Logic

What is logic?

- Logic is an "algebra" for manipulating only two values: true (T) and false (F)
- Nevertheless, logic can be quite challenging
- This talk will cover:
  - Propositional logic--the simplest kind
  - Predicate logic (predicate calculus)--an extension of propositional logic
  - Resolution theory--a general way of doing proofs in predicate logic
  - Possibly: Conversion to clause form
  - Possibly: Other logics (just to make you aware that they exist)

Propositional logic

Propositional logic

- Propositional logic consists of:
  - The logical values true and false (T and F)
  - Propositions: “Sentences,” which
    - Are atomic (that is, they must be treated as indivisible units, with no internal structure), and
    - Have a single logical value, either true or false
  - Operators, both unary and binary; when applied to logical values, yield logical values
    - The usual operators are and, or, not, and implies

Truth tables

- Logic, like arithmetic, has operators, which apply to one, two, or more values (operands)
- A truth table lists the results for each possible arrangement of operands
  - Order is important: ***x op y*** may or may not give the same result as ***y op x***
- The rows in a truth table list all possible sequences of truth values for n operands, and specify a result for each sequence
  - Hence, there are 2^n rows in a truth table for n operands

Unary operators

- There are four possible unary operators:

X	Constant true, (T)
T	T
F	T

X	Identity, (X)
T	T
F	F

X	Constant false, (F)
T	F
F	F

X	**Negation, ¬X**
T	F
F	T

- Only the last of these (negation) is widely used (and has a symbol,¬ ,for the operation

Combined tables for unary operators

X	Constant T	Constant F	Identity	¬X
T	T	F	T	F
F	T	F	F	T

Binary operators

- There are sixteen possible binary operators:

X	Y																
T	T	T	T	T	T	T	T	T	T	F	F	F	F	F	F	F	F
T	F	T	T	T	T	F	F	F	F	T	T	T	T	F	F	F	F
F	T	T	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F
F	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F

- All these operators have names, but I haven't tried to fit them in
- Only a few of these operators are normally used in logic

Useful binary operators

- Here are the binary operators that are traditionally used:

		AND	OR	IMPLIES	BICONDITIONAL
X	Y	$X \wedge Y$	$X \vee Y$	$X \Rightarrow Y$	$X \Leftrightarrow Y$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

- Notice in particular that material implication ($\Rightarrow$) only approximately means the same as the English word "implies"
- All the other operators can be constructed from a combination of these (along with unary not, $\neg$)

Logical expressions

- All logical expressions can be computed with some combination of and ($\wedge$), or ($\vee$), and not ($\neg$) operators
- For example, logical implication can be computed this way:

X	Y	$\neg$X	$\neg$X $\vee$ Y	X $\Rightarrow$ Y
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

- Notice that $\neg X \vee Y$ is equivalent to $X \Rightarrow Y$

Another example

- Exclusive or (xor) is true if exactly one of its operands is true

X	Y	$\neg X$	$\neg Y$	$\neg X \wedge Y$	$X \wedge \neg Y$	$(\neg X \wedge Y) \vee (X \wedge \neg Y)$	X xor Y
T	T	F	F	F	F	F	F
T	F	F	T	F	T	T	T
F	T	T	F	T	F	T	T
F	F	T	T	F	F	F	F

- Notice that $(\neg X \wedge Y) \vee (X \wedge \neg Y)$ is equivalent to X xor Y

Worlds

- A world is a collection of prepositions and logical expressions relating those prepositions
- Example:
  - Propositions: JohnLovesMary, MaryIsFemale, MaryIsRich
  - Expressions:
    MaryIsFemale $\wedge$ MaryIsRich $\Rightarrow$ JohnLovesMary
- A proposition "says something" about the world, but since it is atomic (you can't look inside it to see component parts), propositions tend to be very specialized and inflexible

Models

A model is an assignment of a truth value to each proposition, for example:
  - JohnLovesMary: T, MaryIsFemale: T, MaryIsRich: F
- An expression is satisfiable if there is a model for which the expression is true
  - For example, the above model satisfies the expression MaryIsFemale $\wedge$ MaryIsRich $\Rightarrow$ JohnLovesMary
- An expression is valid if it is satisfied by *every* model
  - This expression is *not* valid:
    MaryIsFemale $\wedge$ MaryIsRich $\Rightarrow$ JohnLovesMary
    because it is not satisfied by this model:
    JohnLovesMary: F, MaryIsFemale: T, MaryIsRich: T
  - But this expression *is* valid:
    MaryIsFemale $\wedge$ MaryIsRich $\Rightarrow$ MaryIsFemale

Inference rules in propositional logic

- Here are just a few of the rules you can apply when reasoning in propositional logic:

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

From aima.eecs.berkeley.edu/slides-ppt, chs 7-9

Implication elimination

- A particularly important rule allows you to get rid of the implication operator, $\Rightarrow$:
  - $X \Rightarrow Y \;\equiv\; \neg X \vee Y$
- We will use this later on as a necessary tool for simplifying logical expressions
- The symbol $\equiv$ means “is logically equivalent to”

Conjunction elimination

- Another important rule for simplifying logical expressions allows you to get rid of the conjunction (**and**) operator, $\wedge$:
- This rule simply says that if you have an **and** operator at the top level of a fact (logical expression), you can break the expression up into two separate facts:
  - MaryIsFemale $\wedge$ MaryIsRich
    - becomes:
  - MaryIsFemale
  - MaryIsRich

Inference by computer

- To do inference (reasoning) by computer is basically a *search* process, taking logical expressions and applying inference rules to them
  - Which logical expressions to use?
  - Which inference rules to apply?
- Usually you are trying to "prove" some particular statement
- Example:
  - it_is_raining ∨ it_is_sunny
  - it_is_sunny ⇒ I_stay_dry
  - it_is_rainy ⇒ I_take_umbrella
  - I_take_umbrella ⇒ I_stay_dry
  - To prove: I_stay_dry

Forward and backward reasoning

- Situation: You have a collection of logical expressions (premises), and you are trying to prove some additional logical expression (the conclusion)
- You can:
  - Do forward reasoning: Start applying inference rules to the logical expressions you have, and stop if one of your results is the conclusion you want
  - Do backward reasoning: Start from the conclusion you want, and try to choose inference rules that will get you back to the logical expressions you have
- With the tools we have discussed so far, *neither* is feasible

Example

- Given:
  - it_is_raining $\vee$ it_is_sunny
  - it_is_sunny $\Rightarrow$ I_stay_dry
  - it_is_raining $\Rightarrow$ I_take_umbrella
  - I_take_umbrella $\Rightarrow$ I_stay_dry
- You can conclude:
  - it_is_sunny $\vee$ it_is_raining
  - I_take_umbrella $\vee$ it_is_sunny
  - $\neg$I_stay_dry $\Rightarrow$ I_take_umbrella
  - Etc., etc. ... there are *just too many* things you can conclude!

Predicate Calculus

Predicate calculus

- Predicate calculus is also known as "First Order Logic" (FOL)
- Predicate calculus includes:
  - All of propositional logic
    - Logical values true, false
    - Variables x, y, a, b,...
    - Connectives $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
  - Constants KingJohn, 2, Villanova,...
  - Predicates Brother, >,...
  - Functions Sqrt, MotherOf,...
  - Quantifiers $\forall$, $\exists$

Constants, functions, and predicates

- A constant represents a “thing”--it has no truth value, and it does not occur “bare” in a logical expression
  - Examples: DavidMatuszek, 5, Earth, goodIdea
- Given zero or more arguments, a function produces a constant as its value:
  - Examples: motherOf(DavidMatuszek), add(2, 2), thisPlanet()
- A predicate is like a function, but produces a truth value
  - Examples: greatInstructor(DavidMatuszek), isPlanet(Earth), greater(3, add(2, 2))

Universal quantification

- The universal quantifier, $\forall$, is read as "for each" or "for every"
  - Example: $\forall x, x^2 \geq 0$ (for all x, x^2 is greater than or equal to zero)
- Typically, $\Rightarrow$ is the main connective with $\forall$:

  $\forall x, \text{at}(x,\text{Villanova}) \Rightarrow \text{smart}(x)$

  means "Everyone at Villanova is smart"
- Common mistake: using $\wedge$ as the main connective with $\forall$:

  $\forall x, \text{at}(x,\text{Villanova}) \wedge \text{smart}(x)$

  means "Everyone is at Villanova and everyone is smart"
- If there are no values satisfying the condition, the result is true
  - Example: $\forall x, \text{isPersonFromMars}(x) \Rightarrow \text{smart}(x)$ is true

Existential quantification

- The existential quantifier, $\exists$, is read “for some” or “there exists”
  - Example: $\exists x, x^2 < 0$ (there exists an x such that x^2 is less than zero)
- Typically, $\wedge$ is the main connective with $\exists$:

  $\exists x$, at(x,Villanova) $\wedge$ smart(x)

  means “There is someone who is at Villanova and is smart”
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

  $\exists x$, at(x,Villanova) $\Rightarrow$ smart(x)

  This is true if there is someone at Villanova who is smart...

  ...but it is also true if there is someone who is *not* at Villanova

  By the rules of material implication, the result of F $\Rightarrow$ T is T

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$

- $\exists x\ \forall y$ is *not* the same as $\forall y\ \exists x$
- $\exists x\ \forall y\ Loves(x,y)$
  - "There is a person who loves everyone in the world"
  - More exactly: $\exists x\ \forall y\ (person(x) \wedge person(y) \Rightarrow Loves(x,y))$
- $\forall y\ \exists x\ Loves(x,y)$
  - "Everyone in the world is loved by at least one person"

- Quantifier duality: each can be expressed using the other
- $\forall x\ Likes(x,IceCream)$ $\qquad \neg\exists x\ \neg Likes(x,IceCream)$
- $\exists x\ Likes(x,Broccoli)$ $\qquad \neg\forall x\ \neg Likes(x,Broccoli)$

From aima.eecs.berkeley.edu/slides-ppt, chs 7-9

Parentheses

- Parentheses are often used with quantifiers
- Unfortunately, everyone uses them differently, so don't be upset at any usage you see
- Examples:
  - $(\forall x)\ person(x) \Rightarrow likes(x,iceCream)$
  - $(\forall x)\ (person(x) \Rightarrow likes(x,iceCream))$
  - $(\forall x)\ [\ person(x) \Rightarrow likes(x,iceCream)\ ]$
  - $\forall x,\ person(x) \Rightarrow likes(x,iceCream)$
  - $\forall x\ (person(x) \Rightarrow likes(x,iceCream))$
- I prefer parentheses that show the scope of the quantifier
  - $\exists x\ (x > 0) \wedge \exists x\ (x < 0)$

More rules

- Now there are numerous additional rules we can apply!
- Here are two exceptionally important rules:
  - $\neg\forall x,\ p(x) \ \Rightarrow\ \exists x,\ \neg p(x)$
    "If not every x satisfies $p(x)$, then there exists a x that does not satisfy $p(x)$"
  - $\neg\exists x,\ p(x) \ \Rightarrow\ \forall x,\ \neg p(x)$
    "If there does not exist an x that satisfies $p(x)$, then all x do not satisfy $p(x)$"
- In any case, the search space is *just too large* to be feasible
- This was the case until 1970, when J. Robinson discovered resolution

Interlude: Definitions

- syntax: defines the formal structure of sentences
- semantics: determines the truth of sentences wrt (with respect to) models
- entailment: one statement entails another if the truth of the first means that the second must also be true
- inference: deriving sentences from other sentences
- soundness: derivations produce only entailed sentences
- completeness: derivations can produce all entailed sentences

Resolution

Logic by computer was infeasible

- Why is logic so hard?
  - You start with a large collection of facts (predicates)
  - You start with a large collection of possible transformations (rules)
    - Some of these rules apply to a single fact to yield a new fact
    - Some of these rules apply to a pair of facts to yield a new fact
  - So at every step you must:
    - Choose some rule to apply
    - Choose one or two facts to which you might be able to apply the rule
      - If there are n facts
        - There are n potential ways to apply a single-operand rule
        - There are n * (n - 1) potential ways to apply a two-operand rule
    - Add the new fact to your ever-expanding fact base
  - The search space is huge!

The magic of resolution

- Here's how resolution works:
  - You transform each of your facts into a particular form, called a clause (this is the tricky part)
  - You apply a *single rule,* the resolution principle, to a pair of clauses
    - Clauses are closed with respect to resolution--that is, when you resolve two clauses, you get a new clause
  - You add the new clause to your fact base
- So the number of facts you have grows linearly
  - You still have to choose a pair of facts to resolve
  - You never have to choose a rule, because there's only one

The fact base

- A fact base is a collection of "facts," expressed in predicate calculus, that are presumed to be true (valid)
- These facts are implicitly "**and**ed" together
- Example fact base:
  - seafood(X) $\Rightarrow$ likes(John, X)   (where X is a variable)
  - seafood(shrimp)
  - pasta(X) $\Rightarrow$ $\neg$likes(Mary, X)   (where X is a *different* variable)
  - pasta(spaghetti)
- That is,
  - (seafood(X) $\Rightarrow$ likes(John, X)) $\wedge$ seafood(shrimp) $\wedge$ (pasta(Y) $\Rightarrow$ $\neg$likes(Mary, Y)) $\wedge$ pasta(spaghetti)
  - Notice that we had to change some Xs to Ys
  - The scope of a variable is the single fact in which it occurs

Clause form

- A clause is a disjunction ("**or**") of zero or more literals, some or all of which may be negated
- Example:
  $sinks(X) \lor dissolves(X, water) \lor \neg denser(X, water)$
- Notice that clauses use only "or" and "not"—they do not use "and," "implies," or either of the quantifiers "for all" or "there exists"
- The impressive part is that *any* predicate calculus expression can be put into clause form
  - Existential quantifiers, $\exists$, are the trickiest ones

Unification

- From the pair of facts (not yet clauses, just facts):
  - seafood(X) ⇒ likes(John, X)   (where X is a variable)
  - seafood(shrimp)
- We ought to be able to conclude
  - likes(John, shrimp)
- We can do this by unifying the variable X with the constant shrimp
  - This is the *same* "unification" as is done in Prolog
- This unification turns seafood(X) ⇒ likes(John, X) into seafood(shrimp) ⇒ likes(John, shrimp)
- Together with the given fact seafood(shrimp), the final deductive step is easy

The resolution principle

- Here it is:
  - From $\quad X \vee$ someLiterals
    and $\quad \neg X \vee$ someOtherLiterals
    \---
    conclude: someLiterals $\vee$ someOtherLiterals
- That's all there is to it!
- Example:
  - broke(Bob) $\vee$ well-fed(Bob)
    ¬broke(Bob) $\vee$ ¬hungry(Bob)
    \--------------------------------------
    well-fed(Bob) $\vee$ ¬hungry(Bob)

A common error

- You can only do *one* resolution at a time
- Example:
  - broke(Bob) $\vee$ well-fed(Bob) $\vee$ happy(Bob)
    ¬broke(Bob) $\vee$ ¬hungry(Bob) $\vee$ ¬happy(Bob)
- You can resolve on broke to get:
  - well-fed(Bob) $\vee$ happy(Bob) $\vee$ ¬hungry(Bob) $\vee$ ¬happy(Bob) $\equiv$ T
- Or you can resolve on happy to get:
  - broke(Bob) $\vee$ well-fed(Bob) $\vee$ ¬broke(Bob) $\vee$ ¬hungry(Bob) $\equiv$ T
- Note that both legal resolutions yield a tautology (a trivially true statement, containing $X \vee \neg X$), which is correct but useless
- But you *cannot* resolve on both at once to get:
  - well-fed(Bob) $\vee$ ¬hungry(Bob)

Contradiction

- A special case occurs when the result of a resolution (the resolvent) is empty, or “NIL”
- Example:
  - hungry(Bob)
    ¬hungry(Bob)

    NIL
- In this case, the fact base is inconsistent
- This will turn out to be a very useful observation in doing resolution theorem proving

A first example

- "Everywhere that John goes, Rover goes. John is at school."
  - at(John, X) $\Rightarrow$ at(Rover, X) (not yet in clause form)
  - at(John, school) (already in clause form)
- We use implication elimination to change the first of these into clause form:
  - $\neg$at(John, X) $\vee$ at(Rover, X)
  - at(John, school)
- We can resolve these on at(-, -), but to do so we have to unify X with school; this gives:
  - at(Rover, school)

Refutation resolution

- The previous example was easy because it had very few clauses
- When we have a lot of clauses, we want to *focus* our search on the thing we would like to prove
- We can do this as follows:
  - Assume that our fact base is consistent (we can't derive NIL)
  - Add the *negation* of the thing we want to prove to the fact base
  - Show that the fact base is now inconsistent
  - Conclude the thing we want to prove

Example of refutation resolution

- "Everywhere that John goes, Rover goes. John is at school. **Prove that Rover is at school.**"
  1. ¬at(John, X) ∨ at(Rover, X)
  2. at(John, school)
  3. ¬at(Rover, school) (this is the added clause)
- Resolve #1 and #3:
  4. ¬at(John, X)
- Resolve #2 and #4:
  5. NIL
- Conclude the negation of the added clause: at(Rover, school)
- This seems a roundabout approach for such a simple example, but it works well for larger problems

A second example

- Start with:
  - it_is_raining $\lor$ it_is_sunny
  - it_is_sunny $\Rightarrow$ I_stay_dry
  - it_is_raining $\Rightarrow$ I_take_umbrella
  - I_take_umbrella $\Rightarrow$ I_stay_dry
- Convert to clause form:
  1. it_is_raining $\lor$ it_is_sunny
  2. $\neg$it_is_sunny $\lor$ I_stay_dry
  3. $\neg$it_is_raining $\lor$ I_take_umbrella
  4. $\neg$I_take_umbrella $\lor$ I_stay_dry
- Prove that I stay dry:
  5. $\neg$I_stay_dry
- Proof:
  6. (5, 2) $\neg$it_is_sunny
  7. (6, 1) it_is_raining
  8. (5, 4) $\neg$I_take_umbrella
  9. (8, 3) $\neg$it_is_raining
  10. (9, 7) NIL
- Therefore, $\neg$($\neg$I_stay_dry)
  - I_stay_dry

Conversion to clause form

A nine-step process

Reference: *Artificial Intelligence*, by Elaine Rich and Kevin Knight

Running example

- All Romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy

- $\forall x, [\ Roman(x) \wedge know(x, Marcus)\ ] \Rightarrow$
  $[\ hate(x, Caesar) \vee$
  $(\forall y, \exists z, hate(y, z) \Rightarrow thinkCrazy(x, y))]$

Step 1: Eliminate implications

- Use the fact that $x \Rightarrow y$ is equivalent to $\neg x \vee y$

- $\forall x$, [Roman(x) $\wedge$ know(x, Marcus)] $\Rightarrow$
  [hate(x, Caesar) $\vee$
  ($\forall y$, $\exists z$, hate(y, z) $\Rightarrow$ thinkCrazy(x, y))]

- $\forall x$, $\neg$[Roman(x) $\wedge$ know(x, Marcus)] $\vee$
  [hate(x, Caesar) $\vee$
  ($\forall y$, $\neg$($\exists z$, hate(y, z) $\vee$ thinkCrazy(x, y))]

Step 2: Reduce the scope of $\neg$

- Reduce the scope of negation to a single term, using:
  - $\neg(\neg p) \equiv p$
  - $\neg(a \wedge b) \equiv (\neg a \vee \neg b)$
  - $\neg(a \vee b) \equiv (\neg a \wedge \neg b)$
  - $\neg\forall x,\ p(x) \quad \equiv \quad \exists x,\ \neg p(x)$
  - $\neg\exists x,\ p(x) \quad \equiv \quad \forall x,\ \neg p(x)$

- $\forall x,\ \neg[\ Roman(x) \wedge know(x, Marcus)\ ] \vee$
  $[hate(x, Caesar) \ \vee$
  $(\forall y,\ \neg(\exists z,\ hate(y, z) \vee thinkCrazy(x, y))]$

- $\forall x,\ [\ \neg Roman(x) \vee \neg know(x, Marcus)\ ] \vee$
  $[hate(x, Caesar) \ \vee$
  $(\forall y,\ \forall z,\ \neg hate(y, z) \vee thinkCrazy(x, y))]$

Step 3: Standardize variables apart

- $\forall x, P(x) \vee \forall x, Q(x)$
  becomes
  $\forall x, P(x) \vee \forall y, Q(y)$
- This is just to keep the scopes of variables from getting confused
- Not necessary in our running example

Step 4: Move quantifiers

- Move all quantifiers to the left, without changing their relative positions

- $\forall x, [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y, \forall z, \neg hate(y, z) \vee thinkCrazy(x, y)]$

- $\forall x, \forall y, \forall z, [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkCrazy(x, y))]$

Step 5: Eliminate existential quantifiers

- We do this by introducing Skolem functions:
  - If $\exists x, p(x)$ then just pick one; call it x'
  - If the existential quantifier is under control of a universal quantifier, then the picked value has to be a function of the universally quantified variable:
    - If $\forall x, \exists y, p(x, y)$ then $\forall x, p(x, y(x))$
- Not necessary in our running example

Exercise

- Convert the following in to skolem form

$$\exists u\ \forall v\ \ \forall x\ \exists y\ P(f(u), v, x, y) \Rightarrow Q(u, v, y)$$

Step 6: Drop the prefix (quantifiers)

- $\forall x, \forall y, \forall z, [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkCrazy(x, y))]$
- At this point, all the quantifiers are universal quantifiers
- We can just take it for granted that all variables are universally quantified
- $[\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y, z) \vee thinkCrazy(x, y))]$

Step 7: Create a conjunction of disjuncts

- [¬Roman(x) ∨ ¬know(x, Marcus)] ∨
  [hate(x, Caesar) ∨ (¬hate(y, z) ∨ thinkCrazy(x, y))]

  becomes

  ¬Roman(x) ∨ ¬know(x, Marcus) ∨
  hate(x, Caesar) ∨ ¬hate(y, z) ∨ thinkCrazy(x, y)

Step 8: Create separate clauses

- Every place we have an $\wedge$, we break our expression up into separate pieces
- Not necessary in our running example

Step 9: Standardize apart

- Rename variables so that no two clauses have the same variable
- Not necessary in our running example

- Final result:

  $\neg Roman(x) \vee \neg know(x, Marcus) \vee hate(x, Caesar) \vee \neg hate(y, z) \vee thinkCrazy(x, y)$

- That's it! It's a long process, but easy enough to do mechanically

Exercise

- Convert the following into clausal form

$$\exists x\ \forall y\ (\forall z\ P(f(x), y, z) \Rightarrow (\exists u\ Q(x, u)\ \&\ \exists v\ R(y, v)))$$