Logic and Automatic Theorem Proving

Overview

- Knowledge representation
- Knowledge bases
- Logic and Frames
- Propositional Logic
- Inference rules
- Normal forms

Knowledge Representation

There are basically two classes of representations in traditional AI:

- **Logic**: methods based on first-order predicate calculus (mathematical logic)
- **Frames**: methods based on networks of nodes representing objects or concepts, and labeled arcs representing relations among nodes

These two are competitive, but complementary.

Representation Hypothesis

A central tenet in AI is the **representation hypothesis**, which states that intelligent behavior is based on

- **representation** of input and output data as symbols
- **reasoning** by processing symbol structures, resulting in new symbol structures

The problem then is what the representations and reasoning process should be.

Strengths and Weaknesses of Logic and Frames

Logic (predicate calculus)

- Strengths: (1) logical power, (2) rigorous mathematical foundation
- Weaknesses: (1) slow (search) (2) rigidity (T/F)

Frames

- Strengths: (1) supports defaults (data is seldom complete), (2) procedural attachment(*)
- Weaknesses: weak logical power

(*) Procedural attachment: pullers (if needed), pusher (if added), if referenced, if deleted, if changed, etc.

Alternatives to the Representation Hypothesis

There are several alternatives:

- analog information: continuous values
- special-purpose hardware: domain specific functions such as vision
- neural networks: subsymbolic approachs
- holographic memories
- etc.

Knowledge Base Systems

Domain-independent algorithms: **Inference engine**
Domain-specific content: **Knowledge base**

- KB: set of sentences in a formal language
- KB is **declarative**: tell what we want, not how we want it done (i.e. procedural)

KB Constructs

- Knowledge base: how to represent knowledge with sentences or formulas $\longrightarrow$ various forms of logic
- Inference engine: how to generate new sentences or formulas given old ones in the KB $\longrightarrow$ various forms of inference procedures

Logic: Language for KBs

Logic is the representational language for KBs:

- logic consists of **syntax** (sentence structure) and **semantics** (how sentences relate to the real world; T/F values)
- **interpretation**: fact to which a sentence refers (T/F assignment)
- **inference**: deriving new sentences from old ones

Inference procedure

- **sound**: no false sentences can be derived from the KB using the inference procedure
- **complete**: inference procedure can derive **all** true conclusions from a set of premises

Types of Logic

Ontological: what exists in the world?

Epistemological: what can we know?

Language	Ontological	Epistemological
Propositional Logic	facts	T/F/?
First-order Logic	facts, objects, relations	T/F/?
Temporal Logic	facts, objects, relations, times	T/F/?
Probability Theory	facts	degree of belief 0..1
Fuzzy Logic	degree of truth	degree of belief 0..1

* first-order logic == predicate calculus

Let's begin with propositional logic.

Well-Formed Formulas in Propositional Logic

Components of well-formed formulas (sentences):

- propositional symbols (atoms): **P**, **Q**, **R**
- parentheses: **( )**
- connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
- constants: **True, False**

Well-Formed Formulas (Cont'd)

Well-Formed Formulas (wff): Syntax

$$wff \quad \Rightarrow \quad atom|constant$$

$$wff \quad \Rightarrow \quad (\neg wff)$$

$$\begin{aligned} wff \quad \Rightarrow \quad & (wff \vee wff) \\ & | \, (wff \wedge wff) \\ & | \, (wff \rightarrow wff) \\ & | \, (wff \leftrightarrow wff) \end{aligned}$$

Operator precedence: $\neg$; $\wedge, \vee$; $\rightarrow, \leftrightarrow$ (decreasing order)

Propositional Logic: Semantics

- atoms can take on **True** or **False** values.
- an interpretation assigns specific truth values to the atoms.
- for a formula with n atoms, there are 2^n possible truth assignments.
- a formula is true **under an interpretation** iff the formula evaluates to **True** with the assignment of truth values within the interpretation.
- a formula is **valid** iff it is **True** under all interpretations.
- a formula is **inconsistent (unsatisfiable)** iff it is **False** under all interpretations.
- a formula G is **valid** if $\neg G$ is **inconsistent**

Propositional Logic: Semantics (cont'd)

- if a formula F is **True** under an interpretation I, then we say I satisfies F. We also say I is a **model** for F
- if formula F is **False** under interpretation I, then we say I falsifies F
- two formulas F and G are equivalent iff F and G have the same truth values under every interpretation I:
$$F \leftrightarrow G$$
- there can be many models (at least one) of a formula F if F is satisfiable.

Basic Laws of Propositional Logic

- $F \vee G = G \vee F$,
$F \wedge G = G \wedge F$ (commutative)
- $(F \vee G) \vee H = F \vee (G \vee H)$,
$(F \wedge G) \wedge H = F \wedge (G \wedge H)$,(associative)
- $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$,
$F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$ (distributive)
- $F \vee \mathbf{False} = F, F \wedge \mathbf{False} = \mathbf{False}$
- $F \vee \mathbf{True} = \mathbf{True}$
$F \wedge \mathbf{True} = F$
- $F \vee \neg F = \mathbf{True}$
$F \wedge \neg F = \mathbf{False}$

Basic Formulas (cont'd)

- $\neg(\neg F) = F$
- $\neg(F \vee G) = \neg F \wedge \neg G$
$\neg(F \wedge G) = \neg F \vee \neg G$ (De Morgan's Law)
- $F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F)$
- $F \rightarrow G = \neg F \vee G$
- $F \wedge F = F$
$F \vee F = F$

Inference Rules

- Modus Ponens:

$$\frac{F \rightarrow G, F}{G}$$

- Unit Resolution:

$$\frac{F \vee G, \neg G}{F}$$

- Resolution:

$$\frac{F \vee G, \neg G \vee H}{F \vee H} \; or\; equivalently \; \frac{\neg F \rightarrow G, G \rightarrow H}{\neg F \rightarrow H}$$

Normal Forms (I)

- **literals**: $atom|\neg atom$
- **clauses**: disjunction of 1 or more **literals**

$$literal \vee literal \vee ...$$

- **terms**: conjunction of 1 or more **literals**

$$literal \wedge literal \wedge ...$$

Normal Forms (II)

- **Conjunctive Normal Form**: conjunction of **clauses**

$$C1 \wedge C2 \wedge C3...$$

$$e.g.\ (\neg F \vee G \vee H) \wedge (\neg G) \wedge (K \vee L)$$

- **Disjunctive Normal Form**: disjunction of **terms**

$$T1 \vee T2 \vee T3...$$

$$e.g.\ (\neg F \wedge G \wedge H) \vee (\neg G) \vee (K \wedge L)$$

Key Points

- Knowledge representation: logic and frames, pros and cons
- Knowledge bases: the basic components
- Propositional Logic: basic laws
- Inference rules: what is inference, basic inference rules
- Normal forms: definitions

Overview

* Don't confuse formula F with false constant $\mathbf{False}$ **(in bold)**.

- Horn clauses
- Theorem proving
- Resolution in propositional logic

Horn Clauses

Horn clauses

- clauses that contain ≤ 1 **positive** literal:

$$F \vee \neg G \vee \neg H, \neg F \vee G$$

Horn Normal Form: conjunction of horn clauses

- for example, $(F \vee \neg G \vee \neg H) \wedge (\neg F \vee G)$
- it is the same as: $((G \wedge H) \rightarrow F) \wedge (F \rightarrow G)$
- Easier to do inference (computationally less intensive) than other normal forms.
- Restrictive, so not all formulas can be represented in horn normal form.

Horn Clauses

Basically, horn clauses are implications where the conclusion part consist of only a single literal. Thus, these two are equivalent.

$$F \vee \neg G \vee \neg H$$

$$(G \wedge H) \rightarrow F$$

As you will see later, this form is suitable for theorem proving with backward chaining.

Converting to Normal Forms

You can transform any formula into a normal form by applying the following rules:

1. Use the laws:
   $F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F)$
   $F \rightarrow G = \neg F \vee G$
   to eliminate $\rightarrow$ and $\leftrightarrow$
2. Repeatedly use the law:
   $\neg(\neg F) = F$
   and the De Morgan's laws:
   $\neg(F \vee G) = \neg F \wedge \neg G$
   $\neg(F \wedge G) = \neg F \vee \neg G$
   to bring negation signs immediately before atoms.
3. Repeatedly use the distributive laws:
   $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$,
   $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$
   and the other laws as necessary.

Exercise: Converting to Normal Forms

Convert the following into CNF and DNF:

$$(P \vee \neg Q) \rightarrow R$$

- DNF :

$$\begin{aligned}(P \vee \neg Q) \rightarrow R &= \neg(P \vee \neg Q) \vee R \text{ :remove connective} \\ &= (\neg P \wedge \neg(\neg Q)) \vee R \text{ :by De Morgan's Law} \\ &= (\neg P \wedge Q) \vee R \text{ :remove double negation}\end{aligned}$$

- CNF : try it yourself (hint: use the distributive law)

Another exercise: find the CNF of $(P \wedge (Q \rightarrow R)) \rightarrow S)$

Logical Consequence

G is a **logical consequence** of formulas $F_1, F_2, ..., F_n$ iff for any interpretation I for which $F_1 \wedge F_2 \wedge ... \wedge F_n$ is true, G is also true (i.e. $(F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G$ is **valid**).

c.f. Modus Ponens $\dfrac{F, F \rightarrow G}{G}$

- $(F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G$ is called a **theorem**.
- $F_1, F_2, ..., F_n$ are called **axioms (postulates, premises)** of G
- G is called the **conclusion**.

Valid vs. Inconsistent

Theorem: G is a logical consequence of $F_1, F_2, ..., F_n$ iff the formula $F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \neg G$ is **inconsistent**

Proof: G is a logical consequence of $F_1, F_2, ..., F_n$ iff $(F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G$ (lets call this H) is valid. Since H is valid iff $\neg H$ is inconsistent, H is valid iff $\neg((F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G)$ is inconsistent. Because

$$\begin{aligned}&\neg((F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G) \\ = \;& \neg(\neg(F_1 \wedge F_2 \wedge ... \wedge F_n) \vee G) \\ = \;& (\neg(\neg(F_1 \wedge F_2 \wedge ... \wedge F_n))) \wedge \neg G \\ = \;& (F_1 \wedge F_2 \wedge ... \wedge F_n) \wedge \neg G,\end{aligned}$$

H is valid iff $F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \neg G$ is **inconsistent**.

Caveats

Anything is a logical consequence of $\mathbf{False}$ (recall that G is a logical consequence of F iff $F \rightarrow G$ is valid).

$$\begin{aligned}&\mathbf{False} \rightarrow G \\ = \;& \neg\mathbf{False} \vee G \\ = \;& \mathbf{True} \vee G \\ = \;& \mathbf{True}\end{aligned} \tag{1}$$

Thus, for a result of a theorem to be meaningful, the premises should be **consistent**.

Logical Consequence: Proving

Model checking (truth table, search), or algebraic application of inference rules:

1. Truth table: the conclusion G must be true whenever the premises $F_1 \wedge F_2 \wedge ... \wedge F_n$ is true.
2. Prove that $(F_1 \wedge F_2 \wedge ... \wedge F_n) \rightarrow G$ is **valid**:
   - truth table, or
   - algebraically reduce the formula to $\mathbf{True}$
3. Prove that $F_1 \wedge F_2 \wedge ... \wedge F_n \wedge \neg G$ is **inconsistent**:
   - truth table, or
   - algebraically reduce the formula to $\mathbf{False}$

Theorem Proving

Given a set of facts (*ground literals*) and a set of rules, a desired theorem can be proved in several different ways:

- **Forward Chaining**: use known facts and rules to discover (or deduce) new facts. When the desired theorem is deduced, stop.
- **Backward Chaining**: work backward from the theorem by finding rules that could deduce it; then try to deduce the premises of those rules.
- **Resolution**: proof by contradiction. Using ground facts, rules, and the **negation** of the theorem, try to derive $\mathbf{False}$ by resolution steps. To prove $F \rightarrow G$ is valid, prove $F \wedge \neg G$ (i.e. $\neg(F \rightarrow G)$) is inconsistent.

Theorem Proving: Example

Given that the following are all $\mathbf{True}$,

$$\begin{array}{rl} A & (1) \\ B & (2) \\ D & (3) \\ A \wedge B \rightarrow C & (4) \\ C \wedge D \rightarrow E & (5) \end{array}$$

Prove that E is valid. Let's consider forward chaining and backward chaining.

Forward Chaining

Given that the following are all $\mathbf{True}$,

$$\begin{array}{rl} A & (1) \\ B & (2) \\ D & (3) \\ A \wedge B \rightarrow C & (4) \\ C \wedge D \rightarrow E & (5) \end{array}$$

Since we know that A and B are true (1 and 2), C is also true (from 4). Let's call this (6), i.e. $C = \mathbf{True}$. From this, and the fact that D is true, we come to the conclusion that E is true (3, 5, and 6).

Backward Chaining

Given that the following are all $\mathbf{True}$,

$$A \quad (1)$$
$$B \quad (2)$$
$$D \quad (3)$$
$$A \wedge B \rightarrow C \quad (4)$$
$$C \wedge D \rightarrow E \quad (5)$$

Find a rule where E is deduced (5). For this rule to be true when E is true, C and D must be true. Since $D = \mathbf{True}$ is given (3), we only need to show that C is true. Find a rule where C is deduced (4), and repeat the same process until all premises are deduced.

*** this strategy is ideal for Horn Normal Form.**

Resolution: An Overview

Given formulas in conjunctive normal form $F = F_1 \wedge F_2 \wedge \ldots \wedge F_n$, where each F_i is a **clause (i.e. disjunctions of literals)**, and the desired conclusion G, to show G is a logical consequence of F, follow these steps:

1. negate G and add it to the list of clauses (make it into CNF if necessary):

   $$F_1 , F_2 , \ldots , F_n , \neg G$$

2. choose two clauses that have **exactly one** pair of literals that are complementary, e.g.:

   $$F_n : \underline{\neg P} \vee Q \vee R \quad \text{and} \quad F_m : S \vee \underline{P}$$

3. Produce a new clause by deleting the complimentary pair and producing a new formula, e.g.:

   $$Q \vee R \vee S$$

4. repeat until the new clause generated is **False**

This assumes that the premises are consistent.

Resolution: An Example

$$A \quad (1)$$
$$B \quad (2)$$
$$D \quad (3)$$
$$\neg A \vee \neg B \vee C \quad (4)$$
$$\neg C \vee \neg D \vee E \quad (5)$$

Given the above, we want to prove that E is true. We simply add the negation of the desired conclusion, and try to draw a contradiction:

$$\neg E \quad (6)$$

Resolution: Solution

Given	
A	(1)
B	(2)
D	(3)
$\neg A \vee \neg B \vee C$	(4)
$\neg C \vee \neg D \vee E$	(5)
$\neg E$	(6)

$\Rightarrow$

Resolution		
(1,4)	$\neg B \vee C$	(7)
(2,7)	C	(8)
(5,6)	$\neg C \vee \neg D$	(9)
(8,9)	$\neg D$	(10)
(3,10)	$\mathbf{False}$	(11)

Resolution: Why Does It Work

The goal of resolution is to show that G is a logical consequence of $F_1 \wedge ... \wedge F_n$ is valid. This is equivalent to showing that $F_1 \wedge ... \wedge F_n \wedge \neg G$ is **inconsistent**.

Note that if H is a logical consequence of $F_1 \wedge ... \wedge F_n$, then $F_1 \wedge ... \wedge F_n = F_1 \wedge ... \wedge F_n \wedge H$:

When $F_1 \wedge ... \wedge F_n$ is

1. $\mathbf{True}$: then H must also be true.
2. $\mathbf{False}$: both sides are false, thus H does not matter.

Thus, we can add any logical consequence of $F_1 \wedge ... \wedge F_n$ or of **any subset** of the F_i's without changing the value of the result. Recall that we **added** newly derived formulas to the list in the previous slide.

What Resolution Is Not

If $C_1 \wedge C_2 \rightarrow H$

- **then** $C_1 \wedge C_2 \wedge H = C_1 \wedge C_2$
- **but not** $C_1 \wedge C_2 = H$

In other words,

$$(C_1 \wedge C_2 \rightarrow H) \rightarrow ((C_1 \wedge C_2 \wedge H) \leftrightarrow (C_1 \wedge C_2))$$

$$(C_1 \wedge C_2 \rightarrow H) \nrightarrow ((C_1 \wedge C_2) \leftrightarrow H)$$

Exercise: Verify the above with $C_1 = (A \vee B), C_2 = (\neg B \vee C)$, and $H = (A \vee C)$.

Resolution Algorithm

1. Convert premises $F_1 \wedge ... \wedge F_n$ into **CNF**, and make a list of resulting clauses.
2. Negate the conclusion, convert to **CNF**, and add to the clause list.
3. **Resolution Step**: pick two clauses from the list with **exactly one** complementary literal; any other literals if they appear on both clauses **must** have the same sign. Form a new clause by disjunction w/o the complementary literals, and **add to the list**.

$$\underbrace{(P \vee C_i), (\neg P \vee C_j)}_{F_i \text{ and } F_j} \Rightarrow \underbrace{(C_i \vee C_j)}_{\text{Add to list}}$$

4. If $\mathbf{False}$ was added to the list of clauses, in step 3, stop; theorem proved. Otherwise, go to step 3.

Limitation of Propositional Logic

Limited expressive power:

- P: All men are mortal
- Q: Socrates is a man
- R: Socrates is mortal

Can you prove $(P \wedge Q) \rightarrow R$ using propositional logic?

Exercise 6.6, p. 181

Given:

If the unicorn is mythical, then it is immortal($M \rightarrow I$), but if it is not mythical, then it is a mortal mammal ($\neg M \rightarrow (\neg I \wedge L)$). If the unicorn is either immortal or a mammal, then it is horned ($(I \vee L) \rightarrow H$). The unicorn is magical if it is horned ($H \rightarrow G$).

Prove or disprove:

1. The unicorn is mythical (M).
2. The unicorn is magical (G).
3. The unicorn is horned (H).$\longleftarrow$ Let's prove this.

Exercise: Solution Using Resolution

1. $\neg M \vee I$
2. (a) $M \vee \neg I$, (b) $M \vee L$
3. (a) $\neg I \vee H$, (b) $\neg L \vee H$
4. $\neg H \vee G$
5. $\neg H$ (negated conclusion)

3a,5	$\neg I$	(6)
3b,5	$\neg L$	(7)
2b,7	M	(8)
1,6	$\neg M$	(9)
8,9	$\mathbf{False}$	(10)

Is The Unicorn Horned?

Given:

1. $M \rightarrow I$
2. $\neg M \rightarrow (\neg I \wedge L)$
3. $(I \vee L) \rightarrow H$
4. $H \rightarrow G$

Prove: H

1,	$\neg I \rightarrow \neg M$	(5)
2 and 5,	$\neg I \rightarrow \neg M \rightarrow (\neg I \wedge L)$	(6)
6,	$I \vee (\neg I \wedge L)$	(7)
7,	$(I \vee \neg I) \wedge (I \vee L) = \mathbf{True} \wedge (I \vee L) = (I \vee L)$	(8)
8 and 3,	$(I \vee L) \rightarrow H$	(9)

Limitation of Propositional Logic

Limited expressive power:

- P: All men are mortal
- Q: Socrates is a man
- R: Socrates is mortal

Can you prove $(P \wedge Q) \rightarrow R$ using propositional logic?

Key Points

- Normal forms: definitions, know how to convert, applying basic laws and inference rules
- Theorem proving: basic approaches. forward and backward chaining concept, and resolution.
- know how to do resolution in propositional logic
- limitation of propositional logic

Predicate Calculus (First-Order Logic)

Treatment of predicate calculus in this lecture closely follows: Chang, C.-L., and Lee, R. C.-T., "Symbolic Logic and Mechanical Theorem Proving", Academic Press, London, 1973. Substitution notation follows Russell & Norvig.

Predicate Calculus (First-Order Logic)

Propositional logic does not allow us to perform any reasoning based on the use of general rules, so its usefulness is limited. Predicate Calculus generalizes Propositional Calculus to allow the expression and use of **general rules**.

- objects
- relations
- properties
- functions: similar to relations but returns only one value

New Concepts Introduced in Predicate Calculus

- **terms** : objects in the domain, and how things get transformed (**functions**)
- **predicates** : properties of objects (certain properties are **True** or **False**)
- **quantifiers** : express properties of large set of objects without enumerating

Terms in Predicate Calculus

A **Term** is:

- constant: $a, b, c, ...$
- variable: $x, y, x, ...$
- $f(t_1, ..., t_n)$, where f is a function symbol and $t_1, t_2, ..., t_n$ are terms.

Terms refer to objects in a domain.

Predicate Calculus Constructs

- Variables: $x, y, z, ...$
- Constants: John, Mary, 3
- Functions: $f(x), g(y), h(z), father(\text{John}), ...$
  - maps **term(s)** to a **term**
- Predicates:
  $P(x, y), GREATER(x, 3), LOVE(father(\text{John}), \text{John})$
  - function whose value is **True** or **False**
- Quantifiers: $\forall$ (for all), $\exists$ (there exists)

Mortality Revisited

Propositional logic

- P: All men are mortal
- Q: Socrates is a man
- R: Socrates is mortal

First-order logic

- P: All men are mortal $\forall x MAN(x) \rightarrow MORTAL(x)$
- Q: Socrates is a man $MAN(\text{Socrates})$
- R: Socrates is mortal $MORTAL(\text{Socrates})$

A Formal Definition

Well Formed Formula (or Sentence):

WFF	= Atomic-Formula | WFF Connective WFF
	| Quantifier Variable, WFF | ¬WFF | (WFF)
Atomic-Formula	= Predicate(Term, ...) | Term = Term
Term	= Function(Term, ...) | Constant | Variable
Connective	= $\rightarrow$ | $\wedge$ | $\vee$ | $\leftrightarrow$
Quantifier	= $\forall$ | $\exists$
Constant	= A | X_1 | John | ...
Variable	= a | x | s | ...
Predicate	= Before | HasColor | Raining | ...
Function	= Mother | LocationOf | ...

Functions vs. Predicates

- Functions: returns a single object (term); relations
  - $FatherOf(GeorgeJr) = GeorgeSr$,
  - $DistanceBetween(MilkyWay, Andromeda) =$ 2-million light years, ...
- Predicates: returns a truth value; properties
  - $IsFather(GeorgeSr, GeorgeJr) = \mathbf{True}$
  - $HeavierThan(Earth, Sun) = \mathbf{False}$

Must disambiguate: $Brother(x, y)$ could be
- $AreBrothers(x, y)$:predicate, or
- $BrotherOf(x, y)$:function, i.e. a common brother of x and y.

Quantifiers

$\forall$var wff

- Universal quantifier $\forall$:
  - Every $Skunk$ is $Stinky$: translates into $\forall x\ Skunk(x) \rightarrow Stinky(x)$
  - *note that the main connective is* $\rightarrow$

$\exists$var, wff

- Existential quantifier $\exists$:
  - There exists a Cat that is $White$: translates into $\exists x\ Cat(x) \wedge White(x)$
  - Same as: Some Cat is $White$
  - *note that the main connective is* $\wedge$

Common Mistakes With Quantifiers

All skunks are stinky:

- Correct: $\forall x\ Skunk(x) \rightarrow Stinky(x)$
- Wrong: $\forall x\ Skunk(x) \wedge Stinky(x)$
  - this means: everything is a skunk **and** it is stinky.

Some cats are white:

- Correct: $\exists x\ Cat(x) \wedge White(x)$
- Wrong: $\exists x\ Cat(x) \rightarrow White(x)$
  - this is true if there is something that is **not** a cat!

Properties of Quantifiers

- $\forall x\ \forall y = \forall y\ \forall x$
- $\exists x\ \exists y = \exists y\ \exists x$
- $\forall x\ \exists y \neq \exists y\ \forall x$
  $\exists x\ \forall y\ Loves(x, y)$ vs.
  $\forall y\ \exists x\ Loves(x, y)$
- quantifiers can be translated using each other:
  $\forall x\ Likes(x, Coffee)$ $\quad\quad$ $\neg\exists x\ \neg Likes(x, Coffee)$
  $\exists x\ Likes(x, Broccoli)$ $\quad\quad$ $\neg\forall x\ \neg Likes(x, Broccoli)$

Semantics of Predicate Calculus

Formulas are true with respect to a **model** and and **interpretation**.

Models contains **objects** and **relations**:

- objects: constants
- relations: predicates
- functional relations: functions

An atomic formula $Predicate(term_1, term_2, ..., term_n)$ is true iff the **objects** referred to by $term_1, term_2, ..., term_n$ are in the **relation** referred to by $Predicate$.

Example: Howling Hounds

1. All hounds howl at night.
2. Anyone who has any cats will not have any mice.
3. Light sleepers do not have anything which howls at night.
4. John has either a cat or a hound.
5. **Prove**: If John is a light sleeper, then John does not have any mice.

Example: Howling Hounds (cont'd)

1. $\forall x\,(HOUND(x) \rightarrow HOWL(x))$
2. $\forall x \forall y((HAVE(x,y) \wedge CAT(y)) \rightarrow$
   $\neg\exists z(HAVE(x,z) \wedge MOUSE(z)))$
3. $\forall x\,(LS(x) \rightarrow \neg\exists y(HAVE(x,y) \wedge HOWL(y)))$
4. $\exists x\,(HAVE(John,x) \wedge (CAT(x) \vee HOUND(x)))$
5. **Prove:**
   $LS(John) \rightarrow \neg\exists x\,(HAVE(John,x) \wedge Mouse(x))$

Canonical Forms of Predicate Calculus

1. Prenex Normal Form: arranged all quantifiers at the front of the formula : use De Morgan's rules (p. 193)
2. Convert the non-quantifier part (called the **matrix**) into Conjunctive Normal Form
3. Skolemization: eliminate existential quantifiers by introducing **Skolem constants** or **Skolem functions**.

Result:

$$\forall x_1 \forall x_2 ... \forall x_n (\mathbf{CNF})$$

Key Points

- predicate calculus basics
- quantifier properties, and common mistakes
- translating English into predicate calculus
- canonical forms for predicate calculus: basics

Overview

- Representing relations in predicate calculus
- Interpretation in predicate calculus
- prenex normal form
- skolemization
- inference

Domain

- A *Domain* is a section of the world about which we wish to express some knowledge.
- The totality of the objects in the part of that world consists the **domain**.
- Basically, the set of all **constants** (i.e. objects) makes up a domain: *John, Bill, Bob,* ...

Example: everyone on Earth, everyone in Texas, everyone in College Station, every computer in the Bright Bldg., etc.

Example: Kinship Domain

$$\forall m \forall c \;\; Mother(c) = m \leftrightarrow (Female(m) \wedge Parent(m, c))$$

$$\forall w \forall h \;\; Husband(h, w) \leftrightarrow Male(h) \wedge Spouse(h, w)$$

$$\forall p \forall c \;\; Parent(p, c) \leftrightarrow Child(c, p)$$

$$\forall g \forall c \; Grandparent(g, c) \leftrightarrow \exists p (Parent(g, p) \wedge Parent(p, c))$$

$$\forall x \forall y \; Sibling(x, y) \leftrightarrow (x \neq y \wedge \exists p (Parent(p, x) \wedge Parent(p, y))$$

Exercise: 7.6, p. 214 – try it out

Interpretation in Predicate Calculus

An **interpretation** of a formula F in first-order logic consists of a nonempty domain D, and an assignment of values to each constant, function, and predicate occurring in F as follows:

1. **constant**:
   assign an element of D (e.g. an integer)
2. **function** with n arguments:
   assign a mapping from D^n to D
3. **predicate** with n arguments:
   assign a mapping from D^n to $\{\mathbf{True}, \mathbf{False}\}$

$D^n = \{(x_1, ..., x_n) | x_i \in D$ for $i = 1, ..., n\}$ Similar to assigning truth values in propositional logic.

Evaluation of a Formula Given an Interpretation

1. if F and G have been evaluated, evaluate
   $\neg F, (F \wedge G), (F \vee G), (F \rightarrow G),$ and $(F \leftrightarrow G)$
2. $\forall x\, G$ is evaluated to $\mathbf{True}$ if G evaluates to $\mathbf{True}$ for *every* x in D; otherwise, G evaluates to $\mathbf{False}$
3. $\exists x\, G$ is evaluated to $\mathbf{True}$ if G evaluates to $\mathbf{True}$ for *any* x in D; otherwise, G evaluates to $\mathbf{False}$

Side Note: Bound vs. Free Variables

- **Scope** of a quantifier:
  the range (parentheses) over which the associated variable takes effect
- **Bound variable**:
  an occurrence of a variable in a formula is **bound** iff the occurrence is within the scope of a quantifier employing the variable.
- **Free variable**:
  an occurrence of a variable in a formula is **free** iff the occurrence is not bound.

Bound: $\forall x \forall y P(x, y)$; Free: $\forall x\, P(x, y)$;
Both Free and Bound: $(\forall x\, P(x, y)) \wedge (\forall y\, Q(y))$

Example: Interpretation and Evaluation

Given the **interpretation**:

- **Domain**: $D = \{Bob, Carol, Ted, Alice\}$
- **Predicates**: $Woman(Carol), Woman(Alice)$
  $Man(Bob), Man(Ted)$
  $Loves(Bob, Carol), Loves(Ted, Alice), Loves(Carol, Ted)$
- **Functions**:
  $Brother(Bob) = Ted, Boss(Alice) = Carol$

Evaluate:

$$\forall x\, (Man(x) \rightarrow \exists y (Woman(y) \wedge Loves(x, y)))$$

Consistency, Satisfiability, and Validity

A formula G is

- **consistent** (satisfiable) iff there exists an interpretation I such that G evaluates to $\mathbf{True}$ in I. In this case, I is a **model** of G and I satisfies G.
- **inconsistent** (unsatisfiable) iff there is no interpretation that satisfies G.
- **valid** iff every interpretation of G satisfies G.
- **invalid** iff there is at least one interpretation I of G such that G evaluates to $\mathbf{False}$ under I.

Difficulty: Many Domains Are Infinite

Algebra, etc.

- There are an infinite number of interpretations of a formula.
- In general, **none** of the properties in the previous slide are **decidable** in general for formulas in predicate calculus.

Standard Forms of Predicate Calculus

1. Prenex Normal Form: arranged all quantifiers at the front of the formula : use De Morgan's rules (p. 193)
2. Convert the non-quantifier part (called the **matrix**) into Conjunctive Normal Form
3. Skolemization: eliminate existential quantifiers by introducing **Skolem constants** or **Skolem functions**.

Result:

$$\forall x_1 \forall x_2 ... \forall x_n (\mathbf{CNF})$$

Prenex Normal Form

A formula F in first-order logic is in **Prenex Normal Form** iff the formula is in the form:

$$\underbrace{Q_1 x_1 \; Q_2 x_2 \; ... Q_n x_n}_{\text{Prefix}} \; \underbrace{(M)}_{\text{Matrix}}$$

where $\mathbf{Q_i}$ is $\forall$ or $\exists$, and M contains no quantifiers.

Conjunctive Normal Form

Analogous to propositional logic.

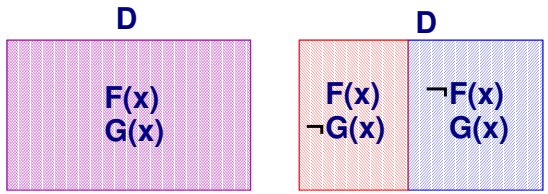
- a list of clauses (disjunction of literals)
- r-literal clause, unit clause, and empty clause.

Quantifier Equivalences: Converting to Prenex Normal Form

Equivalence formulas ($Q = \forall$ or $\exists$):

- $(Qx\ F(x)) \vee G = Qx\ (F(x) \vee G)$
  $(Qx\ F(x)) \wedge G = Qx\ (F(x) \wedge G)$
- $\neg(\forall x\ F(x)) = \exists x\ (\neg F(x))\ \neg(\exists x\ F(x)) = \forall x\ (\neg F(x))$
- $(\forall x\ F(x)) \wedge (\forall x\ G(x)) = \forall x\ (F(x) \wedge G(x))$
  $(\exists x\ F(x)) \vee (\exists x\ G(x)) = \exists x\ (F(x) \vee G(x))$
- $(Q_1 x\ F(x)) \vee (Q_2 x\ H(x)) = Q_1 x\ Q_2 z\ (F(x) \vee H(z))$
  $(Q_1 x\ F(x)) \wedge (Q_2 x\ H(x)) = Q_1 x\ Q_2 z\ (F(x) \wedge H(z))$

Quantifier Equivalences



Equivalence formulas ($Q = \forall$ or $\exists$):

- $\underbrace{(\forall x\ F(x))} \wedge \underbrace{(\forall x\ G(x))} = \forall x\ \underbrace{(F(x) \wedge G(x))}$
- $\underbrace{(\exists x\ F(x))} \vee \underbrace{(\exists x\ G(x))} = \exists x\ \underbrace{(F(x) \vee G(x))}$

What about $(\forall x\ F(x)) \vee (\forall x\ G(x))$ and $\forall x\ (F(x) \vee G(x))$?

Exercise: Conversion to Prenex Normal Form

Convert $(\forall x\ P(x)) \rightarrow (\exists y\ Q(y))$ to Prenex Normal Form:

$$\begin{aligned}(\forall x\ P(x)) \rightarrow (\exists y\ Q(y)) &= \neg(\forall x\ P(x)) \vee (\exists y\ Q(y)) \\ &= (\exists x\ \neg P(x)) \vee (\exists y\ Q(y)) \\ &= \exists x\ \exists y\ (\neg P(x) \vee Q(y))\end{aligned}$$

More exercise:

$\forall x\ \forall y\ ((\exists z\ P(x,z) \wedge P(y,z)) \rightarrow (\exists u\ Q(x,y,u)))$

Example Proof: Motivating Skolemization

Given:

1. No used-car dealer buys a used car for his family.

$$\forall x\ (U(x) \rightarrow \neg B(x))$$

2. Some people who buy used cars are absolutely dishonest.

$$\exists x\ (B(x) \wedge D(x))$$

3. **Prove**: Some absolutely dishonest people are not used-car dealers.

$$\exists x\ (D(x) \wedge \neg U(x))$$

Proof

$$\forall x\ (U(x) \rightarrow \neg B(x)) \quad (2)$$

$$\exists x\ (B(x) \wedge D(x)) \quad (3)$$

Conclusion: $\exists x\ (D(x) \wedge \neg U(x))$ (4)

1. Assume (1) and (2) are true in domain D under interpretation I. Because of (2), there must be an x in D, say "a", such that $B(\mathrm{a}) \wedge D(\mathrm{a})$ is $\mathbf{True}$ under I.
2. Thus, $B(\mathrm{a})$ is $\mathbf{T}$ and $\neg B(\mathrm{a})$ is $\mathbf{F}$.
3. (1) is $\forall x(\neg U(x) \vee \neg B(x))$. $\neg U(x)$ must be $\mathbf{T}$ because $\neg B(x)$ is $\mathbf{F}$.
4. Because of (2), $D(\mathrm{a})$ is also true, thus $D(\mathrm{a}) \wedge \neg U(\mathrm{a})$ is $\mathbf{T}$, and "a" is one example where (3) is $\mathbf{T}$ in domain D.

Skolemization

Eliminate existential quantifiers through replacement of bound variables with **constants** or **functions**.

- Assume the formula is in Prenex Normal Form ($Q = \forall$ or $\exists$):

$$F = Q_1 x_1 Q_2 x_2 .... Q_n x_n (M)$$

- In a general case, where $Q_r x_r$ is $\exists x_r$, delete $Q_r x_r$ and replace every x_r in F by $f(x_{s_1}, x_{s_2}, ..., x_{s_m})$, where $x_{s_1}, x_{s_2}, ..., x_{s_m}$ are all the universally quantified variables appearing to the left of x_r, and f is a new function symbol (**Skolem function**) not appearing in F.
- If there is no universal quantifier $\forall$ to the left of the existential quantifier $\exists$ in question, remove $\exists$ and replace the variable associated with it with a constant a (called a **Skolem constant**).
- Observation: constant is like a function with no arguments.

Skolemization Example

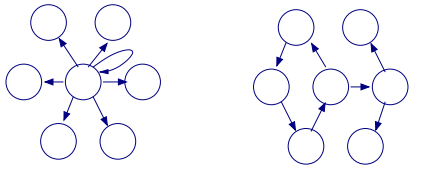
- **Initial formula:**

$$\exists x\ \forall y\ \forall z\ \exists u\ \forall v\ \exists w P(x, y, z, u, v, w)$$

- $\exists x \leftarrow a = h()$
- $\exists u \leftarrow f(y, z)$
- $\exists w \leftarrow g(y, z, v)$
- **Result of Skolemization:**

$$\forall y\ \forall z\ \forall v\ P(a, y, z, f(y, z), v, g(y, z, v))$$

Quantifier Order and Skolemization



Different quantifier order results in different Skolemization:

- $\exists x \, \forall y \; Loves(x, y)$
  - $\forall y \; Loves(a, y)$
- $\forall y \, \exists x \; Loves(x, y)$
  - $\forall y \; Loves(f(y), y)$

a is a new Skolem constant, and $f(\cdot)$ is a new Skolem function.

Standard Form: A Summary

We followed these three steps to convert first-order logic formulas into a standard form amenable to algorithmic verification:

1. Transform formula into **Prenex Normal Form**.
2. Transform the matrix into **Conjunctive Normal Form**.
3. Eliminate existential quantifiers through **Skolemization**.

$\Rightarrow$ A set of **clauses** in **CNF** in which all variables are **universally quantified**

Example Proof: Used-Car Revisited

Given:

$$\forall x \; (U(x) \rightarrow \neg B(x)) \qquad (1)$$
$$\exists x \; (B(x) \wedge D(x)) \qquad (2)$$
Conclusion: $\exists x \; (D(x) \wedge \neg U(x)) \qquad (3)$

Convert to standard form:

$$\neg U(x) \vee \neg B(x) \qquad (1)$$
$$B(a) \qquad (2a)$$
$$D(a) \qquad (2b)$$
$$\neg D(x) \vee U(x) \qquad (3)$$

Example Proof: Resolution Steps

Given the clauses:

$$\neg U(x) \vee \neg B(x) \qquad (1)$$
$$B(a) \qquad (2a)$$
$$D(a) \qquad (2b)$$
$$\neg D(x) \vee U(x) \qquad (3)$$

	1, 2a:	$\neg U(a)$	(4)
Resolution:	3,4:	$\neg D(a)$	(5)
	2b,5:	$\mathbf{False}$	(6)

Note: **unification** is used above, which will be discussed next time.

Note: Resolving

1. $\underbrace{\forall x\ (\neg U(x) \vee \neg B(x))}_{\text{clause 1}} \wedge \underbrace{B(a)}_{\text{clause 2}}$
   - because clause 1 is $\mathbf{T}$, and $B(a)$ is $\mathbf{T}$ (clause 2), $\neg U(a) \vee \neg B(a)$ must be $\mathbf{T}$.
   - from this and $B(a)$, we can derive $\neg U(a)$.
2. $\underbrace{\forall x\ \neg D(x)}_{\text{clause 5}} \wedge \underbrace{D(a)}_{\text{clause 2b}}$
   - it only takes **one** counter example (here, a) to refute the formula above.

Key Points

- Representing relations in predicate calculus: domains,
- Interpretation in predicate calculus: what is an interpretation and how it related to a domain. When is an interpretation true or false.
- prenex normal form: why it is useful, how to convert to, the basic rules used in conversion
- skolemization: why it is useful, how to do it
- inference: basics of resolution – first step is converting to a standard form.

Overview

- Substitution
- Unification algorithm
- Unification in LISP
- Factors
- Resolvents

Resolution for Predicate Calculus

The resolution step is valid for predicate calculus, when two clauses contain complementary predicates. For example, clause C_1 may contain predicate $P(\cdot)$ and clause C_2 may contain predicate $\neg P(\cdot)$.

$$C_1 : \quad P(x) \vee Q(x)$$
$$C_2 : \quad \neg P(f(x)) \vee R(x)$$

We could substitute $f(a)$ for x in C_1 and a for x in C_2, and then resolve to get

$$C_3 : Q(f(a)) \vee R(a)$$

More generally, we could substitute $f(x)$for x in C_1 and resolve to get

$$C_3 : Q(f(x)) \vee R(x)$$

Remaining Issues

Theorem proving steps:

1. Conversion of natural language sentences into first-order logic formulas
2. Conversion to standard form
3. Resolution

Remaining issue: how to substitute variables to resolve two clauses and generate a new clause $\Rightarrow$ do substitution and unification.

Ground Term

A term (constant, variable, or function of terms) is a **ground term** if no variable appears in the term.

- ground constant
- ground literal
- ground clause
- etc.

Substitution

- A **substitution** is a finite set of the form

  $$\{v_1/t_1, ..., v_n/t_n\}$$

  where each v_i is a variable, each t_i is a term (constant, variable, or function of terms), and no two v_i are identical.
- A substitution in which each t_i is a ground term is called **ground substitution**.
- The *empty substitution* $\epsilon = \{\}$ contains no elements.

Why is substitution important: assists in resolving two clauses by making the two clauses with different variables compatible.

Substitution Applied to a Formula

- Let $\theta = \{v_1/t_1, ..., v_n/t_n\}$ be a substitution and E be an expression. Then $E\theta$ is an expression obtained from E by replacing *simultaneously* each occurrence of variable v_i $(1 \leq i \leq n)$ in E by the term t_i.
- $E\theta$ is called an instance of E.

In the textbook, $E\theta$ is denoted $SUBST(\theta, E)$.

Substitution Examples

- $\theta = \{x/a, y/f(b), z/c\}, E = P(x, y, z)$
  - $E\theta = P(a, f(b), c)$
- $\theta = \{x/f(x), y/x\}, E = P(x, y)$
  - $E\theta = P(f(x), x)$
- $\theta = \{x/Socrates\}, E = \neg MAN(x) \vee MORTAL(x)$
  - $E\theta = \neg MAN(Socrates) \vee MORTAL(Socrates)$

Composition of Substitutions

Let $\theta = \{x_1/t_1, ..., x_n/t_n\}$ and $\lambda = \{y_1/u_1, ..., y_m/u_m\}$ be substitutions. Then the **composition** of θ and λ, denoted $\theta \circ \lambda$ is the substitution obtained from the set

$$\{x_1/t_1\lambda, ..., x_n/t_n\lambda, y_1/u_1, ..., y_m/u_m\}$$

by deleting any element $x_j/t_j\lambda$ such that $t_j\lambda = x_j$ (e.g. x_k/x_k is meaningless) and any element y_i/u_i such that $y_i \in \{x_1, ..., x_n\}$ (because y_i is already covered by θ).

Examples: Composition of Substitution

Given

$$\theta = \{x_1/t_1, x_2/t_2\} = \{x/f(y), y/z\}$$
$$\lambda = \{y_1/u_1, y_2/u_2, y_3/u_3\} = \{x/a, y/b, z/y\}$$

$$\begin{aligned}
\theta \circ \lambda &= \{x_1/t_1\lambda, x_2/t_2\lambda, y_1/u_1, y_2/u_2, y_3/u_3\} \\
&= \{\underbrace{x/f(y)\lambda, y/z\lambda}_{\theta}, \underbrace{x/a, y/b, z/y}_{\lambda}\} \\
&= \{x/f(b), \underbrace{y/y}_{\text{identity}}, \underbrace{x/a, y/b}_{\text{appeared in } \theta}, z/y\} \\
&= \{x/f(b), z/y\}
\end{aligned}$$

Unification

- A substitution θ is called a **unifier** for a set $\{E_1, ..., E_k\}$ iff $E_1\theta = E_2\theta = ... = E_k\theta$.
- The set $\{E_1, ..., E_k\}$ is said to be **unifiable** if there is a unifier for it.
- A unifier σ for a set $\{E_1, ..., E_k\}$ of expressions is a **Most General Unifier** iff for each unifier θ for the set there is a substitution λ such that $\theta = \sigma \circ \lambda$.
- A Most General Unifier will avoid unnecessary substitution(s).

Examples: Unification

$P(x, g(x))$ will unify with:

Expression	Necessary Substitution
$P(x, y)$	$\{y/g(x)\}$
$P(z, g(z))$	$\{z/x\}$ or $\{x/z\}$
$P(Socrates, g(Socrates))$	$\{x/Socrates\}$
$P(x, g(y))$	$\{x/y\}$ or $\{y/x\}$
$P(g(y), z)$	$\{x/g(y), z/g(g(y))\}$

but not with $P(Socrates, f(Socrates))$ or $P(g(y), y)$

Disagreement Set

Let W be a nonempty set of expressions $\{E_1, ..., E_n\}$. The **disagreement set** D of W is obtained by locating the first symbol (counting from the left) at which not all the expressions in W have exactly the same symbol, and then extracting from each expression E_i in W the subexpression that begins with the symbol occupying that position.

Example:

$$W = \{P(x, y, a, \mid f(x)),\ P(x, y, a, \mid g(x)),\ P(x, y, a, \mid z)\}$$

Symbols to the right of the vertical bar differ.

$$D = \{f(x), g(x), z\}$$

Disagreement Set: More Examples

Examples:

1. $W = \{P(a), P(x)\} \qquad D = \{a, x\}$
2. $$W = \{P(x, \mid f(y, a)),\ P(x, \mid a),\ P(x, \mid g(h(k(x))))\}$$
   $$D = \{f(y, a), a, g(h(k(x)))\}$$
3. $$W = \{P(x, f(g(\mid h(y)))),\ P(x, f(g(\mid z)))\}$$
   $$D = \{h(y), z\}$$

Unification Algorithm

Let $W = \{E_1, ..., E_n\}$ be the set of expressions to be unified.

1. If necessary, **rename** variables so that no pair (E_i, E_j) from different clauses has any variables in common.
2. **Set** $k = 0, W_k = W, \sigma_k = \epsilon$ (empty substitution).
3. **If** W_k **is a singleton** (contains only one expr), stop; σ_k is a most general unifier for W. **Otherwise**, let D_k be the disagreement set for W_k.
4. **If** there exist elements v_k and t_k in D_k such that v_k **is a variable that does not occur in term** t_k, go to step 5. **Otherwise**, stop; W is not unifiable.
5. Let $\sigma_{k+1} = \sigma_k \circ \{v_k/t_k\}$ and $W_{k+1} = W_k\{v_k/t_k\}$. (Note that $W_{k+1} = W_k\sigma_{k+1}$)
6. Set $k = k + 1$ and go to step 3.

Unification Theorem

If W is a finite nonempty *unifiable* set of expressions, then the unification algorithm will always terminate at step 3, and the last σ_k is a most general unifier for W (i.e. not unnecessary substitutions).

The algorithm must terminate because each pass through the loop reduces the number of variables by 1, and there are only finitely many of them.

Unification Example

$P(x, f(x), z)$ vs.
$P(g(y), f(g(a)), y)$:

1. $\{x/g(y)\}$:
   $P(g(y), f(g(y)), z)$
   $P(g(y), f(g(a)), y)$
2. $\{y/a\}$:
   $P(g(a), f(g(a)), z)$
   $P(g(a), f(g(a)), a)$
3. $\{z/a\}$:
   $P(g(a), f(g(a)), a)$

Unifier: $\{x/g(a), y/a, z/a\}$

Representation of Predicates and Terms in LISP

- Constants: a = `(A)`, Socrates = `(SOCRATES)`
- Variables: x = `X`, y = `Y`
- Functions: f(x)= `(F X)`, f(a,y,z)= `(F (A) Y Z)`
- Predicates: P(x)= `(P X)`, P(x,b,f(z))= `(P X (B) (F Z))`

Note how the representation of the constants can come in handy.

SUBLIS : substitution in LISP

`(sublis <list-of-alist> <expr>)`: simultaneous substitution

- **alist**, or association list: `(A . B)`, which is the same as `(cons 'A 'B)` (note that B is not a list but an atom in this case).
- `<list-of-alist>`: a list of `(<pattern> <replace>)` pairs.
- `<expr>`: the expression to be worked on.
- Replace every occurrence of `<pattern>` in `<expr>` with `<replace>`.

Another useful function: `(subst <repl> <pattern> <expr>)`

SUBLIS Examples

Basically, replace `(car alist)` with `(cdr alist)` of each element in the `<list-of-alist>`:

```
>(sublis '((x . (20))) '(* x 1))
(* (20) 1)

>(sublis '((x 20)) '(* x 1))
(* (20) 1)

>(sublis '((x . 20)) '(* x 1))
(* 20 1)

>(sublis '((x . 20) (y . 10)) '(* x (/ 5 y)))
(* 20 (/ 5 10))
```


Unification in LISP

```
(defun unify (u v)
  (let ((*u* (copy-tree u))
        (*v* (copy-tree v)) *subs*)
      (declare (special *u* *v* *subs*))
    (if (unifyb *u* *v*) (or *subs* (list (cons t t)))) )

(defun unifyb (u v)
  (cond ((eq u v))
        ((symbolp u) (varunify v u))
        ((symbolp v) (varunify u v))
        ((and (consp u) (consp v)
              (eq (car u) (car v))
              (eql (length (cdr u))
                   (length (cdr v))))
          (every #'unifyb (cdr u) (cdr v)) )) )
```


Unification in LISP (cont'd) [a]

```
(defun varunify (term var)
    (declare (special *u* *v* *subs*))
  (unless (occurs var term)
    (dolist (pair *subs*)
      (setf (cdr pair)
            (subst term var (cdr pair))))
    (nsubst term var *u*)
    (nsubst term var *v*)
    (push (cons var term) *subs*)))
```

[a] Code in this and the previous page by Gordon Novak, http://www.cs.utexas.edu/users/novak. Also downloadable at http://www.cs.tamu.edu/faculty/choe/courses/02spring/src/sunify.lsp

UNIFY : examples

```
(unify '(p x) '(p (a)))
(unify '(p (a)) '(p x))
(unify '(p x (g x) (g (b))) '(p (f y) z y))
(unify '(p (g x) (h w) w) '(p y (h y) (g (a))))
(unify '(p (f x) (g (f (a))) x) '(p y (g y) (b)))
(unify '(p x) '(p (a) (b)))
(unify '(p x (f x)) '(p (f y) y))
```

Resolution in Predicate Calculus

- Factors
- Binary resolvent
- Properties of resolution

Factor of a Clause

Definition: If two or more literals of a clause C (with the **same** sign) have a most general unifier σ, then $C\sigma$ is called a **Factor** of C. If $C\sigma$ is a unit clause, it is called a **Unit Factor** of C.

Example: $C = P(x) \vee P(f(y)) \vee \neg Q(x)$.

- The first two literals have a unifier $\sigma = \{x/f(y)\}$, so C has a factor $C\sigma = P(f(y)) \vee \neg Q(f(y))$.

Note: Factors of a clause are much succinct and when two clauses C_1 and C_2 cannot be resolved directly, their factors (let's call them C_1' and C_2' **can be** resolved.

Resolving Two Clauses

Definition: Let C_1 and C_2 be two clauses (called *parent clauses*) with no variables in common, and with complementary literals L_1 and L_2 such that L_1 and $\neg L_2$ have a most general unifier σ. Then the clause

$$(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$$

is called a **binary resolvent** of C_1 and C_2. The literals L_1 and L_2 are called **the literals resolved upon**.

Note: A clause can be treated as a set of literals.

$$\{P(x)\} \cup \{Q(x)\} = \{P(x), Q(x)\} = P(x) \vee Q(x)$$

Example: Resolve the following (hint: $\sigma = \{x/a\}$)
$C_1 = P(x) \vee Q(x)$ and $C_2 = \neg P(a) \vee R(y)$.

Resolvent

Definition: A *resolvent* of parent clauses C_1 and C_2 is one of the following binary resolvents:

1. a binary resolvent of C_1 and C_2
2. a binary resolvent of C_1 and a factor of C_2
3. a binary resolvent of a factor of C_1 and C_2
4. a binary resolvent of a factor of C_1 and a factor of C_2

Example: resolve the two clauses
1. $C_1 = P(x) \vee P(f(y)) \vee R(g(y))$ and
2. $C_2 = \neg P(f(g(a))) \vee Q(b)$.

(hint: resolve the factor of C_1 and clause C_2)

Property of Resolution for First-Order Logic

- **Complete**: If a set of clauses S is unsatisfiable, resolution will *eventually* derive $\mathbf{False}$.
  - *Everything that is true can be proved (eventually).*
- **Sound**: If $\mathbf{F}$ is derived by resolution, then the original set of clauses S is unsatisfiable.
  - *Everything that is proved is true.*

Weakness of Resolution

Basically, resolution tries to derive

Axioms $\wedge \neg$ Theorem = $\mathbf{F}$

- Is there a $\mathbf{F}$ in the axioms? If there is, the whole formula will always be unsatisfiable no matter what.
- Can we tell whether axioms alone can derive $\mathbf{F}$? (generally, this is not the case)

Key Points

- substitution and unification: why are these necessary and how to do them.
- unification algorithm
- factors : definition, and how to derive, why factors are important
- resolvent : definition, and how to derive

Overview

- Resolvents
- Resolution in first order logic: example
- Theorem proving strategies
- Application of theorem proving: question answering

Resolving Two Clauses: Revisited

Resolving two clauses C_1 and C_2 with the most general unifier σ:

$$(C_1\sigma - L_1\sigma) \cup (C_2\sigma - L_2\sigma)$$

This is basically:

1. Find the most general unifier σ.
2. Apply σ to both C_1 and C_2
3. Remove the complimentary literal from C_1 and C_2.

Resolving Two Clauses: Example Revisited

Example: Resolve the following (hint: $\sigma = \{x/a\}$)

$$C_1 = \underbrace{P(x)}_{L_1} \vee Q(x) \text{ and } C_2 = \underbrace{\neg P(a)}_{L_2} \vee R(y)$$

$$C_1 = P(x) \vee Q(x) \qquad C_2 = \neg P(a) \vee R(y)$$
$$| \qquad\qquad\qquad | $$
$$\sigma = \{x/a\} \qquad\qquad \sigma = \{x/a\}$$
$$\downarrow \qquad\qquad\qquad \downarrow$$
$$C_1\sigma = P(a) \vee Q(a) \qquad C_2\sigma = \neg P(a) \vee R(y)$$
$$| \qquad\qquad\qquad | $$
$$\text{remove } L_1\sigma = P(a) \qquad \text{remove } L_2\sigma = \neg P(a)$$
$$\downarrow \qquad\qquad\qquad \downarrow$$
$$(C_1\sigma - L_1\sigma) = Q(a) \qquad (C_2\sigma - L_2\sigma) = R(y)$$

Resolvent: A Full Example

Example: resolve the two clauses

1. $C_1 = P(x) \vee P(f(y)) \vee R(g(y))$ and
2. $C_2 = \neg P(f(g(a))) \vee Q(b)$.

1. Get the factor of C_1:
$$C_1\{x/f(y)\} = P(f(y)) \vee R(g(y))$$
2. Resolve factor of C_1 and C_2:
$$P(f(y)) \vee R(g(y)) \text{ vs. } \neg P(f(g(a))) \vee Q(b)$$
3. $\sigma = \{y/g(a)\}$:
$$\underbrace{P(f(g(a)))}_{\text{remove}} \vee R(g(g(a))) \text{ vs. } \underbrace{\neg P(f(g(a)))}_{\text{remove}} \vee Q(b)$$
4. Result:
$$R(g(g(a))) \vee Q(b)$$

Example Proof Using Resolution [a]

Given: (1) The customs officials searched everyone who entered the country who was not a VIP. (2) Some of the drug dealers entered the country, and they were only searched by drug dealers. (3) No drug dealer was a VIP.

Prove: (4) Some of the customs officials were drug dealers.

[a] Chang & Lee, Example 5.22

Example: Predicates

1. $C(x)$: x is a customs official
2. $E(x)$: x entered the country
3. $V(x)$: x is a VIP
4. $S(x, y)$: x was searched by y
5. $D(x)$: x is a drug dealer

Example: English to First Order Logic

(1) The customs officials searched everyone who entered the country who was not a VIP. (2) Some of the drug dealers entered the country, and they were only searched by drug dealers. (3) No drug dealer was a VIP. (4) Some of the customs officials were drug dealers.

1. $\forall x((E(x) \wedge \neg V(x)) \rightarrow \exists y(S(x, y) \wedge C(y)))$
2. $\exists x(E(x) \wedge D(x) \wedge \forall y(S(x, y) \rightarrow D(y)))$
3. $\forall x(D(x) \rightarrow \neg V(x))$
4. $\exists x(D(x) \wedge C(x))$

Example: Standard Form (I)

$$
\begin{array}{lll}
\text{(1)} & & \forall x((E(x) \wedge \neg V(x)) \rightarrow \exists y(S(x, y) \wedge C(y))) \\
\{rm\ \rightarrow\} & = & \forall x(\neg(E(x) \wedge \neg V(x)) \vee \exists y(S(x, y) \wedge C(y))) \\
\{prenex\} & = & \forall x \exists y(\neg E(x) \vee V(x) \vee (S(x, y) \wedge C(y))) \\
\{skol\} & = & \forall x(\neg E(x) \vee V(x) \vee (S(x, f(x)) \wedge C(f(x)))) \\
\{add()\} & = & \forall x((\neg E(x) \vee V(x)) \vee (S(x, f(x)) \wedge C(f(x)))) \\
\{dist\} & = & \forall x(\underbrace{(\neg E(x) \vee V(x) \vee S(x, f(x))}_{} \\
 & & \quad \wedge \underbrace{(\neg E(x) \vee V(x) \vee C(f(x))))}_{}
\end{array}
$$

Clauses:

(1a) $\neg E(x) \vee V(x) \vee S(x, f(x))$

(1b) $\neg E(x) \vee V(x) \vee C(f(x))$

Example: Standard Form (II)

$$
\begin{array}{lll}
\text{(2)} & & \exists x(E(x) \wedge D(x) \wedge \forall y(S(x, y) \rightarrow D(y))) \\
\{rm\ \rightarrow\} & = & \exists x(E(x) \wedge D(x) \wedge \forall y(\neg S(x, y) \vee D(y))) \\
\{prenex\} & = & \exists x \forall y(E(x) \wedge D(x) \wedge (\neg S(x, y) \vee D(y))) \\
\{skol\} & = & \forall y(\underbrace{E(a)} \wedge \underbrace{D(a)} \wedge \underbrace{(\neg S(a, y) \vee D(y))})
\end{array}
$$

Clauses:

(2a) $E(a)$

(2b) $D(a)$

(2c) $\neg S(a, y) \vee D(y)$

Example: Standard Form (III)

(3) $\forall x(D(x) \rightarrow \neg V(x))$

$\{rm \ \rightarrow\}$ = $\forall x(\neg D(x) \vee \neg V(x))$

Clause:

(3) $\neg D(x) \vee \neg V(x)$

(4) $\exists x(D(x) \wedge C(x))$

$\{negate\}$ $\Rightarrow$ $\neg(\exists x(D(x) \wedge C(x)))$

$\{prenex\}$ = $\forall x \neg(D(x) \wedge C(x)))$

$\{CNF\}$ = $\forall x(\neg D(x) \vee \neg C(x))$

Clause:

(4) $\neg D(x) \vee \neg C(x)$

Example: Clauses

(1a) $\neg E(x) \vee V(x) \vee S(x, f(x))$

(1b) $\neg E(x) \vee V(x) \vee C(f(x))$

(2a) $E(a)$

(2b) $D(a)$

(2c) $\neg S(a, y) \vee D(y)$

(3) $\neg D(x) \vee \neg V(x)$

(4) $\neg D(x) \vee \neg C(x)$

Note: The input to your theorem prover will be in a standard form like the above.

Exercise 1: rewrite the above in LISP representation.

Exercise 2: use resolution to derive $\mathbf{F}$.

Basic Theorem Proving Algorithm

Level saturation resolution method (or two-pointer method)

Generate all possible resolvents:

- Generate sequences of clauses $S^0, S^1, S^2, ...$, where

  S^0 = S (original set of clauses)

  S^n = { all possible resolvents of clauses $C_1 \in (S^0 \cup ...S^{n-1})$ and $C_2 \in S^{n-1}$ }

- This is basically a **breadth first search** method, and it can be extremely inefficient except for small problems.
- The problem is that **irrelevant** derivations are made: in generating an n-step proof, we also generate **all possible derivations** of n-1 steps.

Deletion Strategy

To reduce the huge number of generated clauses, we would like to delete clauses whenever possible. We can delete:

1. Any tautology, e.g. $P(a) \vee \neg P(a) \vee Q(x)$.
2. Any clause which duplicates an existing clause.
3. Any clause which is **subsumed by** an existing clause.

A clause C **subsumes** a clause D iff there is a substitution σ such that $C\sigma \subseteq D$ (recall that a clause can be represented as a set of literals). D is called a **subsumed clause**.

Deletion strategy will be complete if it is used with certain resolution algorithms (such as level saturation).

Subsumed Clause: Example (I)

Example:

- $C = P(x)$
- $D = P(a) \vee Q(a)$
- If $\sigma = \{x/a\}$, then
  $C\sigma = P(a) = \{P(a)\}$
  $\subseteq \{P(a), Q(a)\} = P(a) \vee Q(a) = D$.
- Since $C\sigma \subseteq Q$, C subsumes D, and D can be deleted.

Strategies to Improve Resolution

1. **Deletion strategy**: remove tautology, duplicates, and subsumed clauses.
2. **Unit preference**: resolve with clauses with the fewest literals.
3. **Set of support**: begin with set T consisting of the clauses from the **negated conclusion**. Each resolution step must involve a member of T, and the result is added to T.
4. **Linear resolution** (Depth First): Each step must be a resolution step involving the clause produced by the last step.

Advantages and Disadvantages of Resolution

- **Advantages:** (1) Resolution is universally applicable to problems which can be described in first-order logic. (2) The theorem proving engine can be decoupled from any particular domain.
- **Disadvantage:** (1) Resolution is too inefficient to be generally applicable. (2) This is partly because resolution is purely syntactic, and it does not consider what the predicates *mean*. For this reason, developing a domain-dependent heuristic is impossible. (3) A contradiction in the axiom set may allow anything to be proved. (4) It is difficult for a human to understand proof by resolution prover.

Application of the Theorem Prover: Question Answering

- Given a database of facts (ground instances) and axioms, we can pose questions in predicate calculus and answer them using resolution.
- Resolution can answer **Yes**/**No** answers, but it can be extended to answer more complex questions such as **Who?** or **What?**, etc. This is called **Answer Extraction**.

Question Answering: Example

Example:

1. $\forall x \forall y \forall z((Parent(x, z) \wedge Parent(z, y)) \rightarrow Grandparent(x, y)$
2. $\forall x \forall y(Mother(x, y) \rightarrow Parent(x, y))$
3. $\forall x \forall y(Father(x, y) \rightarrow Parent(x, y))$
4. $Father(Zeus, Ares)$
5. $Mother(Hera, Ares)$
6. $Father(Ares, Harmonia)$

Question: "Who is a grandparent of Harmonia?"

1. $\exists x(Grandparent(x, Harmonia))$
   Negated: $\neg \exists x(Grandparent(x, Harmonia))$
   $= \forall x(\neg Grandparent(x, Harmonia))$

Question Answering: Result

- Resolution on the previous example generates $\mathbf{F}$ in the end, but what that answers is the question **"Is there a grandparent of Harmonia?"**. Of course the answer is **yes**, but the question is **who?**.
- The negated question in the above examples was $\neg Grandparent(x, Harmonia)$. Clearly, the binding which x ultimately receives is the desired answer!
- Observation: one substitution along the way, starting from $\neg Grandparent(x, Harmonia)$, the negated conclusion, is $\{x/Hera\}$, thus $Hera$ must be an answer.

Exercise: use resolution to derive $\mathbf{F}$ in the example in the previous slide.

Answer Extraction

We can introduce special predicates to extract the answers.

- **Answer predicate**:
  $\neg Grandparent(x, Harmonia) \vee Answer(x)$
- The answer predicate has these properties:
  - It does not resolve with anything, but it keeps track of variable bindings.
  - The theorem prover recognize a clause consisting only of the $Answer$ predicate as $\mathbf{F}$.
- For example, resolution on the previous example results in:
  $$Answer(Hera)$$
  as the final clause.

First-Order Logic: Summary

- Standard forms: prenex normal form, skolemization, CNF.
- Resolution: negated conclusion, substitution, unification, factors and resolvents.
- Theorem provers: two-pointer method, various deletion strategies, various speed up strategies.
- Application of theorem provers: question answering.

Key Points

- resolvent : definition, and how to derive
- properties of resolution: sound and complete
- theorem proving algorithm: level saturation (two pointer method)
- theorem proving: strategies for efficient resolution
- advantages and disadvantages of resolution.
- application: answer extraction.