **quantify** over variables. There are two quantifiers:
  - **universal quantification**, meaning "for all x". e.g., "All men are mortal" is written as:
    $$\forall\ x\ man(x) \rightarrow mortal(x)$$
  - **existential quantification**, meaning "there exists an x". e.g., "Someone is sad" is written as:
    $$\exists\ x\ sad(x)$$

1 Inference Rules

Universal Elimnation ∀ x Likes(x,IceCream) via substitution {x/Ben} we infer Like(Ben,IceCream).

Existential Elimination (Skolemization From ∃ x Kill(x,Victim) infer Kill(Murderer,Victim).

Existential Introduction From Likes(Jerry,IceCream) infer ∃ x Likes(x,IceCream)

With these additions, we now have first order predicate calculus. It is also called first order logic (FOL).

The obvious advantage is that we can say a lot more. One disadvantage is that while theorem proving is still **sound**, (that is, we can always prove true theorems), it is now **undecidable** (the theorem prover may never halt on untrue statements).

2 Using first order logic

First order logic has been used in modelling planning, expertise, natural language processing, etc.

Answering questions (e.g., Is Socrates mortal?) or solving problems (e.g., solving the Towers of Hanoi puzzle) now consists of two problems:

- **Axiomatization** of our knowledge of the problem (state description and legal operators) in FOL
- Application of some **proof technique** to show that a desired solution follows deductively from the axioms and rules of inference.

Unfortunately, neither of these is a simple task. This is because of the following problems:

- **Representation:** Axiomatization of knowledge is an art rather than a science, i.e. we have no algorithm.
- **Processing** Although **proof techniques** can be mechanized, the search space or computation may be enormous.

3 A simple example: Using Predicate Logic

Given:

Spot is a dog.
A dog is an animal.
Animals have hearts.

Conclude:

Does Spot have a heart?

First, form axioms.

1) Dog(Spot)
2) $\forall$ x Dog(x) $\rightarrow$ Animal(x)
3) $\forall$ x Animal(x) $\rightarrow$ Hasa(x,Heart)

Now try to prove:

Hasa(Spot,Heart)

By combining 1 and 2 we can conclude:

4) Animal(Spot)

and by combining 3 and 4 we can conclude:

Hasa(Spot,Heart)

4 Translating English into FOL

Consider the following set of English assertions and their translation into FOL. Is any information lost?
Note: No procedure/algorithm exists for mapping English or natural problems into formal logic.

- Marcus was a man
  Man(Marcus).
- Marcus was a Pompeian
  Pompeian(Marcus).
- Marcus was born in 40 A.D
  Born(Marcus,40)
- All men are mortal
  $\forall x \;\; Man(x) \Rightarrow Mortal(x)$
- All Pompeians died when the volcano erupted in 79 A.D.
  $Erupted(Volcano, 79) \wedge \forall x \;\; Pompeian(x)$
  $\rightarrow Died(x, 79)$
- No mortal lives longer than 150 years.
  $\forall x \; \forall t1 \; \forall t2 \; Mortal(x) \wedge Born(x, t1) \wedge$
  $GT(t2 - t1, 150) \rightarrow Dead(x, t2)$
- It is now 1985
  Now = 1985.
- Alive means not dead
  $\forall x \; \forall t \;\; Alive(x, t) \leftrightarrow\sim Dead(x, t)$
- If someone dies, then he is dead at all later times.
  $\forall x \; \forall t1 \; \forall t2 \;\; Died(x, t1) \wedge GT(t2, t1) \rightarrow Dead(x, t2)$

5 Problems with Axiomatization

Given an axiomatization of some aspects of longevity and Pompeians, we can now ask a question and hope to find an answer:

- Is Marcus alive?
  $Alive(Marcus, Now)$

This translation has not been an easy task.

In general, we would like a one-to-one mapping from English to FOL sentences.

We see, however, that we have been forced into somewhat *ad hoc* representational commitments for:

- verb tense
- the meaning of predicates
- representation of time
- representation of numerical quantities
- treatment of causality
- disambiguation of referents
- inclusion of implicit information
- exclusion of extraneous information

6 Making life simpler: Horn clauses and Resolution

Logical proofs can be very complicated.

Horn clauses are a canonical that will simplify the inference task.

- representing all expressions is **clause** or Horn form
- using the **resolution** inference process

Briefly, what this entails is converting all expressions into a form that uses only OR and NOT (clausal form). Using this representation, only a single form of inference (resolution) is used.

There is no loss in representation or inference power, although whatever natural reasoning validity there was is sacrificed.

Horn clause logic can be efficiently implemented and programs with adequate performance can be written quickly. Prolog is an approximation to Horn clause logic.

7 Conversion from FOL to Clausal Form

No one knows these!.

Given a list of FOL axioms, an equivalent list can be produced which uses only the connectives OR and NOT. It is then in clausal form.

There is a largely mechanical conversion from FOL to clausal form. It can be implemented as a nine step process. The idea is to keep breaking things down into simpler forms. This process can be automated. Most find it easier to simply represent the information in Horn form to start with.

1. Eliminate implication.
2. Reduce the scope of negation. (move not inward)
3. Standardize variables with respect to quantifiers.
4. Move quantifiers to the left.
5. Skolemize (remove existential quantifiers)
6. Drop universal quantifiers ($\forall$ is now implicit).
7. Distribute $\wedge$ over $\vee$
8. Treat each conjunct as a clause.
9. Standardize apart variables across clauses.

8 Conversion from FOL to Clausal Form

Lets look at simple examples of each step.

1. Eliminate $\rightarrow$ by using the fact that $A \rightarrow B$ is equal to $\sim A \vee B$.

$$A \rightarrow (B \rightarrow C)$$

$$\sim A \vee (\sim B \vee C)$$

2. Reduce the scope of $\sim$ by using:
   $\sim (\sim A)$ equals A
   $\sim (A \vee B) = \sim A \wedge \sim B$
   $\sim (A \wedge B) = \sim A \vee \sim B$
   $\sim \forall x \ P(x) = \exists x \sim P(x)$
   $\sim \exists x \ P(x) = \forall x \sim P(x)$
   $\sim (\forall x \ P(x) \vee \exists y \ Q(y))$
   $\exists x \sim P(x) \wedge \forall y \sim Q(y)$

3. Standardize variables so that each quantifier has a unique variable.

$$\forall x \ P(x) \vee \forall x \ Q(x)$$

$$\forall x \ P(x) \vee \forall y \ Q(y)$$

4. Move all quantifiers to the far left without changing their relative order.
   $\forall x \;\; P(x) \vee \exists \;\; Q(y)$
   $\forall x \exists y \;\; P(x) \vee Q(y)$

5. Eliminate existential quantifiers. If $\exists x$, then we can give it a particular name say X1. If this occurs within the scope of some $\forall y$ then make X1 a function of y. This is called a **Skolem** function. A Skolem function with no arguments is a **Skolem constant**.
   $\forall a \forall b \exists c \;\; P(a, c) \vee Q(b, c)$
   $\forall a \forall b \;\; P(a, C1(a, b)) \vee Q(b, C1(a, b))$

6. Drop the universal quantification symbol. In clausal form it is assumed that all variables are universally quantifed.
   $\forall x \forall y \;\; P(x) \vee Q(y)$
   $P(x) \vee Q(y)$

7. Convert to conjunctive normal form (AND of ORs). This can be done by continued use of the distributive rules.
   $(A \wedge B) \vee (C \wedge D)$
   $(A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)$

8. Split into separate **clauses**.
   $(W \vee X) \wedge (Y \vee Z)$
   $(W \vee X)$
   $(Y \vee Z)$

9. Standardize variables apart. That is, rename variables in each clause so that they are all different.

9 Resolution: Propositional Form

Once sentences are in clausal form, we can apply the **resolution** inference process. One rule is enough.

Given two clauses, this process is:

- Find two complementary terms (e.g., A and $\sim A$) in the two clauses.
- cancel them
- form a new clause containing all the remaining terms.

e.g. resolving
$X \vee Y \vee Z$
$\sim X \vee A$

$Y \vee Z \vee A$
which is a valid deduction.

10 Loss of directionality

For another example, resolving

$$\sim X \vee Y$$

and

$$X$$

results in

$$Y$$

Notice that this is equivalent to the **modus ponens** inference rule.

However, by using clausal form and resolution, we have lost any intended **directionality** of inference. For example,

$$(A \wedge B \wedge C) \rightarrow D$$

seems to represent one possible inference, while

$$\sim A \vee \sim B \vee \sim C \vee D$$

may be resolved on any of it literals.

11 Resolution: Predicate Form

Resolution can also be applied to predicates, providing that the variable bindings are handled appropriately, i.e. by unification.

Robinson (1965) showed that there is no loss of inferential power using only the resolution inference process.

Given two clauses, the process is:

- find two (potentially) complementary terms (e.g., $P(x)$ and $\sim P(a)$)
- find variable bindings that make then (exactly) complementary, i.e **unify** terms.
- apply those bindings throughout the clauses
- cancel the complementary terms
- form a new clause containing all the remaining terms.

12 Resolution Proof

For example, from

$$Rare(x) \rightarrow Expensive(x)$$
$$\sim Rare(x) \vee Expensive(x)$$
$$Rare(Oldstamps)$$

What can we conclude.

We need to substitute Oldstamps for x to yield

$$\sim Rare(Oldstamps) \vee Expensive(Oldstamps)$$

$$Rare(Oldstamps)$$

which can be resolved to yield

$$Expensive(Oldstamps)$$

13 Unification

We need a systematic way of finding substitutions for variables so that parts of clauses become identical and can be resolved away. The process of finding variable bindings to make expressions identical is called **unification**.

Two terms **unify** if there is a substitution for the variables which makes the terms identical. Unification is the process of finding the substitutions. The process is rather straightforward. Remember, however, that:

- Predicates and functions must match or the clauses can't be unified.
- You can't substitute for anything except variables.
- If you substitute for a variable, you must substitute for it everywhere in the clause.
- Constants, variables or functions can be substituted for variables, but not predicates (that's second order logic).
- A variable can't be replaced by a function containing the same variable.

14 Examples of unification

Here we use the PROLOG convention of variables in upper case.

You can run Prolog to test your knowledge.

$$p(X)$$
$$p(a)$$

are unified by a/X

$$p(X)$$
$$p(f(Y))$$

are unified by f(Y)/X

$$p(X, X)$$
$$p(Y, Z)$$

are unified by X/Y, Y/Z

$$p(X)$$
$$q(Y)$$

can't be unified

$$p(X, X)$$
$$p(f(a), Y))$$

are unified by f(a)/X, f(a)/Y

$$p(X, X)$$
$$p(f(a), a))$$

can't be unified

15 Resolution refutation

Now that we have a standard representation scheme and a standard inference process, lets work on a standard proof strategy.

The most obvious approach is to try and derive the goal clause. That works, but we are constantly comparing clauses against the goal.

This can be avoided by using the **refutation** process. This consist of two steps, and are the steps that PROLOG uses:

1. add the negation of the goal clause to the set of clauses
2. try and derive a contradiction

Since the original clauses are assumed to be consistent, any contradiction must be due to the negated goal clause. If assuming the goal clause to be **false** leads to a contradiction, then the goal clause itself must be **true**.

16 Refutation Proof

As before, suppose we know:

1. $\sim$ Rare(x) $\vee$ Expensive(x)
2. Rare(Oldstamps)

The same statements, expressed in Prolog.
expensive(X):-rare(X).
rare(oldstamps).

We now ask the question what's expensive, via
$Expensive(x)$.

We add 3 $\sim$ $expensive(X)$ to our domain or proper axioms and resolve away, binding variables as needed.

- resolving 1 and 3 yields $\sim rare(X)$
- resolving this and 2 yields $\emptyset$ with X=oldstamps.

This contradiction demonstrates that oldstamps are expensive. This is exactly how PROLOG works.

17 Example Proof, with conversion to Normal form

Conversion will to standard Prolog. This took about 10 minutes.

> The law says that is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

```
 it is a crime to sell weapons
       to hostile nations

criminal(X):- american(X),weaon(Y),nation(Z),
             hostile(Z),sells(X,Z,Y,).

Nono .. has some missiles

   owns(nono,xo).
    missile(xo).

All of its missiles were sold to it
        by Colonel West.

  sells(west,non,X):-owns(nono,X),missile(X).

Background knowledge: (often the hard part)

    weapon(X):-missile(X).
    hostile(X):-enemy(x,america).

The country Nono...

    country(nono).

Nono, an enemy of America

    enemy(nono,america).
```

```
nation(america).
```

18 Prolog Proof

```
    :-consult[example287].
 ?- trace,criminal(west).

   (1) 0 Call: criminal(west) ?
   (1) 1 Head [1]: criminal(west) ?
   (2) 1 Call: american(west) ?
   (2) 2 Head [1]: american(west) ?
   (2) 1 Done: american(west) ?
   (3) 1 Call: weapon(_6858) ?
   (3) 2 Head [1]: weapon(_6858) ?
   (4) 2 Call: missile(_6858) ?
   (4) 3 Head [1]: missile(_6858) ?
   (4) 2 Done: missile(x0) ?
   (3) 1 Done: weapon(x0) ?
   (5) 1 Call: nation(_6862) ?
   (5) 2 Head [1->2]: nation(_6862) ?
   (5) 1 Exit: nation(nono) ?
   (6) 1 Call: hostile(nono) ?
   (6) 2 Head [1]: hostile(nono) ?
   (7) 2 Call: enemy(nono,america) ?
   (7) 3 Head [1]: enemy(nono,america) ?
   (7) 2 Done: enemy(nono,america) ?
   (6) 1 Done: hostile(nono) ?
   (8) 1 Call: sells(west,nono,x0) ?
   (8) 2 Head [1]: sells(west,nono,x0) ?
   (9) 2 Call: owns(nono,x0) ?
   (9) 3 Head [1]: owns(nono,x0) ?
   (9) 2 Done: owns(nono,x0) ?
   (10) 2 Call: missile(x0) ?
   (10) 3 Head [1]: missile(x0) ?
   (10) 2 Done: missile(x0) ?
   (8) 1 Done: sells(west,nono,x0) ?
   (1) 0 Exit: criminal(west) ?

yes
```

19 Generating proofs in FOL

In comparison to **axiomatization** of the task domain, a computational mechanism for generating proofs is nearly trivial, at least to do poorly.

One method is simply **generate-and-test**.

- Given a set of axioms, the theorem to be proven, and inference rules of FOL: **generate** a statement by applying a rule of inference to an axiom.
- **Test** the resulting deduction (another sentence in FOL) to see if it is equivalent to the theorem to be proven.

This is a **forward chaining** approach to theorem proving. This is usually applied if you are given an additional fact and you want to see what you can conclude, e.g. if the alarm sounded, what would you concluded.

One may also **backward chaining** from the statement of the theorem to be proven, constructing what amounts to an AND/OR tree of subgoals and alternative proof plans. (This will be covered in the section on search). This is how Prolog works. Backward chaining is usually applied when you have a goal and a fixed set of facts, such as occurs when generated a diagnosis.

Of course, either sort of proof technique requires some mechanism for finding appropriate variable bindings in order to propagate substitutions throughout sentences in the proof.

20 Direction of search

Except for the problem of producing the original axioms, we now have a completely mechanized approach to theorem proving. However, we have left one step underspecified. In what order do we pick clauses for resolution?

Since this can be viewed a search problem, we can choose a number of strategies for selecting parent clauses.

- forward chaining: work from the "data" axioms toward the "goal" theorem.
- backward chaining: work backward from the goal by always selecting at least one of its decedents for each resolution. This is called the **set of support** strategy and is what PROLOG does.
- no particular direction: just throw the negated goal clause in with the rest and don't distinguish between them.

Backward chaining is generally superior in that it focuses attention on the task at hand and thus produces fewer irrelevant inferences.

Resolution has no preferred direction of inference, and so can be used with any search strategy.

21 Method of search

Beyond the direction of search, there are still a number of options available. You will study these search methods in the next part of the course.

- A **breadth-first strategy** is guaranteed to find a contradiction if one exists, but may store an exponential number of inferred clauses.
- A **depth-first strategy** may not find a contradiction if one exists. If failed paths are retracted, it requires much less memory, but at the expense of possibly deriving the same clause multiple times.
- **Iterative deepening** permits you the advantages and disadvantages of both depth-first and breadth-first.
- With the **unit-preference** strategy we always prefer selecting parents which are single literal clauses.
- **Best first**? An obvious strategy if there was some way of coming up with good heuristics.

22 Other improvements

For efficiency purposes, we could **index** clauses by the predicates and functions they contain, also indicating whether it is negated or not. Then, having selected one clause, we can immediatly locate all other clauses which which it can be resolved.

In addition, we can improve things a little by dropping:

1. **tautologies**: That is, clauses that are always true (e.g., $A \vee \sim A$)
2. clauses that are **subsumed** by other clauses: That is, which are less general (e.g., $A \vee B$ is subsumed by A)

A failure to find necessary complementary literals or the inability to generate novel resolvents tells us that we will not find a contradiction. If this occures, we can quit.

Keeping track of bindings allows us to **answer questions** after having found a proof. Abstractly, this is the basis of the programming language, PROLOG.

23 Theorems about of Predicate Logic

Godel 1930 The theorems in first order logic are exactly all the valid sentences, i.e. FOL is sound and complete.

Godel 1930 Any logic that includes arithmetic (the natural numbers with addition) is incomplete, i.e. there are sentences w such that neither w nor $\sim w$ can be proven. Incompleteness of Arithmetic.

Tarski 1935 There are logics in which valid statements cannot be proven.

Church 1936 FOL is undecidable, i.e. you cannot write an algorithm which will separate theorems from non-theorems. Completeness is a theoretical property, but does not give you an algorithm for deciding.

Robinson 1965 Resolution is a procedure that will find a proof for a true statements, but you do not know how long it will take, i.e. first order logic is semidecidable. Also consistency is semidecidable.

24 Problems with FOL representations

Using FOL for knowledge representation gives us a fairly uniform language and a range of proof techniques that fit well within the search framework we've developed.

However, there are some serious problems.

Axiomatization/Representatability of non-trivial task domains may be a major undertaking. Consider, for example, an axiomatization of common-sense knowledge, e.g. warm air rises.

Efficiency Cast as a search problem, construction of a proof (a series of deductions) suffers from a high branching factor. Many rules of inference may apply at any point in the proof.

Decidability We cannot be assured that our proof mechanism will ever halt, i.e. FOL is fundamentally undeciable.

Comprehensibility Logical representations and the problem solving techniques attendant to them strongly violate some people's senses of psychological plausibility.

25 Limitations of FOL

Theoretical Incompleteness FOL is incomplete. PROLOG assumes that if it can't prove a statement it is false, and that is not a valid argument. This problem doesn't occur in practice.

Computational Cost There is no limit on the amount of computation time that deriving a conclusion may take.

Inference Selection Given a set of facts, there is no mechanism for deciding which inferences are best.

Representational Incompleteness These problems occur all the time.

Probablistic No way to say 40% of the people voted for Clinton.

Uncertainty Can't deal with quantifiers "most or some", e.g. most people get married.

Generality Can't say that a relation is transitive.

Logic is the best understood scheme for representing and using knowledge.

26 Extensions to Predicate Calculus

- Second Order Logic allows quantification over predicates. For example, $\forall$ R is transitive if $\forall$ x,y,z R(x,y),R(y,z)$\rightarrow$ R(x,z).
- Arithmetic Logics provide axioms to write down the integers. Gödel showed that *any* such logic will be incomplete, i.e. there are true statements that are unprovable.
- Include **functions** which return objects. e.g., father(x).
- Include **computable functions** which return objects or truth values by direct computation. For arithmetic computations, this is almost unavoidable. For example "greaterthan(4+2,5)" must both compute 4+2 and compare integers.
- Define **equivalence** of objects. By saying A=B we assert that A and B are interchangeable in all predicates and functions.

When we start adding computable functions and other goodies to predicate logic, we may lose **completeness**.

The relationship between predicate calculus/logic and "meaning" is still a subject of some debate.