

Objectives:

1. To introduce the first order predicate calculus, including the syntax of WFFs
2. To introduce formalization of knowledge using predicate calculus
3. To introduce resolution-refutation theorem proving, including unification and conversion to clause form
4. To discuss the formal basis for Prolog in predicate calculus
5. To discuss resolution-refutation proof as a search problem

Materials:

1. Handout with laws of equivalence of formulas
2. Projectable Form of unification algorithm
3. Demonstration of Project 3

I. Introduction to the First Order Predicate Calculus

A. One notation system that is widely used in AI systems is the notation of formal mathematical logic.

1. Formal logic is much older than AI or computer science. Formal logic is a mathematical formalism for reasoning about assertions.
2. Formal logic is not only used by mathematicians, but also by philosophers. Here at Gordon, the philosophy department teaches a course that (among other things) teaches and uses formal logic.
3. Behind formal logic is a history of hundreds of years of work. Thus, it is a well-understood tool. (In fact, it is generally recognized that the work of logicians like Boole, Frege, Russell and others helped lay the foundation for AI.)
4. The particular type of formal logic we will use is called the first order predicate calculus. This is the formalism most widely used by AI workers.
 - a) Predicate calculus formulas can easily be represented using LISP, and some of the first work using predicate calculus for AI was done in this language.
 - b) Predicate calculus is also the basis of Prolog. This will become obvious in the next series of lectures (on Prolog).

- c) Your book (and many AI books) eases into predicate calculus by way of a less powerful system of notation called the PROPOSITIONAL CALCULUS. Since you should have seen some of this before in Discrete Math, we will jump straight into the predicate calculus

(Predicate calculus is essentially propositional calculus with the building blocks being predicates).

- 5. Predicate calculus is not a panacea for all problems, though. In particular, it has serious difficulties dealing with realities we often encounter in human reasoning, such as:

- a) Incomplete knowledge. Example: in a medical diagnostic system it may be necessary to take the patient's age into consideration in certain cases. How shall the system cope with a situation where the age of a particular patient is not known?
- b) Inexact knowledge. To continue the above example, maybe all we know is that the age is between 40 and 50.
- c) Uncertain knowledge. Often times we face situations where we say things like "I'm fairly sure that this is true".
- d) Non-monotonic knowledge. Sometimes new information causes us to invalidate a belief we had previously accepted. Example: if told that a certain creature is a bird, we would ordinarily assume it can fly. If we are later told it is a penguin, that assumption and all inferences built on it would have to be canceled.
- e) We will look at traditional formal logic now. Later in the course, we will look briefly at some approaches to addressing issues like these in the general context of formal logic.

- 6. Nonetheless, predicate calculus will serve our purposes well.

- 7. One good source if you wish to study this material beyond what is in the text is:

Nils Nilsson. *Principles of Artificial Intelligence*. (Palo Alto: Tioga, 1980).

This book has a very readable and thorough discussion of predicate calculus.

- B. The first order predicate calculus is a formal language for expressing propositions. Like any formal language, it has a well-defined syntax.
- C. A properly-formed predicate calculus expression is called a well-formed formula or WFF (pronounced wiff). WFFs are built up out of several basic types of building block:

1. Constants: a constant is a symbolic name for a real-world person, object, event etc.

Ex: Suppose we were building a database of information about pets in my home. Constants that might appear would include:

rocco (my dog)
alexander (my cat)
etc.

- a) Note that logicians often use the convention that constant names begin with an uppercase letter (or sometimes a numeral). This is the exact opposite of the convention in Prolog - where constants (atoms) begin with lowercase letters. The book follows the Prolog conventions; we will, as well.
- b) Numbers (often integers) can also serve as constants where appropriate.
- c) Constants are also called atoms.

2. Predicates (or relation constants)

- a) A predicate is an assertion that some property or relationship holds for one or more arguments.

For example, to assert that Rocco is a dog, we would write:

dog(rocco)

To assert that Rocco chases Alexander, we would write:

chases(rocco, alexander)

- b) Logicians often use the convention that predicate names begin with an uppercase letter. Again, this is the opposite of the convention in Prolog, where predicate names begin with a lower-case letter. Again, the book follows the Prolog conventions; we will, as well.

c) Predicates have a truth value - they are either true or false - e.g.

dog(rocco)	is true
dog(alexander)	is false

d) Note that when a predicate has more than one argument, the order of arguments is significant. However, the choice of order is up to the designer of the predicate. I simply chose, in defining the chases predicate, to put the chaser first and the chasee second. I could have equally well used the reverse convention.

e) Further, even the number of arguments (arity) for expressing a given concept as a predicate is flexible. For example, instead of a chases predicate of two arguments, I could have defined a specialized chasesCats predicate of one argument, yielding WFFs like

chasesCats(rocco)

(Which variant I would prefer would depend on what I was planning to do with the database. If chasing cats were a major category of information, then the second form might be preferred; but if I wished to deal with chasing other animals in a more general sense, I might prefer the former variant.)

f) Lurking in the background of our discussion here is a profound philosophical issue.

(1) The rules of predicate calculus are SYNTACTIC rules - they specify the FORM of predicates, but not their meaning (their SEMANTICS).

(2) An AI system that uses predicate calculus manipulates formulas according to syntactic rules. If the system is designed correctly, then the results it produces make sense when interpreted in light of the semantics intended by the designer.

(3) But is there any sense in which a system that manipulates formulas according to syntactic rules can be said to UNDERSTAND what it is doing? This question lies at the heart of some of the debates over the very nature of AI - for example, the Searle article we will discuss in two weeks.

3. Variables

- a) A variable stands for a (currently unknown) constant, or for all possible constants. For example:

$\text{dog}(X)$

is a predicate that is true for any X that is a canine.

- b) Logicians often use the convention that variables are given lower-case letters as names - often names like x, y, z . Once again, this is the opposite of the convention of Prolog, where variables either begin with an upper-case letter, or with the underscore character ($_$). Again, the book follows the Prolog conventions; we will, as well. The following are all valid variable names:

X
 Dog
 $_1$

- c) Note that, in first-order predicate calculus, a variable may only appear in place of a constant. There are no “predicate variables”. This is what distinguishes first-order predicate calculus from higher-order logics, in which variables can stand for predicates as well as constants. Most AI work has stuck with the first-order calculus, which turns out not to be unduly constraining - there are some “tricks” that can be used to get the effect of predicate variables, as we shall see later in the course.

4. Connectives

- a) Complex WFFs can be built up by joining simpler WFFs using the following connectives:

(1) \cap - and. This operation is also called conjunction.

(2) \cup - or. This operation is also called disjunction.

(3) \neg - not. This operation is also called negation.

(4) \supset or \rightarrow implies. This operation is also called implication.

(a) The expression on the left is called the antecedent

(b) The expression on the right is called the consequent

Ex: $\text{dog}(X) \rightarrow \text{barks}(X)$

- $\text{dog}(X)$ is the antecedent
- $\text{barks}(X)$ is the consequent

(5) \equiv equivalence. (Two formulas are equivalent if the first is true just when the second is true and vice versa)

b) The first three are familiar from conventional programming languages. The implies connective may be new to you.

(1) It is what lets us translate if .. then rules.

Ex: The English statement "dogs chase cats" can be represented by an if .. then rule as:

if animal is a dog
then animal chases cats

This can be written as the following WFF:

$\text{dog}(X) \rightarrow \text{chasesCats}(X)$

or

$\text{dog}(X) \cap \text{cat}(Y) \rightarrow \text{chases}(X, Y)$

(2) It is important to note carefully the conditions under which an implication is true. An implication is true if the antecedent is true and the consequent is true. It is also true if the antecedent is false, regardless of the value of the consequent. For example, the following are true implications (given our commonsense understanding of things.)

(a) $\text{beagle}(X) \rightarrow \text{dog}(X)$

(b) $\text{madeOf}(\text{greenCheese}, \text{moon}) \rightarrow \text{president}(\text{us}, \text{bush})$

(c) $\text{madeOf}(\text{greenCheese}, \text{moon}) \rightarrow \text{president}(\text{us}, \text{kerry})$

The first is true because whenever any animal is a beagle it is also a dog. The second and third are true (regardless of the truth value of the consequents) because the moon is not made of green cheese.

(3) Note well that $A \rightarrow B$ is equivalent to $\neg A \cup B$. We will make use of this equivalence many times.

c) The meaning of the connectives can be represented by truth tables as follows:

<u>A</u>	<u>B</u>	<u>$A \cap B$</u>	<u>$A \cup B$</u>	<u>$\neg A$</u>	<u>$A \rightarrow B$</u>	<u>$A \equiv B$</u>
F	F	F	F	T	T	T
F	T	F	T	T	T	F
T	F	F	T	F	F	F
T	T	T	T	F	T	T

5. Quantifiers

a) Consider a predicate applied to some variable - e.g. $p(X)$. There are two ways to interpret such a statement:

(1) We could interpret it as meaning to make a statement that is true for all possible values of the variable. The above formulation of the “dogs chase cats” rule is of this sort.

$$\text{dog}(X) \cap \text{cat}(Y) \rightarrow \text{chases}(X, Y)$$

The variables X and Y are understood as applying to all dogs and cats, respectively

(2) We could interpret it as meaning that there is at least one value of the variable for which the predicate is true. For example, taken by itself the predicate $\text{dog}(X)$ would have to be understood in this sense. Certainly it is not the case that the dog predicate is true for all animals in the pets world (to say nothing of all possible values of X - including us!); but read as “there is some pet for which $\text{dog}(X)$ is true” it does make sense.

b) The correct interpretation of a variable is made explicit by preceding the WFF in which it appears with one of two quantifiers - \forall or \exists .

(1) The universal quantifier \forall (read “for all”) indicates that any value may be substituted for the indicated variable.

(2) The existential quantifier \exists (read “there exists”) indicates that there is some value (at least one, perhaps more) of the indicated variable for which the WFF is true.

(3) When a variable appears in a WFF, the choice of quantifier affects the meaning of the WFF.

- (a) For example, for the rule about dogs barking, the universal quantifier is probably what we want:

$$(\forall X) (\text{dog}(X) \rightarrow \text{barks}(X))$$

This says “all dogs bark” (“for all X such that X is a dog, X barks”)

Since an implication is true whenever its antecedent is true (regardless of the value of its consequent), we are not saying anything about any X that is not a dog.

(The existential quantifier would also yield a meaningful sentence, but probably not what we want:

$$(\exists x) (\text{dog}(X) \rightarrow \text{barks}(X))$$

“There exists at least one creature such that its being a dog implies that it barks” (but there might be other dogs that don’t bark). Of course, given the meaning of implies, this would be true even if there were no dogs in the world, provide there was at least one non-dog!

- (b) On the other hand, if we had the WFF $\text{dog}(X)$, we clearly want the existential quantifier:

$$(\exists X) (\text{dog}(X))$$

This says “some dog exists” (“there exists an X such that X is a dog.”)

(If we used the universal quantifier

$$(\forall X) (\text{dog}(X))$$

we would be saying “everything is a dog” - Rocco, Alexander, you, me, the chair you’re sitting in ...)

- c) Quantifiers should be chosen to reflect the meaning intended for the WFF.

- (1) For example, in our formula about dogs chasing cats the variables should be quantified universally:

$$(\forall X) (\forall Y) \text{dog}(X) \cap \text{cat}(Y) \rightarrow \text{chases}(X, Y)$$

This reflects our intention that it be interpreted as a rule describing the behavior of all dogs toward all cats.

- (2) On the other hand, suppose we wanted to assert that “in our world there is a cat that chases dogs”. Here, we would use the existential quantifier for the “cat variable” to get:

$$(\exists Y) \text{cat}(Y) \cap ((\forall X) \text{dog}(X) \rightarrow \text{chases}(Y, X))$$

(Note also the rearrangement of \cap and \rightarrow)

“There is some creature who is a cat and who chases any creature that is a dog”.

- d) Normally, in a WFF, all variables are quantified. Such variables are said to be bound. Any variable that is not quantified is said to be free. We will not work with WFFs containing free variables.
- e) We speak of the scope of a quantifier as the region in which the variable it names is bound. For example, in

$p(a) \cup (\forall X) (q(a, X)) \cup (\exists X) r(a, X)$, the scope of the $(\forall X)$ quantifier is $q(a, X)$, and the variable “X” that appears in $r(a, X)$ is not necessarily the same as the variable “X” that appears in $q(a, X)$

- f) We will see later that any WFF can be written in such a way as to eliminate all existential quantifiers. When a WFF is in this form, all variables are of necessity universally quantified; therefore, it is common practice to drop the quantifiers. Thus, if you see a WFF with variables but no quantifiers, you may generally assume that all the variables are universally quantified.

Ex: $(\forall X) (\forall Y) \text{dog}(X) \cap \text{cat}(Y) \rightarrow \text{chases}(X, Y)$

is often written as simply

$$\text{dog}(X) \cap \text{cat}(Y) \rightarrow \text{chases}(X, Y)$$

with $(\forall X) (\forall Y)$ understood.

6. Functions

- a) A predicate calculus function stands for some (currently not known) constant that is related in some way to its argument(s). For example, in our pets world every pet has an owner. We may therefore invent an owner function to use in our formulas.

Ex: “The owner of Rocco” could be represented by

owner(rocco)

Ex. The statement “the owner of a cat that is chased by a dog will be mad at the owner of the dog” can be represented by:

$$(\forall X) (\forall Y) [(dog(X) \cap cat(Y) \cap chases(X, Y)) \rightarrow madAt(owner(Y), owner(X))]$$

- b) By convention, function names are written in lower case letters. Often, a single letter near the middle of the alphabet (f, g, h etc.) is used.
- c) Notice that for every function we can invent a related predicate - e.g. for the function owner(X) we have the related predicate owner(X, Y) that asserts that the owner of X is Y (or vice versa if you prefer - just be consistent).

(1) The meanings are different, however:

(a) owner(X) is a function yielding an individual

(b) owner(X, Y) is a predicate that is true just when Y owns X

(2) Therefore, the following is certainly true:

$$(\forall X) (pet(X) \supset owner(X, owner(X)))$$

(3) However, we would rarely use both the function and the related predicate in the same WFF, because this can be quite confusing!

- d) In our example about the owners of chased cats, we could have used the owner predicate instead:

$$(\forall W) (\forall X) (\forall Y) (\forall Z) [(dog(W) \cap cat(X) \cap chases(W, X) \cap owner(W, Y) \cap owner(X, Z)) \rightarrow madAt(Z, Y)]$$

We will see shortly that if our database includes assertions about owners, then the latter form would be more easily used in reasoning. In practice, functions are avoided where possible because predicate calculus has no mechanism for expressing algorithms for computing them.

D. Rules of equivalence for WFFs

1. We say that two WFFs are equivalent if the first is true when, and only when, the second is true. We denote this by $W1 \equiv W2$.

For example, we have already noted that:

$$(A \rightarrow B) \equiv (\neg A \cup B)$$

2. DISTRIBUTE, GO OVER HANDOUT WITH RULES OF EQUIVALENCE

Especially note properties of quantifiers, including effect of negation

II. Formalization using the First Order Predicate Calculus.

- A. We use the term "formalization" to describe the process of converting an English statement into an equivalent expression in some formal language like predicate calculus. Any sort of automated reasoning process will require some sort of formalization scheme.

B. In general, when we formalize an English statement:

1. We convert verbs to predicates
2. We convert nouns and adjectives to constants.
3. Clauses of the form: "Every ____ has ____ property" or "All members of ____ category ____" generally translate into a WFF with one (or more) universally-quantified variables with an implication in the scope of the quantifier.

Ex: "Dogs bark" (i.e. "every dog has the barks property" or "all members of the dog category bark")

$$(\forall X) (\text{dog}(X) \rightarrow \text{barks}(X))$$

As a general rule, universally quantified variables always contain a top-level implication in their scope - otherwise, we are making a statement about every conceivable entity!

3. “Rocco chases cats” $\text{chasesCats}(\text{rocco})$
or $(\forall X) (\text{Cat}(X) \rightarrow \text{chases}(\text{rocco}, X))$
4. “Cats eat mice” $(\forall X) (\text{cat}(X) \rightarrow \text{eatsMice}(X))$
or $(\forall X) (\forall Y) (\text{cat}(X) \cap \text{mouse}(Y) \rightarrow \text{eats}(X, Y))$
5. “The cat who lives at #11 is orange” $\text{address}(\text{cat1}, 11) \cap \text{color}(\text{cat1}, \text{orange})$
or $(\exists X)(\text{cat}(X) \cap \text{address}(X, 11) \cap \text{color}(X, \text{orange}))$

Note: There are a number of wrong ways to formalize this:

$$(\forall X) (\text{cat}(X) \cap \text{address}(X, 11) \rightarrow \text{color}(X, \text{orange}))$$

- This does not require that there actually be such a cat - it is satisfied if no cat lives at #11.

- This says that any cat who lives at #11 is orange - not that a particular cat that lives there is orange. (This would allow for an arbitrary number of orange cats at #11, and would preclude a non-orange cat at #11 - though probably the latter is intentional.)

$$(\forall X) (\text{cat}(X) \cap \text{address}(X, 11) \cap \text{color}(X, \text{orange}))$$

- This says that everything is a cat, lives at #11, and is orange - i.e. all of us are orange cats living at #11!

D. Some more complicated examples arise where we need quantifiers inside the scope of other quantifiers:

1. “Every mailman has been chased by a dog”

$$(\forall X) [\text{mailman}(X) \supset (\exists Y) \{ \text{dog}(Y) \cap \text{hasChased}(Y, X) \}]$$

2. “Every city has a dogcatcher who has been bitten by every dog in town” (Example from Nilsson’s earlier book)

$$(\forall X) [\text{city}(X) \rightarrow (\exists Y) \{ \text{dogCatcher}(Y) \cap \text{worksFor}(Y, X) \cap (\forall Z) (\text{dog}(Z) \cap \text{livesIn}(Z, X) \rightarrow \text{hasBitten}(Z, Y)) \}]$$

E. It should be clear that formalizing nontrivial statements is, at best, difficult. In fact, the debate over whether all knowledge is, in principle, formalizable lies in the background over some of the debates about the very possibility of strong symbolic AI. (Dreyfus discusses this when he talks about the relationship between Western philosophy and strong AI).

There are several difficulties that those who wish to formalize all knowledge must address:

1. The choice of an appropriate set of predicate constants to facilitate inference.

Ex: Do we use special-purpose predicates like $\text{chasesCats}(X)$, $\text{chasesMice}(X)$, or a more general-purpose predicate like $\text{chases}(X, Y)$ or even $\text{property}(\text{chases}, X, Y)$?

2. The difficulty of coping with knowledge that is not purely “boolean” - as we noted earlier, knowledge that is incomplete, inexact, or uncertain. (We will discuss logics that attempt to deal with these issues later - e.g. probabilistic and fuzzy logics)

III. Making inferences using predicate calculus

A. One of the most important reasons for using predicate calculus in AI is that there exists a body of well-understood mechanisms for making inferences from predicate-calculus WFFs. The terminology often used in discussing this is the terminology of mathematical proof - note that predicate calculus was first developed as a tool for use in the proof process.

1. An axiom is a WFF that is asserted to be true without proof. In an AI system, the axioms would be:
 - a) The domain-specific knowledge rules in the database, and
 - b) The input data supplied by the user.
2. A theorem is a WFF that can be proven true on the basis of the axioms. In an AI system, the theorems would be:
 - a) Inferences that can be drawn from the rules and input data (in a forward chaining system.)
 - b) Questions posed by the user.

Note, for example, that a question like “Who chases Alexander?” can be turned into a predicate calculus theorem

$(\exists X) (\text{chases}(X, \text{alexander}))$

As we shall see, the method of proof we use with theorems containing existentially quantified variables has, as a side effect, the finding in the knowledge base of a value for the variable for which the desired condition holds. This, of course, would answer the original question.

c) Technically, we say that our task is to prove that a given WFF is a theorem - i.e. it is only a theorem if it can be shown to be true.

3. Thus, “reasoning” in a logic-based AI system is accomplished by using methods of mathematical proof. Since these have a long history, they provide a wealth of resources for us to draw on in doing AI.

B. In formulating proofs, one of our most important tools are the laws of inference which allow us to form new theorems from axioms and other theorems. A rule of inference says given the truth of some specified WFF, we may infer the truth of some other specified WFF. (Note: this is implication, not equivalence. The second WFF must be true when the first one is, but not vice versa necessarily.) Several of the ones that will be important to us actually come from propositional logic:

1. Modus ponens:

Given $A \rightarrow B$ -- where A and B are any WFFs
and A

We may infer B

Note: We say that a rule of inference is sound if it can be shown to always yield correct results. Note that modus ponens can be proved to be sound as follows:

Given:	$(A \rightarrow B) \cap A$	
	$(\neg A \cup B) \cap A$	Meaning of \rightarrow
	$(\neg A \cap A) \cup (B \cap A)$	Distributive property
	false $\cup (B \cap A)$	Law of non-contradiction
	$(B \cap A)$	Property of false
	B	Meaning of “and”

2. Modus tollens:

Given $A \rightarrow B$ -- where A and B are any WFFs
and $\neg B$

We may infer $\neg A$

3. Resolution

Given $A \cup B$ -- where A,B, C are any WFFs
 and $\neg A \cup C$

We may infer $(B \cup C)$

- a) The term $B \cup C$ is called the resolvent.
- b) To show the soundness of resolution, we may construct a truth table as follows:

<u>A B C</u>	<u>A \cup B</u>	<u>\negA \cup C</u>	<u>(A \cup B) \cap (\negA \cup C)</u>	<u>(B \cup C)</u>
F F F	F	T	F (irrelevant row)	F
F F T	F	T	F (irrelevant row)	T
F T F	T	T	T	T
F T T	T	T	T	T
T F F	T	F	F (irrelevant row)	F
T F T	T	T	T	T
T T F	T	F	F (irrelevant row)	T
T T T	T	T	T	T

Observe that, while the last two columns are not identical, $(B \cup C)$ is true whenever $(A \cup B) \cap (\neg A \cup C)$ is true (as well as at other times.) Hence, $[(A \cup B) \cap (\neg A \cup C)] \rightarrow (B \cup C)$, even though it is not the case that $[(A \cup B) \cap (\neg A \cup C)] \equiv (B \cup C)$ and resolution is a valid rule of inference. (The rows marked “irrelevant row” can be ignored, because at least one of $A \cup B$ or $\neg A \cup C$ is false.)

- c) Note that B and C may be single expressions, complex expressions, or NIL - empty.

(1) For example, if we resolve:

$$\begin{array}{l} A \cup B \\ \neg A \end{array}$$

we get B since “C” is NIL

(Formally, we are resolving $(A \cup B)$ with $(\neg A \cup \text{false})$, which leads to $(B \cup \text{false})$, which implies B)

- (2) If we resolve $A \cup B_1 \cup B_2 \cup B_3 \dots$
 with $\neg A \cup C_1 \cup C_2 \cup C_3 \dots$,
 we get $B_1 \cup B_2 \cup B_3 \dots \cup C_1 \cup C_2 \cup C_3 \dots$

(We do this by using the associative property of disjunction, treating $B_1 \cup B_2 \cup B_3 \dots$ as “B” and $C_1 \cup C_2 \cup C_3 \dots$ as “C”.)

- d) If we resolve $\neg A$ with A - ie. both “B” and “C” are NIL, we get NIL. If the outcome of any resolution operation is NIL, then the clauses involved are contradictory. We will use this in the method of resolution refutation, which is a special form of proof by contradiction.

It turns out that this particular proof strategy is easily automated, and is the basis of the inference procedure used by Prolog.

IV. Unification

- A. We have learned about proof rules in the propositional calculus that can be also used in the predicate calculus. What happens, though, when we try to do a proof with axioms/theorems that contain variables?

1. The rule of universal specialization (instantiation) says:

Given $(\forall X)w(X)$ -- where w is some WFF containing X
 We may infer $w(a)$ -- where a is any constant
 Ex: From $(\forall X) (\text{dog}(X) \rightarrow \text{chasesCats}(X))$
 We infer $\text{dog}(\text{rocco}) \rightarrow \text{chasesCats}(\text{rocco})$

(We also infer $\text{dog}(\text{bjork}) \rightarrow \text{chasesCats}(\text{bjork})$ - which is a true statement since I am not a dog!)

2. Our proof rules must now be adapted to allow for the presence of variables. In particular, we will consider the way resolution is adapted to handle variables. We will see later in this lecture that one requirement for doing resolution effectively is that our axioms be expressed as a set of clauses - i.e. a set of lists of literals or'ed together. Then, if we find two clauses of the form

$(A \cup B_1 \cup B_2 \cup B_3 \dots)$ and $(\neg A \cup C_1 \cup C_2 \cup C_3 \dots)$

we can resolve them to yield a new resolvent clause that can be added to the set.

3. When our clauses contain variables it is possible to use specialization to resolve clauses containing disjuncts that are not exact opposites.

Ex: Suppose we have $(\forall X) (\text{dog}(X) \rightarrow \text{chasesCats}(X))$
 and $\text{dog}(\text{rocco})$

The former is equivalent to $\neg\text{dog}(X) \cup \text{chasesCats}(x)$, from the definition of implies. However, clearly $\text{dog}(\text{rocco})$ is not the same as $\text{dog}(X)$, which is what we would need to resolve them. By using universal specialization we could change $\text{dog}(X)$ to $\text{dog}(\text{rocco})$ (and at the same time $\text{chasesCats}(X)$ to $\text{chasesCats}(\text{rocco})$, since both are in the scope of the same quantifier), allowing us to resolve:

$\neg\text{dog}(\text{rocco}) \cup \text{chasesCats}(\text{rocco})$ with $\text{dog}(\text{rocco})$ to conclude $\text{chasesCats}(\text{Rocco})$, as desired

- B. The process of making substitutions for variables in two clauses so that they can be resolved is called unification, and the set of substitutions is called a unifier.

Ex: In the above, the unifier is $X = \text{rocco}$

(Note that the substitution is applied to the entirety of the clause, not just to the “A” parts that will cancel.)

- C. Unification is important for two reasons:

1. It makes resolution of clauses containing variables possible.
2. We will see later that the unifier(s) used in a resolution proof provide our handle for using the proof outcome to answer questions.

- D. Generally speaking, the necessary unifier will be apparent when examining two clauses. However, there is a whole body of theory on unification, including a unification algorithm, that can help us:

1. A substitution is a set of pairs $\{t1/V1, t2/V2 \dots\}$ - meaning that $t1$ is to be substituted for $V1$, $t2$ for $V2$ etc.
2. We write an expression followed by a substitution to denote the expression that results from making the specified substitution to the specified expression.

Ex: $[p(X, f(X, Y))] \{a/X, b/Y\} \equiv p(a, f(a,b))$

3. If we have two formulas A and B (at least one of which contains variables) and there is a substitution s that makes them identical, then s is a unifier for A and B .

Ex: A unifier for $p(a, X)$ and $p(Y, Z)$ is $\{a/Y, X/Z\}$.

4. Often there will be more than one possible unifier for a pair of formulas

Ex: Another unifier for the above is $\{a/Y, b/X, b/Z\}$

(Note that we can substitute a variable, or a term containing variables for another variable - provided that the variable being substituted for does not appear in the substitution.)

5. When there are multiple possible unifiers, there will be at least one, called a most general unifier (mgu), that has the property that all other unifiers can be derived by applying some substitution AFTER applying the mgu.

Ex: In the above, $\{a/Y, X/Z\}$ is an mgu. If we apply it, and then apply a second substitution b/X , we get the second unifier we showed. The reverse would not be possible.

a) Note that the mgu preserves as much generality as possible for the two formulas. When we use unification as part of resolution, we must apply the substitution not only to the “A” clauses but also to the “B” and “C” clauses. By using the mgu, we leave the maximum flexibility for the resolvent $B \cup C$ to resolve with other clauses.

b) Note that the mgu is not necessarily unique.

For example, $\{a/Y, Z/X\}$ is also an mgu for our example

E. There is an algorithm for finding an mgu, which can be used on two formulas in list form - e.g.

$\text{dog}(\text{rocco})$ and $\text{dog}(X)$ (the literals we are trying to unify)

would be expressed as: (dog rocco) and $(\text{dog } X)$

1. The algorithm returns a (possibly empty) list of substitutions (the mgu) or FAIL if no mgu exists.

PROJECT - Algorithm on page 69

2. Example: Suppose we wish to unify $\text{chases}(X, \text{alexander})$ with $\text{chases}(\text{rocco}, Y)$

a) In list form, we have $(\text{chases } X \text{ alexander})$ and $(\text{chases rocco } Y)$

- b) We call unify with $E1 = (\text{chases } X \text{ alexander})$ and $E2 = (\text{chases rocco } Y)$
- c) Since $E1$ and $E2$ are both lists, we drop to “otherwise”, ultimately generating two recursive calls to unify
 - (1) The first with $E1 = \text{chases}$ and $E2 = \text{chases}$ returns $\{\}$ - no substitutions are needed to make them unify, since they are already identical.
 - (2) The result of applying this to $TE1$ and $TE2$ is to leave the two sublists $(X \text{ alexander})$ and $(\text{rocco } Y)$ unchanged, so the second recursive call attempts to unify $(X \text{ alexander})$ with $(\text{rocco } Y)$
- d) When we call unify on the two sublists $(X \text{ alexander})$ and $(\text{rocco } Y)$, we again drop to “otherwise” and generate two recursive calls.
 - (1) The first call with $E1 = X$ and $E2 = \text{rocco}$ returns $\{ \text{rocco} / X \}$
 - (2) When we apply this to $TE1$ and $TE2$ is to leave the two sublists (alexander) and (Y) unchanged.
- e) We generate a final recursive call for (alexander) and (Y)
 - (1) When we call unify on the two sublists (alexander) and (Y) , we again drop to “otherwise” to generate two recursive calls, since both expressions are (one-element) lists.
 - (2) The first call with $E1 = \text{alexander}$ and $E2 = Y$ returns $\{ \text{alexander} / Y \}$
 - (3) Of course, applying this substitution to $TE1$ and $TE2$ (empty lists) results in o change. We generate a second call on the now-nil tails, which immediately returns $\{\}$.
- f) Finally, as the recursion unwinds we append our various substitutions to produce the mgu
 - $\{ \text{rocco} / X, \text{alexander} / Y \}$

V. Resolution Refutation as A Means of Inference

A. Many AI systems that represent knowledge using predicate calculus use RESOLUTION REFUTATION as an inference technique for deriving new knowledge. The new knowledge is derived by discovering a proof that it must be true if the given knowledge is true - i.e. it is a theorem, given the current knowledge as axioms.

1. Recall that resolution is a sound proof rule that derives a new clause from two known clauses as follows:

Given $A \cup B$ -- where A,B, C are any WFFs
and $\neg A \cup C$
We may infer: $B \cup C$

2. To use resolution, we must put our axioms into the form of a set of clauses

- a) We will define this more formally later, but for now we will note that a clause is a set of literals or'ed together. (A literal is a simple formula like $\text{dog}(X)$ or its negation.) Moreover, all variables in a clause must be universally quantified, so the quantifiers are omitted.

- b) To do this, we take advantage of the equivalence

$$A \rightarrow B \equiv \neg A \cup B$$

3. Resolution refutation operates as follows: to our set of axioms, we add the NEGATION of the theorem we wish to prove. We then use resolution until we get to clauses that resolve down to nil - e.g.

A and $\neg A$

- a) This situation indicates that there is a contradiction in our set of clauses.

- b) If our original axioms were consistent, then the contradiction must have arisen because we introduced the negation of what we want to prove. If it is contradictor to believe the negation of our theorem, then our theorem must be true.

B. An Example:

Given the following axioms

$$(\forall X) (\text{dog}(X) \cap \text{cat}(Y) \supset \text{chases}(X, Y))$$
$$\text{dog}(\text{rocco})$$
$$\text{cat}(\text{alexander})$$

Prove $\text{chases}(\text{rocco}, \text{alexander})$

1. Convert the axioms to clause form

The first is equivalent to the clause

(1) $\neg \text{dog}(X) \cup \neg \text{cat}(Y) \cup \text{chases}(X, Y)$

(note how negating the antecedent turns the and to or by DeMorgan's theorem; and how the universal quantifier is dropped because implicit)

The second and third are already clauses - we will refer to these as (2) and (3) in the proof

(2) $\text{dog}(\text{rocco})$

(3) $\text{cat}(\text{alexander})$

2. Add the negation of the theorem to the database:

(4) $\neg \text{chases}(\text{rocco}, \text{alexander})$

3. Resolve (4) with (1) with unifier $\{ \text{rocco}/X, \text{alexander}/Y \}$

(5) $\neg \text{dog}(\text{rocco}) \cup \neg \text{cat}(\text{alexander})$

4. Resolve (5) with (2)

(6) $\neg \text{cat}(\text{alexander})$

5. Resolve (6) with (3)

NIL - a contradiction

6. Since we have been able to “prove” false, our set of axioms plus negated theorem are contradictory. Since the axioms are assumed to be non-contradictory, the ability to prove false must have arisen from our negated theorem, which must not be true. But if the negated theorem is not true, the original theorem must be true - QED

Note: if we have an initial set of axioms that is contradictory, we can prove anything by resolution refutation!

- C. Resolution refutation was introduced as an automated reasoning technique by Robinson in 1965. Its attraction is that it is much easier to automate than modus ponens. It is, in fact, the basis for the proof mechanism of Prolog (though this is not obvious until one looks closely at what is going on.)

VI. Set of Clause form

- A. Before we can use resolution refutation as a proof technique, we first have to do some preliminary work on our representation. Resolution refutation requires that all our axioms be expressed in a particular form: they must comprise a set of clauses. Likewise, Prolog requires that entries in the database be clauses.

- B. Formal definition of "clause".

1. Preliminary definitions:

- a) A predicate with its argument(s) is called an atomic formula.

Ex: $\text{dog}(\text{rocco})$
 $\text{chases}(X, Y)$

But not: $\neg \text{dog}(\text{rocco})$
 $\text{dog}(X) \cup \text{cat}(Y)$
 $\text{dog}(X) \cap \text{likesChildren}(X)$
 $\text{dog}(X) \rightarrow \text{chasesCats}(X)$
 $(\exists X) \text{dog}(X)$
 $(\forall X) (\text{dog}(X) \rightarrow \text{chasesCats}(X))$

- b) Atomic formulas and negated atomic formulas are called literals.

Ex: $\text{dog}(\text{rocco})$ is a literal because its an atomic formula
 $\neg \text{dog}(\text{rocco})$ is a literal, though it was not an atomic formula

The remaining examples above are not literals. either

2. A WFF consisting of a disjunction of literals is called a clause.

Ex: all of the above literals
 $\text{dog}(X) \cup \text{cat}(Y)$

But neither of the other examples, though

$\text{dog}(X) \rightarrow \text{chasesCats}(X)$

is easily converted to the equivalent clause

$$\neg \text{dog}(X) \cup \text{chasesCats}(X)$$

3. Quantifiers do not occur in clauses. Instead, all of the variables occurring in any clause are considered to be universally quantified. (E.g. the clause $\neg \text{dog}(X) \cup \text{chasesCats}(X)$ is implicitly considered to be $(\forall X) (\neg \text{dog}(X) \cup \text{chasesCats}(X))$ [If an existentially-quantified variable occurs, it is replaced by a Skolem constant or function].
- C. A set of clauses is just that - a set of clauses, which are considered to be implicitly connected by \cap . By taking advantage of the rule of universal specialization, we arrange for the variable names occurring in each clause to be distinct.
- D. The book gives an algorithm for converting any WFF into a set of clauses. (§13.2.2). We will not discuss this. (Any time you are given a resolution proof problem on homework or an exam, the database will be given to you in set of clause form.)

VII. Actually Using Resolution Refutation

- A. After doing all this preliminary work, we are ready to actually do proofs by resolution refutation.
- B. To prove a theorem by resolution refutation, we proceed as follows:
 1. Convert the axioms to a set of clauses.
 2. Negate the theorem, convert the result to a clause, and add it to the set of axioms. This amounts to saying “Assume that our axioms are true and our theorem is false”.
 - Note: This tends to eliminate existential quantifiers. Existential quantifiers tend to appear in theorems more often than in axioms (though this is not an absolute rule by any means.) Negating an existential quantifier turns it into a universal quantifier.
 3. Use resolution repeatedly, adding each new resolvent to the set of axioms, until some resolvent is NIL (false). This amounts to saying “The outcome of our assumption that our theorem is false is a contradiction. Therefore, our theorem must be true.”
 4. This method is called resolution refutation because we prove a theorem true by refuting its negation.

5. It can be shown (though we will not attempt it) that resolution refutation is a complete proof method provided that our axioms are in clause form. That is, any theorem that can be proved from a given set of axioms can be proved by resolution refutation.

C. Resolution refutation can not only be used to prove theorems, but also to answer questions.

1. Example: Earlier we developed a resolution refutation proof of $\text{chases}(\text{rocco}, \text{alexander})$, given the following axioms (in clause form)

- (1) $\neg \text{dog}(X) \cup \neg \text{cat}(Y) \cup \text{chases}(X, Y)$
- (2) $\text{dog}(\text{rocco})$
- (3) $\text{cat}(\text{alexander})$

Suppose, instead, we were trying to answer the question “What animal chases Alexander?”

a) We can formalize the question as

$(\exists X) (\text{chases}(X, \text{alexander}))$

-- where our goal is to prove that such an animal exists in such a way as to find out who it is.

b) Negating and converting to clause form, using the algorithm in the book:

$\neg(\exists X) (\text{chases}(X, \text{alexander}))$

$(\forall X) (\neg \text{chases}(X, \text{alexander}))$

$(\forall Z) (\neg \text{chases}(Z, \text{alexander}))$

$\neg \text{chases}(Z, \text{alexander})$

- reducing the scope of negation

- standardizing variables apart

- dropping universal quantifier

(Call this clause (4))

c) Resolving (4) with axiom (1), with unifier $X/Z, \text{alexander}/Y$:

(5) $\neg \text{dog}(X) \cup \neg \text{cat}(\text{alexander})$

d) Resolving (5) with axiom (3)

(6) $\neg \text{dog}(X)$

e) Resolving (6) with axiom (2), with unifier rocco/X

NIL

f) To answer our original question, we apply the composition of the unifiers we used to the original query (with the standardized apart variable names.)

(1) We used the unifiers X/Z , $alexander/Y$ and $rocco/X$ in that order

(2) This gives us $chases(Z, alexander) \{X/Z, alexander/Y, rocco/X\}$
or
 $chases(rocco, alexander)$

2. We can automate this process by using a strategy called Green's device (named after the logician Cordell Green)

What we do is to augment the original question with a "dummy" term $answer(-- \text{whatever variable(s) we want the value for})$. Then we resolve down to a clause in which only this dummy term appears, and it is our answer. Applying this to the previous example:

a) Augmented goal:

(4) $\neg chases(Z, alexander) \cup answer(Z)$

b) After first resolution:

(5) $\neg dog(X) \cup \neg cat(alexander) \cup answer(X)$

c) After second resolution:

(6) $\neg dog(X) \cup answer(X)$

d) After final resolution

$answer(rocco)$

D. The original process (or actually its equivalent in Prolog) is actually the process used by the demonstration program I showed you at the start of the course. Let me demonstrate it again.

```
Demo: Start prologj
      [myproj3].
      proj3.
      rocco is a dog.
      alexander is a cat.
      dogs chase cats.
      save database to a file, then examine
      assertz(print_translation).
      proj3.
      who chases alexander?
```