CPS331 Lecture: The Predicate Calculus last revised January 26, 2012

Objectives:

1. To introduce propositional calculus
2. To introduce the first order predicate calculus, including the syntax of WFFs
3. To introduce formalization of knowledge using predicate calculus
4. To introduce resolution-refutation theorem proving, including unification and conversion to clause form

Materials:

1. Equivalence of WFF's Handout
2. Projectable of Cawsey Figure 2.2 (semantic net used in KR lecture)
3. Demo interactive_isa.pro
4. Resolution Proof that Jesse is an ancestor of Jesus

I. Introduction

A. One notation system that is widely used in symbolic 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 the programming languages widely used in AI (LISP and Prolog).

   b) In fact, predicate calculus is the formal basis of Prolog. This will become obvious in the a subsequent series of lectures (on Prolog).

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. In the next series of lectures, 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.

II. **Propositional Calculus**

Your book (and many AI books) eases into predicate calculus by way of a less powerful system of notation called the propositional calculus.

A. A propositional calculus formula is composed of atomic propositions, which area simply statements that are either true or false.

Ex: "It is sunny outside"
"It is raining outside".
"It is snowing outside"
"It is precipitating outside"
"It is hot outside"
"It is cold outside"

B. For convenience (and basically to save a lot of writing), we generally refer to an atomic proposition by an upper case letter or a short phrase. So, we might decide to on a scheme like this:

sunny =	"It is sunny outside"
raining =	"It is raining outside".
snowing =	"It is snowing outside"
precipitating =	"It is precipitating outside"
hot =	"It is hot outside"
cold =	"It is cold outside"

C. Atomic propositions are combined with various connectives to form more complex sentences. The connectives used are

^ - and
v - or
¬ - not
→ - implies
↔ - equivalence

1. For example, given the above definitions, sunny ^ hot means “it is sunny outside and it is hot outside”; snowing ^ cold means “it is snowing outside and it is cold outside”.

2. Again, raining v snowing means “it is raining outside or it is snowing outside”.

3. ¬ raining means “it is not raining outside”.

4. snowing → cold means “it is snowing outside implies it is cold outside” or “if it is snowing outside, then it is cold outside”

5. precipitating ↔ (raining v snowing) means “The statements ‘it is precipitating outside’ and ‘it is raining outside or it is snowing outside’ are equivalent. (For simplicity, lets not worry about other forms of precipitation!)

D. The meaning of the various connectives is formally defined by truth tables, which allow us to draw a conclusion about truth of a statement from the truth of its component parts.

1. For example, the connective ^ has the following truth table (where F stands for “false” and T for “true”)

A	B	A ^ B
F	F	F
F	T	F
T	F	F
T	T	T

That is, A ^ B is true just when both A and B are true; it is false when either is false.

2. The connective v has the following truth table:

A	B	A v B
F	F	F
F	T	T
T	F	T
T	T	T

That is, A v B is true just when at least one of A and B is true; note that it is therefore true when both are true. (In formal logic, v is what we call "inclusive or" rather than "exclusive or".)

3. The connective ¬ has the following truth table:

A	¬A
F	T
T	F

That is, ¬A is true just when A is false and vice versa.

4. The connective → has the following truth table:

A	B	A → B
F	F	T
F	T	T
T	F	F
T	T	T

That is, A → B is true so long as B is true whenever A is. A → B is always true if A is false - e.g. the statement moon_is_made_of_green_cheese → george_washington_is_us_president is true!

5. The connective ↔ has the following truth table:

A	B	A ↔ B
F	F	T
F	T	F
T	F	F
T	T	T

That is, A ↔ B is true if A and B are either both true or both false

E. The power of symbolic logic comes from the fact that we can prove statements by formulating them symbolically and then applying the rules of symbolic logic to them.

1. For example, it is the case that

$(A \rightarrow B) \leftrightarrow (\neg A \vee B)$

That is, A implies B is equivalent to not A or B. We can demonstrate this using truth tables, as follows

(FILL in the three rightmost columns as a class exercise)

A	B	(A → B)	¬ A	(¬A v B)
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	T	F	T

Note that the columns for A → B and for ¬ A v B are identical - that is, in all cases A → B and ¬ A v B have the same value - which is what we mean by equivalence.

2. Example - let's develop a proof of the following in a similar way:

   $(A \leftrightarrow B) \leftrightarrow ((A \rightarrow B) \wedge (B \rightarrow A))$

   [Develop in class]

F. There are several proof rules that are often used with propositional calculus statements

1. Modus ponens says: given $A \rightarrow B$, and given A, we can conclude B

   Symbolically; $A \rightarrow B,\ A$

   B

   (Here we call $A \rightarrow B$ and A the premises, and B the conclusion. The rule of inference says that whenever the premises are true, the conclusion is as well. Of course, this is only valid in the real world if our premises are sound to begin with!)

   Example: if we believe "it is snowing outside implies it is cold outside", and we are told "it is snowing outside", it is reasonable to infer that "it is cold outside".

   Symbolically snowing $\rightarrow$ cold, snowing

   ---

   cold

2. Modus tolens says: given $A \rightarrow B$, and given $\neg B$, we can conclude $\neg A$

   Symbolically; $A \rightarrow B,\ \neg B$

   $\neg A$

   Example: if we believe "it is snowing outside implies it is cold outside", and we are told "it not cold outside", it is reasonable to infer that "it is not snowing outside".

Symbolically snowing → cold, ¬ not cold

¬ snowing

3. Note well that it is <u>not</u> sound to say "given A → B, and given B, we can conclude A". This sort of reasoning is called abduction. While it sometimes the case, it is not always the case.

   Example: if we believe "it is snowing outside implies it is cold outside", and we are told "it cold outside", it is <u>not necessarily</u> the case that "it is snowing outside".

4. Another sound rule of inference that happens to be particularly easy to automate in AI systems is called <u>resolution.</u> Like modus ponens and modus tolens (but unlike abduction) it is a sound rule of inference. Given A v B and ¬B v C, we can infer A v C

   Symbolically: A v B, ¬B v C

   A v C

   (We refer to A v C as the "resolvent" of the two clauses A v B and ¬B v C)

   Demonstration using truth tables - note that we do not care about rows where it is not the case that both premises are true

A	B	C	A v B	¬B v C	Both true	A v C
F	F	F	F	T	-	-
F	F	T	F	T	-	-
F	T	F	T	F	-	-
F	T	T	T	T	÷	T
T	F	F	T	T	÷	T
T	F	T	T	T	÷	T
T	T	F	T	F	-	-
T	T	T	T	T	÷	T

Example: If we believe that it is precipitating outside or it is sunny outside, and we believe if it is sunny outside then it is hot outside, then we can conclude it is precipitating outside or it is hot outside. (Again, I'm not making claims about the New England weather - just illustrating formal logic, so let's assume these premises are valid!)

Symbolically; Given: precipitating v sunny, sunny → hot

Which is equivalent to: precipitating v sunny, ¬ sunny v hot

---

precipitating v hot

III. **Introduction to Predicate Calculus**

A. The first order predicate calculus is a formal language for expressing the content of propositions. Like any formal language, it has a well-defined syntax.

B. 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 a few years ago. Constants that might appear would include:

   rocco (my dog at the time I wrote this lecture)
   alexander (who was then 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 read later in the course.

3. Variables

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

      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 same connectives as were used with propositional calculus. That is, an atomic predicate calculus formula is simply a formalized way of writing a proposition, as opposed to writing it in English. (Using this formal notion facilitates automated reasoning.)

b) By way of review, here are the connectives again. I've also listed alternate forms for most which one sometimes sees:

(1) ^ (alternate ∩)- and. This operation is also called conjunction.

(2) v (alternate ∪) - or. This operation is also called disjunction.

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

(4) → (alternate ⊃) 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: dog(X) → barks(X)

- dog(X) is the antecedent
- barks(X) is the consequent

(5) ↔ (alternate ≡) equivalence. (Recall that two formulas are equivalent if the first is true just when the second is true and vice versa)

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

A	B	A ^ B	A v B	¬ A	A → B	A ↔ B
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 p is true for all possible values of the variable. For example, we might formulate "dogs chase cats" rule as follows.

dog(X) ^ cat(Y) → 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 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 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 - ∀ or ∃.

(1) The universal quantifier ∀ (read "for all") indicates that any value may be substituted for the indicated variable.

(2) The existential quantifier ∃ (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:

(∀ X) (dog(X) → 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:

(∃x) (dog(X) → 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 barking dogs in the world, provided there was at least one non-dog!)

(b) On the other hand, if we had the WFF dog(X), we clearly want the existential quantifier:

(∃X) (dog(X))

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

(If we used the universal quantifier

(∀ X) (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:

(∀ X) (∀ Y) (dog(X) ^ cat(Y) → 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:

(∃Y) cat(Y) ^ ((∀ X) (dog(X) → chases(Y, X)))

(Note also the rearrangement of ^ and →)

"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) v (∀ X) (q(a,X)) v (∃ X) (r(a, X)), the scope of the (∀ X) quantifier is q(a, X), and the variable "X" that appears in r(a, X) is <u>not</u> necessarily the same as the variable "X" that appears in q (a, X)

f) It turns out to be the case 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: (∀ X) (∀ Y) (dog(X) ^ cat(Y) → chases(X, Y))

is often written as simply

dog(X) ^ cat(Y) → chases(X, Y)

with (∀ X) (∀ 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:

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

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:

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

(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:

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

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.

C. 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 $\leftrightarrow$ W2.

   For example, we have already noted that:

   $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$

2. DISTRIBUTE, GO OVER HANDOUT WITH RULES OF EQUIVALENCE

   Especially note properties of quantifiers, including effect of negation

IV. **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")

   $(\forall X)\ (dog(X) \rightarrow barks(X))$

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

   Ex: $(\forall X)\ (createdBy(X, god))$ might seem to be an exception, but this says that God created even Himself, so even this WFF is not really correct!

4. Where we have a qualified generic noun, we have two basic options:

   Ex: "The dog who lives with Garfield does not chase cats" ("There is a dog who lives with Garfield and it is not the case that this dog chases cats")

   a) We can use an existentially-quantified variable

      $(\exists X)\ (dog(X) \wedge livesWith(X, garfield) \wedge \neg\ chasesCats(X))$

   b) We can create a special sort of name - called a Skolem constant in honor of its inventor - to stand for the otherwise unknown name of "the dog", since we do not have the dog's name - only the statement that it is the dog who lives with Garfield. We can invent a name like dog1, and formalize as:

dog(dog1) ^ livesWith(dog1, garfield) ^ ¬ chasesCats(dog1)

That is, sentences of the form "Some _____" generally translate into a WFF with one (or more) existentially-quantified variables, or Skolem constant(s).

As a general rule, existentially quantified variables do not contain a top-level implication in their scope - otherwise, we are making a vacuous statement

Ex: (∃X) (god(X) → createdWorld(X)) does not say that God created the world - it is also satisfied by any instantiation of X where the antecedent is false (e.g. it is true for X = satan, since Satan is not God!) The correct form of the statement would be (∃X) (god(X) ^ createdWorld(X))

C. Some examples from the pets world. [have various ones in class do - try to find more than one way of formalizing each]

1. "Rocco is a dog".	dog(rocco) or species(rocco, dog)
2. "Garfield is orange"	orange(garfield) or color(garfield, orange)
3. "Rocco chases cats"	chasesCats(rocco) or (∀ X) (Cat(X) → chases(rocco, X)
4. "Cats eat mice"	(∀ X) (cat(X) → eatsMice(X)) or (∀ X) (∀ Y) (cat(X) ^ mouse(Y)→ eats(X, Y))
5. "The cat who lives at #11 is orange"	address(cat1, 11) ^ color(cat1, orange) or (∃X)(cat(X)^address(X, 11)^color(X.orange))

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

$(\forall X)\ (cat(X) \wedge address(X, 11) \rightarrow color(X, orange))$

- 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.)

$(\forall X)\ (cat(X) \wedge address(X, 11) \wedge color(X, orange)$

- 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"

   $(\forall X)\ [mailman(X) \rightarrow (\exists Y)\ \{ dog(Y) \wedge hasChased(Y,X) \}]$

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

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

E. Do semantic net example from Knowledge Representation lecture

1. PROJECT Cawsey Figure 2.2

2. Class exercise: Translate some of the link in net into predicate calculus.

F. 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.

V. **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

   $(\exists X)$ (chases(X, 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. Recall the rules of inference we talked about in conjunction with propositional calculus.

1. Modus ponens: $\dfrac{A \rightarrow B,\ A}{B}$

2. Modus tolens: $\dfrac{A \rightarrow B,\ \neg B}{\neg A}$

3. Resolution $\dfrac{A \vee B,\ \neg B \vee C}{A \vee C}$

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

(1) For example, if we resolve:

A v B
¬B

we get A since "C" is NIL

(Formally, we are resolving (A v B) with (¬B v false), which leads to (A v false), which implies A)

(2) If we resolve A1 v A2 v A3 v B ...
with ¬B v C1 v C2 v C3 ...,
we get A1 v A2 v A3 ... v C1 v C2 v C3 ...

(We do this by using the associative property of disjunction, treating A1 v A2 v A3... as "A" and C1 v C2 v C3... as "C".)

b) If we resolve B with ¬B - ie. both "A" 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.

VI. **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 (∀ X)w(X) -- where w is some WFF containing X

We may infer w(a) -- where a is any constant

Ex: From (∀ X) (dog(X) → chasesCats(X)
We infer dog(rocco) → chasesCats(rocco)

(We also infer dog(bjork) → chasesCats(bjork) - which is a true statement since I am not a dog!)

2. 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 (∀ X) (dog(X) → chasesCats(X)
and dog(rocco)

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

¬dog(rocco) v chasesCats(rocco) with dog(rocco) to conclude chasesCats(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 = rocco

(Note that the substitution is applied to the entirety of the clause, not just to the 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 ...} - 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} ↔ 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 "B" clauses but also to the "A" and "C" clauses. By using the mgu, we leave the maximum flexibility for the resolvent A v 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 we will not discuss.

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

a) A clause is a set of literals or'ed together, where a literal is a simple formula like dog(X) or its negation.) Moreover, all variables in a clause must be universally quantified, so the quantifiers are omitted.

To do this, we take advantage of the equivalence

$A \rightarrow B \leftrightarrow \neg A \vee B$

b) In set of clause form, we also standardize variables apart to that each clause uses different variable names.

Recall that the variable name bound by a given quantifier can be changed to any other variable name as long as all occurrences of the name in the given scope are changed.

c) There is a general algorithm for doing this, which we won't discuss.

2. 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.

B and ¬B

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 contradiction to believe the negation of our theorem, then our theorem must be true.

B. An Example:

Given the following axioms

$(\forall X)(\forall Y)\ (dog(X) \wedge cat(Y) \rightarrow chases(X, Y))$
dog(rocco)
cat(alexander)

Prove chases(rocco, alexander)

1. Convert the axioms to clause form

   The first is equivalent to the clause

   (1) ¬ dog(X) v ¬ cat(Y) v 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) dog(rocco)
   (3) cat(alexander)

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

   (4) ¬ chases(rocco, alexander)

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

   (5) ¬ dog(rocco) v ¬ cat(alexander)

4. Resolve (5) with (2)

   (6) ¬ cat(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 <u>not</u> true, the original theorem must be true - QED

   Note: if we have an initial set of axioms that is contradictory, we can prove <u>anything</u> 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.)

VIII. **Actually Using Resolution Refutation**

A. 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 he 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.

B. An example:

1. Given the following statements:

   The father of someone or the mother of someone is an ancestor of that person.
   An ancestor of someone's ancestor is also an ancestor of that person.
   Jesse is the father of David.
   David is an ancestor of Mary.
   Mary is the mother of Jesus.

   Prove: Jesse is an ancestor of Jesus

2. Formalization (with conversion to clause form given for the rules):

   a) (∀X)(∀Y) [(father(X, Y) v mother(X, Y)) → ancestor(X, Y)]

      ¬father(X1, Y1) v ancestor(X1, Y1) [1]
      ¬mother(X2, Y2) v ancestor(X2, Y2) [2]

   b) ∀R)(∀S)(∀T) [(ancestor(R, S) ^ ancestor(S, T)) → ancestor (R, T)]

      ¬ancestor(R, S) v ¬ancestor(S, T) v ancestor(R, T) [3]

   c) father(jesse, david)[4]
   d) ancestor(david, mary) [5]
   e) mother(mary, jesus) [6]

      Prove: ancestor(jesse, jesus)

      PROJECT

3. Proof

a) Negated theorem: ¬ancestor(jesse, jesus) [7]

b) Resolving [7] with [3], using { jesse/R, jesus/T }
¬ancestor(jesse, S) v ¬ancestor(S, jesus) [8]

c) Resolving this with [1], using { jesse/X1, S/Y1 }
¬father(jesse, S) v ¬ancestor(S, jesus) [9]

d) Resolving this with [4], using { david/S }
¬ancestor(david, jesus) [10]

e) Resolving this with [3], using { david/R, jesus/T }
¬ancestor(david, S) v ¬ancestor(S, jesus) [11]

f) Resolving this with [5] , using { mary/S }
¬ancestor(mary, jesus) [12]

g) Resolving this with [2], using { mary/X2, jesus/Y2 }
¬mother(mary, jesus) [13]

h) Resolving this with [6]
NIL - QED

PROJECT

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 chases(rocco, alexander), given the following axioms (in clause form)

   (1) ¬ dog(X) v ¬ cat(Y) v chases(X, Y)
   (2) dog(rocco)
   (3) cat(alexander)

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

   a) We can formalize the question as

      (∃X) (chases(X, 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 rules of equivalence for WFFs)

      ¬(∃X) (chases(X, alexander)) becomes
      (∀ X) (¬chases(X, alexander)) becomes
      (∀ Z) (¬chases(Z, alexander)) becomes
      ¬chases(Z, alexander) (universal quantifier implicit)

      (Call this clause (4))

   c) Resolving (4) with axiom (1), with unifier X/Z, alexander/Y:

      (5) ¬dog(X) v ¬cat(alexander)

d) Resolving (5) with axiom (3)

(6) ¬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(-- 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) ¬chases(Z, alexander) v answer(Z)

b) After first resolution:

(5) ¬dog(X) v ¬cat(alexander) » answer(X)

c) After second resolution:

(6) ¬dog(X) v answer(X)

d) After final resolution

answer(rocco)

3. The original process (or actually its equivalent in Prolog) can be used to produce a question answering system

Demo: Start Prolog from command line.
Consult interactive_isa
`interactive_isa.`
rocco is a dog.
alexander is a cat.
dogs chase cats.
quit, saving database to a file, then examine
`assertz(print_translation).`
`interactive_isa.`
who chases alexander?