Knowledge Representation

Knowledge Representation

- As we pointed out, we need to be able to construct symbolic representations in order to solve problems
- Need to represent knowledge of the domain (jugs, pouring)
- Need to represent knowledge of the specific problem within the domain (size of jugs, current content of each)
- This has to be represented in a form that a search process can manipulate

General Needs for Knowledge Representation in AI

(Text, pp. 34-41)

- Handle qualitative knowledge
- Allow new knowledge to be inferred from facts and rules
- Allow representation of general principles as well as specific situations
- Capture complex semantic meaning
- Allow for meta-level reasoning
  - i.e. reasoning about the knowledge itself

Mathematical Logic

- A well understood formal representation
- Few surprises because of its mathematical nature (that is, we know how any two pieces can be put together, we won't find out later that something can't work)
- There are many flavours and varieties of logic, some specialized for certain types of reasoning, others general.
  - You're already used to basic Boolean logic, for example, that's used in logic circuit design

Propositional Calculus

- Propositional calculus is one of the most basic forms of mathematical logic
- We define symbols: P, Q, Pour5Into2, etc.
- We have basic connectives for putting symbols together: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $=$
- We connect symbols with connectives into propositions: statements about the world that may be true or false
- We have rules for connecting symbols:

Propositional Calculus Sentences

- Every symbol and truth symbol (True, False) is a sentence (e.g. A)
- The negation of a sentence is a sentence ($\neg$ A)
- The conjunction of two sentences is a sentence (A $\wedge$ B)
- The disjunction of two sentences is a sentence (A $\vee$ B)
- The implication of one sentence for another is a sentence (A $\Rightarrow$ B)
- The equivalence of two sentence is a sentence (A=B)

Building Propositions

- I can build complex propositions using these: ((P $\wedge$ Q) $\Rightarrow$ R) = $\neg$ P $\vee$ Q $\vee$ R, and so on
- These rules are the syntax of propositional calculus. Every logic also has a semantics: the meaning associated with its parts
- Every propositional symbol corresponds to a statement about the world
- Each operator has a truth table that describes the meaning of using the operator (see p. 50)
- True and False are atomic

Logical Operators

- From the semantics of the basic connectives, we can prove more complex equivalences and use them
- For example, the truth tables of and, not, and or allow us to show the equivalence of $\neg$(P $\wedge$ Q) and $\neg$P $\vee$ $\neg$Q (de Morgan’s law)
- These equivalences let us manipulate the symbolic expressions into useful forms
- Newell and Simon’s Logic Theorist used this kind of thing to prove many things in the *Principia Mathematica*

What's Missing here?

- There are many useful things in the propositional calculus, but it's actually fairly weak in terms of what we can say about the real world
- For example, a car on my street could be represented as CARONMYSTREET (or more legibly CAR-ON-MY-STREET, but that's one atomic concept – not possible to use the pieces in new propositions
- Also missing is the idea of making very general statements: all birds have wings, for example. We can't say "for everything"

Making General Statements

- What are we doing when we say something like "ALL birds have wings"?
- *Quantifying*
- We're saying something holds true for all objects of a certain type
- Propositional calculus doesn't let us quantify over statements, so we need to move up a level to a logic that will!
- FIRST-order logic
- FOL, also known as First order PREDICATE calculus or First Order Predicate Logic

First-Order Logic

- First-order predicate logic (FOL) was one of the first languages to be used in AI because automatic theorem proving was a major research topic in AI
- Allows quantification over sentences, and also allows us to relate symbols much more easily than propositional calculus (has predicates!)
- First-order logic can be used to represent knowledge for a wide variety of AI systems
- FOL does have limitations – there are specialized versions for specialized tasks to deal with these (temporal logic, for example)

Objects

- Object constants are symbols that represent individual objects

  bill   sally

  Also base truth values **true** and **false**
- Predicates are used to define facts about objects and relationships between/among objects

  male(bill)

  female(sally)

  parent(bill, sally)
- all of these begin with lowercase letters

Connectives

$\wedge$ (AND)

$\vee$ (OR [really, inclusive or])

~ $\neg$ (NOT)

$\rightarrow$ (IMPLICATION)

$\leftrightarrow$ (EQUIVALENCE)

- Like Propositional logic, FOL has connectives that can only be applied to predicates
- But not to objects – idea of logical connectives between objects is weird anyway!
- You're used to the formal use of all but one of these connectives

Implication

- Implication tends to be difficult for people who have not seen it formally to grasp right away
- As would appear by the name, the implication operator is used to imply something (A $\rightarrow$ B)
- You're used to an informal use of this – more or less "if you got A, then you got B".
  - e.g. weather(rainy) $\rightarrow$ wet(sidewalk)
- Formally, (mainly because of the fact that we can use a single logical statement in a number of ways to derive new ones), we have to go a little deeper than this

Implication, more formally

- If the left side (the premise) of the implication operator is false, the implication sentence is true *regardless* of whether the right side is true or false
- Think about what that means…
  - Say it ***isn't*** raining. Does that mean the sidewalk isn't wet?

Not Necessarily…

SO…

- The implication of raining making the sidewalk wet is STILL true, in and of itself, despite the fact that the sidewalk might have gotten wet otherwise.
- so, we can't actually say anything about whether it's raining or not if the sidewalk is wet.
- i.e. if we have weather(rainy) $\rightarrow$ wet(sidewalk), that DOES NOT mean we can assume or use wet(sidewalk) $\rightarrow$ weather(rainy) (!!!!)
- Having both directions work is equivalence – this is one of the most common mistakes people make when working with logical systems

Looking at it from the other side…

- Look at the right side. Suppose sw is wet?
  - well it **might** be raining, or something else. Either way the *implication* is still true.
- But suppose the right side is false – i.e. the sidewalk isn't wet…
- Well by our same reasoning, if it were raining, the sidewalk would be wet, so it better not be raining for the implication to still be true!
- The only time that the result of an implication is false is if the **left side is true and the right side is false** – i.e. it directly contradicts what we're trying to imply!

What does this lead to?

- We just said that if the sidewalk is dry, it can't possibly be raining or our implication would be false
- This lets us to derive one more principle. If we have an implication like:
- if it's rainy, then the sidewalk is wet
  - weather(rainy) $\rightarrow$ wet(sidewalk)
- …then We CAN say that if the sidewalk isn't wet, then it isn't rainy!
  - ~wet(sidewalk) $\rightarrow$ ~ weather(rainy)

Implication's Truth Table

A	**B**	**A $\rightarrow$ B**
T	T	T
F	T	T
F	F	T
T	F	F

- Looking at the truth table alone, this looks odd
- Thinking of an example will help you remember it!
- Logically, this is the same as saying **~A $\vee$ B**

Summing All This Up

"The fact that some geniuses were laughed at does not imply that all who are laughed at are geniuses. They laughed at Columbus, they laughed at Fulton, they laughed at the Wright brothers. But they also laughed at Bozo the Clown."

– Carl Sagan

Sentences

- An atomic sentence is a predicate applied to one or more objects or predicates. Think of a sentence as making a statement about objects

  male(bill)

  friends(father(david),father(andrew))
- A compound sentence is a set of atomic sentences joined (in a valid manner) by connectives

  female(sally) ^ jobclass(sally,construction)
- A valid sentence is also called a well-formed formula (wff)

Axioms and Logical Systems

- An axiom is a well-formed-formula that is true within a logical system
  - i.e. more than just syntactically valid!
- A logical system is a collection of axioms that describe a real or hypothetical world or situation
- When representing things logically, we aim for CONSISTENCY – no contradictions in a set of sentences
  - there's lots of ways we can show inconsistency – like implications that can't be true, for example

Kicking it Up a Notch

- The vocabulary developed so far permits the definition of facts about objects and relationships between/among objects:

  bird(tweety)

  canary(tweety)

  has-wings(tweety)

  can-fly(tweety)

First-Order Logic

- Having to make each fact explicit is very time consuming; we would prefer to represent only the basic information and have the system deduce the missing information
- First-order logic (FOL) permits general statements to be made using quantifiers and variables

Variables

- Well, you're already used to these from formal programming languages
- Variables let us substitute for a specific object in place, consistently, in every spot we use the variable in a sentence
- These allow us to make logical statements that can be used to pertain to numbers of items
- Variables appear in Capital Letters
- To be able to use them, we need to talk about Quantifiers, though, which are operators on variables

Quantifiers

- We can quantify over variables to indicate the objects that are intended to participate as potential contents or bindings for the variable
- In fact, first order logic gets its name because it's allowed to do this
- We can't quantify over predicates or functions
- If we could, we'd be working with a higher-order logic
- There are two basic Quantifiers: **Existential** and **Universal**

Universal Quantifier

- The universal quantifier states that a formula *wff* is true for all object constants in the system

  $\forall$ X: wff(X)
- For any object constant that is substituted for X, wff(X) is true
- The variable can be replaced by any object constant and the formula wff will be true
- The universal quantifier permits us to make statements about sets of objects

Universal Quantifier

- Unification is very powerful
- For example, saying $\forall$X: has-wings(X) means?
- EVERY entity in the logical system has wings
- Because it makes such sweeping generalities, we most often use it with implication, to pare down the number of situations in which it can be applied…

Universal Quantifier

- $\forall$X: bird(X) $\rightarrow$ has-wings(X)
- This statement states that for any object constant, X, if X is a bird, then X has wings
- Remember the implication is still true whether X is a bird or not!
- If X is not a bird, then the statement itself is still true – it just doesn't indicate whether or not X has wings

Universal Quantifier

- The truth of the statement can be tested by examining all object constants in the system:
  - by definition, the statement is true for all constant symbols that are not birds because A$\rightarrow$B is always true if A is false
  - if the statement is true of all birds in the logical system, then the statement is true in general
- The statement is true even if there are **no** birds in the system!

Universal Quantifier

$\forall$ X: bird(X)

- Similarly, while this statement says that all objects X in the system are birds, it still does not guarantee that there are actually are any objects at all in the system
- If there are objects, though, they're all birds!
- We've seen the use of implication to narrow the applicability of a quantifier
  - e.g. $\forall$X: bird(X) $\rightarrow$ has-wings(X)
- Suppose instead I used:
  - $\forall$X: bird(X) $\wedge$ has-wings(X)

Universal Quantifier

- $\forall$ X: bird(X) $\wedge$ has-wings(X)
- This statement states that all objects in the system are birds and that all objects in the system have wings
- i.e. more sweeping generalizations – obviously very different from
  - $\forall$X: bird(X) $\rightarrow$ has-wings(X)
- What statement could we add to our implication to make them the same?
- $\forall$X: bird(X)

Existential Quantification

- The other major quantifier
- The existential quantifier states that the formula *wff* is true for **at least** one object constant in the system
  - $\exists$ X: wff(X)
- There exists some object constant that can be bound to X such that wff(X) is true

Existential Quantifier

- The statement does not identify which object (or **objects**) satisfy the formula, nor how many
- In one sense it's more powerful than the universal quantifier – since it states there is at least one, there MUST be one
- Recall that the universal quantifier applies to all objects that fit a variable, but doesn't guarantee there actually are any such objects
- In another sense the existential quantifier is limited - there may be more than one object that satisfies the formula but we can't indicate this fact

Existential Quantifier

- The existential quantifier is often used with the AND connective, to state that there are objects with certain features
  - $\exists$ X: bird(X) $\wedge$ can-fly(X)
- This statement states that at least one bird that can fly
- The statement is false only if all birds in the logical system can not fly or if there are no birds in the system
- The statement does not identify which bird is able to fly (most common source of errors!)

Existential Quantifier

- Consider: $\exists$ X: bird(X)
- This statement states that at least one object in the system is a bird
- Now add: $\exists$ X: can-fly(X)
- This statement states that at least one object in the system can fly
- Are these two statements the same as the conjunction of the two?
- NO - because there's nothing to say that the objects satisfying each predicate are the SAME ones

Models

- Our logical systems are models of worlds in which we are interested
- Most models are based on some aspect of the real world (normally referred to as a domain)
  - a model could be of an imaginary world as well
- We can't model everything, so a model is a subset of all the statements that could ever be made about some domain
  - Most models are significant simplifications
- The only restriction is that the model not contain any contradictions

Models

- All sentences in a model must be true
- A portion of any sentence could be false, but the entire sentence must be true:

  $\forall$X: (elephant(X)^colour(X,pink)) $\vee$ person(X)
- If we assume that pink elephants do not exist, then person(X) must always be true for the disjunction to be true
- If a model contains any sentence that is not true, then the conclusions generated will not necessarily be correct

Creating a Model

- The first decision is what what level we're going to use for our representation, because we can represent the same concepts in many ways…
- Representation using only propositions

  robin-fred   bird-fred   red-fred
- Representation using predicates

  robin(fred) bird(fred)

  red(fred)
- Representation using quantified predicates

  robin(fred)

  $\forall$ X: robin(X) $\rightarrow$ bird(X)

  $\forall$ X: robin(X) $\rightarrow$ red(X)

Predicate Selection

- Representation using more generic predicates

  robin(fred)

  $\forall$X: robin(X)$\rightarrow$colour(X,red)

  – Can now determine the colour of an object
- Even more generic:

  member(fred, robin)

  $\forall$ X: member(X, robin) $\rightarrow$ member(X, bird)

  $\forall$ X: member(X, robin) $\rightarrow$ attribute(X, colour, red)
- Can determine the names of each class, attributes of an object

Rules of Inference

- A WFF that can be generated from the axioms in a logical system is a THEOREM
- FOL has numerous rules of inference that permit us to derive new theorems using existing axioms…for example:
- Modus Ponens: a, a$\rightarrow$b, $\therefore$b
  – if a and a$\rightarrow$b are axioms, b is an axiom too.
- Modus Tollens:~b, a$\rightarrow$b, $\therefore$~a
- Universal Instantiation: a, $\forall$ X:P(X), $\therefore$P(a)
- Hypothetical Syllogism: a$\rightarrow$b, b$\rightarrow$c, $\therefore$a$\rightarrow$c

Rules of Equivalence

- As alluded to earlier, we also have many equivalence rules that will allow us to change the form of an axiom:

~(A^B) is equivalent to ~Av~B

~(AvB) is equivalent to ~A^~B

A$\rightarrow$B is equivalent to ~AvB (see truth table!)

$\forall$ X:P(X) is equivalent to ~ $\exists$X:~P(X)

$\exists$X:P(X) is equivalent to ~$\forall$ X:~P(X)

Putting all this together

- To perform formal reasoning about a problem, we need to translate it to FOL, then use rules of inference to attempt to prove our goal (using rules of equivalence to manipulate statements into forms that are convenient for us to work with as we go)
- Let's consider an example. Consider the following collection of statements…

Example

- Marcus was a person.
- Marcus was a Pompeian.
- All Pompeians were Romans.
- Caesar was a ruler.
- All romans were either loyal to Caesar or hated him.
- Everyone is loyal to someone.
- People only try to assassinate rulers they are not loyal to.
- Marcus tried to assassinate Caesar.
  - Now, put these into FOL

Example

- Marcus was a person.
  - person(marcus) (1)
- Marcus was a Pompeian.
  - pompeian(marcus) (2)
- All Pompeians were Romans.
  - $\forall$ X: pompeian(X) $\rightarrow$ roman(X) (3)
- Caesar was a ruler.
  - ruler(caesar) (4)

Example

- All romans were either loyal to Caesar or hated him.
  - $\forall$ X: roman(X) $\rightarrow$ loyal(X,caesar) v hate(X,caesar) (5)
  - I just picked the hate(X,Y) so we could represent people hating others in general. what is this assuming?
  - Inclusive or! if we wanted exclusive or, we'd say:
  - $\forall$ X: roman(X) $\rightarrow$ (loyal(X,Caesar) v hate(X,Caesar)) ^ ~ (loyal(X,Caesar) ^ hate(X,Caesar)).
- Everyone is loyal to someone.
  - $\forall$ X: $\exists$ Y: loyal(X,Y) (6)
  - really everybody is loyal to at least one person

Example

- People only try to assassinate rulers they are not loyal to.
  - $\forall$ X,Y: person(X)^ruler(Y)^ try-assassin(X,Y) $\rightarrow$ ~loyal(X,Y) (7)
- Marcus tried to assassinate Caesar.
  - try-assassin(marcus,caesar) (8)
- Now, attempt to prove ~loyal(marcus,caesar) (9)
- Pretty easy…

Proving…

person(marcus) (1)
pompeian(marcus) (2)
∀ X: pompeian(X) → roman(X) (3)
ruler(caesar) (4)
∀ X: roman(X) → loyal(X,caesar) v hate(X,caesar) (5)
∀ X: ∃ Y: loyal(X,Y) (6)
∀ X,Y: person(X)^ruler(Y)^ try-assassin(X,Y) →
    ~loyal(X,Y) (7)
try-assassin(marcus,caesar) (8)

- use universal instantiation on 7,1,4, and 8, then modus ponens to go directly to our conclusion
  - to use modus ponens with multiple conditions in the implication, need another inference rule: and introduction A, B, therefore A^B

Complications

- Here it was fairly obvious…suppose we started from the beginning (which we'd have to do, mechanically)?
- Modus ponens & universal instantiation (2&3) gives roman(marcus) (10)
- Modus ponens & Universal instantiation (10 & 5) gives loyal(marcus,caesar) v hate(marcus,caesar)
- ends up being a fairly dead path. How many of these are there?

Complications

- LOTS – there are literally dozens of logical operations we could perform on any state to get new ones. the search space is ENORMOUS
- there was a nice obvious solution if we started at the right spot…but there is no "right spot" in general…we just have to be systematic.
- Because of this it seems that FOL so far seems pretty computationally infeasible!

In General

- What can we do?
  - decrease the number of potential states?
  - decrease the number of operators!!!!
  - Chapter 13.2!

Resolution

- There are LOADS of potential operators
- However, they are all beautifully subsumed by ONE single operator
- meaning that in any state there's only one operator we need to use
- There is in fact such an operator. It's called the *Resolution Rule of Inference*
- It's possible to convert pure FOL into a form that allows this single operator to use it
- We call the restricted form of FOL Conjunctive Normal Form or Clause Form

Conjunctive Normal Form

- aka clause form.
- As you'd think, based on having a bunch of conjunctions
- CNF allows us to get rid of most things more complex than a conjunction
- There's a few steps to take a standard FOL statement through to clause form.
- We'll go thru these one by one, then do an example

1 – Toss Implications

- To get to CNF, we first get rid of implications
- Remember that A $\rightarrow$ B is the same as ~A v B, so this is easy
- A and B might be complicated expressions at this point, but that's fine, the other steps will simplify things

2 – Move Negations In

- We then move negations in as far as possible
- often with de Morgan's law
  - e.g. ~(A^B) becomes ~A v ~B
- If quantifiers are negated there's an equivalence rule to move these in
  - e.g. ~ $\exists$X:P(X) is equivalent to $\forall$ X:~P(X)
- remove any ~~'s that might crop up
  - Keep doing this until the ~'s are on atoms

3 – Standardize Variables

- we then standardize variables so that every quantifier uses a unique variable – then we don't have to worry about variable scope in a statement
- Doesn't change the truth value of the wff, since the variables are just arbitrary names
- e.g. ∀ X:t(X) ^ ∀ X:k(X) becomes
  ∀ X:t(X) ^ ∀ Y:k(Y)
- the scope of each of the ∀ 's is separate, so we're just changing names.
- It looks like we're not doing much, but it's really for the next step:

4 - Move all Quantifiers Left

- Move all quantifiers to the left side without changing their order. Since each quantifier's scope is the entire statement, this has no effect on the truth of the wff.
- So ∀ X:t(X) ^ ∀ Y:k(Y) becomes
  ∀ X,Y:t(X) ^ k(Y)
- Now we need to eliminate these quantifiers
- How do we eliminate an existential quantifier? What does the existential quantifier say?
- It says there's some object that makes this true…

5 – Eliminate Existential Quantifiers

- So we just pick one! Not one we know about, but we make up an object that represents the "at least one" that this is true about
- Obviously its vital that this not be some object we're already reasoning about, or we'd make the unreasonable assumption that it's a specific useful object instead of a random one
- So for example, ∃X:bird(X) could become bird(fred), where fred is the symbol we picked

5 – Eliminate Existential Quantifiers

- fred is a Skolem constant: the unused value picked to satisfy the existential quantifier
  - really we use a skolem function without an argument to generate this, but it's simpler to think of it as an object
- Since we haven't used it in reasoning thus far, it isn't making the assumption that it's some specific object we already know about
- But we can still reason about it in future – i.e. make inferences involving bird(fred), because we know there is such an object. We're just calling that object fred!

6 – Eliminate Universal Quantifiers

- Now there's just Universal Quantifiers left…
- So we can just remove them and assume that ALL variables are now universally quantified. We can do this because we've eliminated all the existentials
  - those of you who know some Prolog already should be clued in here…
- Now we just have a mixture of and's, or's, and not's…

7 – Get a Conjunction of Disjuncts

- You can use association and distribution to convert the statement into a number of or components (disjuncts) connected together with ands (a conjunction)
- distribution: i.e. a v (b ^ c) = (a v b) ^ (a v c)
- repetition of this may be required:
- (winter ^wearboots) v (summer ^wearsandals)
- = [winter v (summer ^ wearsandals)] ^ [wearboots v (summer ^ wearsandals)] *{now repeat}*
- (winter v summer) ^ (winter v wearsandals) ^ (wearboots v summer) ^ (wearboots v wearsandals)

8. Split conjuncts into separate clauses

- i.e. a^b is the same as two statements 1) a; 2) b…
- Easily done…just rewriting

- just one more! – we have variables in each clause potentially, so we need to…

9. Standardize Variables Apart

- i.e. rename the variables so no two clauses make use of the same variable
- This sounds like it doesn't matter (because $\forall$X:p(X) ^ q(X) = $\forall$X:p(X) ^ $\forall$X:q(X) so separating is no big deal
- Turns out though that when we use resolution, we will sometimes need to pick an instance and bind a variable to it (e.g. "*ok if p applies to all X, it applies to bob here…*") and we need to make sure when we do that we aren't binding a variable in **other** clauses too and restricting what we can reason about…

NOW

- we have a whole bunch of disjunctions, some using variables, some using skolem constants, some using regular constants (I.e. domain objects that were there in the first place)
- The system will generally have many more, but much simpler statements than before
- but the SYSTEM has the same **logical meaning** that it did before. It looks very different, but it's just in a form that is more amenable to mechanized reasoning.
- Now let's talk about Resolution!

Resolution

- A **very** simple iterative process
- Each step involves two parent clauses, that are resolved to yield a new clause that is inferred from them
- New clause represents ways in which two parent clauses interact with each other
- e.g. say we have

winter v summer
~winter v cold

Resolution

winter v summer
~winter v cold

- Both must be true, but only one of winter or ~winter *ever could be* – if winter is true, cold must be for the 2nd sentence to still be true
- Similarly if ~winter is true, summer must be true for the first sentence to still be true
- so what do we really have here?
  - summer v cold!
  - congratulations, you just derived the resolution rule of inference! Now you see why we want clause form!

Resolving Single Clauses

- Suppose we have winter v summer, and ~winter
- These are still resolveable - ~winter is exactly the same as ~winter v false. See why?
- we usually write ~winter v nil though for clarity
- the result of the two pairs is summer v nil, which is simplified to just summer
- whereever you have single clauses you can do this (add a nil disjunct)

Resolution, Formally

- Pick two parent clauses, A and B. To be useful, they must share a common literal L with L in one of A and B, and ~L in the other
  - you can actually do resolution on any two clauses, but if you don't pick 2 clauses with this property, our result is just the conjunction of the original A and B, which gets us nowhere…
- The resulting clause, the *resolvent*, is the disjunction of all the literals in A and B, without L and ~L (they cancel out).
- What happens if the resolvent is null/false?

Null Resolvent?

- Suppose we actually performed a resolution and there was nothing left over (no resolvent)?
- resolving winter and ~winter will do this
- YEAH it means there's a contradiction in our original set of statements
  - or we screwed up converting them to clause form, but we won't do that will we?
- So now we have a way to do proofs with this approach too

Resolution Refutation

- Convert your logical system to clause form
  - these are all axioms, the system has no contradictions
- say we want to prove A. Add ~A to our list of axioms (this has to be in clause form too!)
- Repeat resolution until no progress, or we generate a null resolvent.
- If we generate a null resolvent, we have a contradiction – the inconsistency must lie in ~A, because the system was consistent without it! This therefore proves A.

Putting this into Practice

- To put this into practice we need one more thing
- Assume we have a bunch of statements in clause form. What will likely be different from the tiny examples we've seen
- They'll likely have VARIABLES in them
- Suppose we have
  season(winter) v wear(Y)
  ~season(X) v wear(hat)
- how do we resolve these? It only works if we assign specific values to the variables!

Unification

- 2.3.2 in text
- An algorithm for determining what substitutions have to be made regarding variables in order to have two sentences match
- Really, a pattern-matching algorithm
- We've been using only one type of substitution: universal instantiation
- really, lots of other types can be made too

For example…

- if we have foo(X,a,bar(Y))
- This matches foo(fred,a,bar(Z) if we substitute fred for X, Z for Y {fred/X,Z/Y}
- It also matches foo(W,a,bar(gus)) with the substitution {W/X, gus/Y}
- It also matches foo(X,a,bar(foobar(Q))) with the substitution {foobar(Q)/Y}
- really what this means is that using the substitutions we can take each of the two terms we're unifying and turn it into the same common instance

More Examples

– e.g. data(X,5) and data(apple,Y) can be unified with the substitution {apple/X, 5/Y}. The common instance is data(apple,5).
– f(X,X) and f(a,a) can be unified {a/X).
– f(X,X) and f(a,b) cannot be unified
– f(X,X) and f(g(h(8,Z)),g(h(Y,Z))) can be unified for {g(h(8,Z))/X, 8/Y} (Z is still unbound)
– f(X,X) and f(P,Q) can be unified for {X/P,X/Q}
– f(X,Y) and f(X,X) can be unified for {Y/X}

Unification Rules

- To Unify T1 and T2…
- One variable can be substituted for another
- One variable can be substituted with a constant (symbol/predicate/number), *if it is not already bound*
- If both are predicates, success if same functor (predicate name), same arity (# symbols involved in relationship), and all terms inside T1 unify with all terms inside T2 for a common substitution
  – i.e. match the predicate and recursively match what's inside!

One No No

- substitution of a variable using a structure that contains that variable
  - this should make sense: infinitely recursive!
- e.g. f(X,g(5,X)) with f(Y,Y)
  - functor, arity match, we substitute Y/X for the first term
  - matching the second requires g(5,X) to be substituted for Y
  - But this is g(5,Y)/Y if we follow the first substitution
    - =g(5,g(5,Y)) = g(5,g(5,G(5,y))) …
    - So not possible to do!

PUTTING IT ALL TOGETHER!

- OK now that we've covered resolution and unification we can put the two together and do some automated reasoning!
- Lets look at how we'd tackle a sample logical system - Our old friends marcus & caesar
- Now we already have this in standard fol – this is often the hardest part. In systems like this you may get to the point of performing resolution and realize you needed the representation to be different for it to work! Just change it and go back!

Example

person(marcus) (1)

pompeian(marcus) (2)

$\forall$ X: pompeian(X) $\rightarrow$ roman(X) (3)

ruler(caesar) (4)

$\forall$ X: roman(X) $\rightarrow$ loyal(X,caesar) v hate(X,caesar) (5)

$\forall$ X: $\exists$ Y: loyal(X,Y) (6)

$\forall$ X,Y: person(X)^ruler(Y)^ try-assassin(X,Y) $\rightarrow$ ~loyal(X,Y) (7)

try-assassin(marcus,caesar) (8)

- We now convert these to clause form. 1,2,4, & 8 are already there

Converting to Clause form

- $\forall$ X: pompeian(X) $\rightarrow$ roman(X) (3)

remove the implication:

$\forall$ X: ~pompeian(X) v roman(X)

remove the quantifier:

~pompeian(X) v roman(X)

- $\forall$ X: roman(X) $\rightarrow$ loyal(X,caesar) v hate(X,caesar) (5)

remove the implication & quantifier as above:

$\forall$ X: ~roman(X) v loyal(X,caesar) v hate(X,caesar)

~roman(X) v loyal(X,caesar) v hate(X,caesar)

Converting to Clause Form

- ∀ X: ∃ Y: loyal(X,Y) (6)

remove existential:

∀ X: loyal(X,gus) gus is our skolem constant (really it's a function of X so that we get a different object for each X – the someone X is loyal to)

now remove the univ. quantifier: loyal(X,gus)

∀ X,Y: person(X)^ruler(Y)^ try-assassin(X,Y) → ~loyal(X,Y) (7)

remove implication:

∀ X,Y: ~(person(X)^ruler(Y)^ try-assassin(X,Y)) v ~loyal(X,Y)

Converting to Clause form

∀ X,Y: ~(person(X)^ruler(Y)^ try-assassin(X,Y)) v ~loyal(X,Y)

move negation in:

∀ X,Y: ~person(X)v~ruler(Y)v~try-assassin(X,Y) v ~loyal(X,Y)

get rid of quantifiers:

~person(X) v ~ruler(Y) v ~try-assassin(X,Y) v ~loyal(X,Y)

Done Individual Clauses

- Now we have to put the system together
- And remember we have to rename the variables so that they're different between each statement
- Here's what we have now:

System in Clause Form

1. person(marcus) (1)
2. pompeian(marcus) (2)
3. ~pompeian(X) v roman(X) (from 3)
4. ruler(caesar) (4)
5. ~roman(Q) v loyal(Q,caesar) v hate(Q,caesar) (from 5)
6. loyal(R,gus) /*gus is our skolem constant*/ (from 6)
7. ~person(A) v ~ruler(B) v ~try-assassin(A,B)) v ~loyal(A,B) (from 7)
8. try-assassin(marcus,caesar) (8)

– Now say we want to prove ~loyal(marcus,caesar) (H)

– we introduce the negation of H in clause form (already there!) and see if we can resolve to a contradiction!

9. loyal(marcus,caesar)

You say you wanna Resolution…

- resolve 7 and 9:

  ~person(A) v ~ruler(B) v ~try-assassin(A,B)) v
  ~loyal(A,B) (7)

  loyal(marcus,caesar) (9)

  This gives:~person(marcus) v ~ruler(caesar) v ~try-assassin(marcus,caesar) (10)

  With the substitution {marcus/A, caesar/B}

I Resolve to Keep Goin

- Now we can resolve 10 and 1:

~person(marcus) v ~ruler(caesar) v
~try-assassin(marcus,caesar) (10)

person(marcus) (1)

Giving:

~ruler(caesar) v ~try-assassin(marcus,caesar) (11)

with no substitution. Similarly, 11 and 4 (ruler(caesar)) give us:

~try-assassin(marcus,caesar) (12)

- resolving 12 and 8 give us nil (false), a contradiction!

Contradiction

- This means that our null hypothesis has to be false, and therefore our original hypothesis must be true
- What if we couldn't generate a contradiction?
- then the original hypothesis would be true
  - but we'd have to explore all possibilities finding out ☹
- also obviously hinges on no contradictions without the null hypothesis!
- Also we could have used many strategies for choosing which items to resolve at any point, but there's only one operator to worry about