First Order Predicate Calculus
working with formulas

Yesterday we learned the Theory of First Order Logic (FOL).
Today we're learning how to work with formulas.

- Representing Formulas
- Reducing formulas
- Normal Forms

Review

Let's start by converting sentences to First Order Logic

- Every program either halts, or it runs forever
- for every function f from $A \to B$, and every function g from $C \to D$, if C and A are the same, then there is an $h = f \circ g$.
- Every number is the sum of four primes.
- define n-colorable

- $\forall p \in \mathit{Programs}.\ \mathit{Halts}(p) \vee (\forall t \in \mathit{Time}.\ \mathit{Runs}(p, t)$
- $(\forall f : A \to B.(\forall g : C \to D.D = A \to (\exists h : C \to B.h = f \circ g)))$
- $\forall(n \in \mathbb{N}).\exists(p_1, p_2, p_3, p_4 \in \mathbb{P}).n = \sum_{i=1}^{4} p_i$
- $\forall v_i \in V.\ \ \mathit{color}(v_i) \in \{1 \ldots n\}$
  $(\forall v_j \in V.\ \ (v_i, v_j) \in E \to \mathit{color}(v_i) \neq \mathit{color}(v_j))$

Quantifiers

What can we do with quantifiers?

From predicate logic we know

$$
\begin{array}{rcll}
a \wedge \top & = & a & Id_{\wedge} \\
a \vee \bot & = & a & Id_{\vee} \\
a \wedge b & = & b \wedge a & Com_{\wedge} \\
a \vee b & = & b \vee a & Com_{\vee} \\
a \wedge (b \vee c) & = & (a \wedge b) \vee (a \wedge c) & Dis_{\wedge} \\
a \vee (b \wedge c) & = & (a \vee b) \wedge (a \vee c) & Dis_{\vee} \\
a \wedge a & = & a & Item_{\wedge} \\
a \vee a & = & a & Item_{\vee} \\
a \vee \top & = & \top & Anul_{\vee} \\
a \wedge \bot & = & \bot & Anul_{\wedge} \\
a & = & \neg\neg a & \neg\neg \\
\neg(a \wedge b) & = & \neg a \vee \neg b & DM_{\wedge} \\
\neg(a \vee b) & = & \neg a \wedge \neg b & DM_{\vee} \\
a \rightarrow b & = & \neg a \vee b & Imp \\
a \vee \neg a & = & \top & LEM
\end{array}
$$

Quantifiers

$$
\begin{array}{rcll}
(\forall x.a) \wedge b & = & \forall x.(a \wedge b) & \forall\wedge \\
(\exists x.a) \wedge b & = & \exists x.(a \wedge b) & \exists\wedge \\
(\forall x.a) \vee b & = & \forall x.(a \vee b) & \forall\vee \\
(\exists x.a) \vee b & = & \exists x.(a \vee b) & \exists\vee \\
a \to (\forall x.b) & = & \forall x.(a \to b) & Cov_{\forall} \\
a \to (\exists x.b) & = & \exists x.(a \to b) & Cov_{\exists} \\
(\forall x.a) \to b & = & \exists x.(a \to b) & Cont_{\forall} \\
(\exists x.a) \to b & = & \forall x.(a \to b) & Cont_{\exists} \\
\forall xy.a & = & \forall yx.a & Com_{\forall} \\
\exists xy.a & = & \exists yx.a & Com_{\exists} \\
\neg(\forall x.a) & = & (\exists x.\neg a) & DM_{\forall} \\
\neg(\exists x.a) & = & (\forall x.\neg a) & DM_{\exists} \\
\forall x.a & = & a \text{ if } x \notin a & Item_{\forall} \\
\exists x.a & = & a \text{ if } x \notin a & Item_{\exists} \\
\forall x.a & = & \forall z.a[x \mapsto z] & Rep_{\forall} \\
\exists x.a & = & \exists z.a[x \mapsto z] & Rep_{\exists}
\end{array}
$$

Rules and free variables

We have to be careful with these rules. We saw last time that you can't move quantifiers around free variables. This is true for all of these rules.

$(\forall x.P(x)) \wedge Q(x)$
Is not the same as
$\forall x.P(x) \wedge Q(x)$
Even though the rule $\forall\wedge$ looks like it applies.

This seems like a big problem, but we can always rename variables.
$(\forall x.P(x)) \wedge Q(x)$
$(\forall z.P(z)) \wedge Q(x) Rep_{\forall}$
$\forall z.P(z) \wedge Q(x) \forall\wedge$

Normal Forms

We only have one new normal form to worry about in FOL.

A formula is in **prenex** normal form is all of the quantifiers are at the top level.

Here the Top level means they are outside the rest of the formula.
So they're all the way on the left.
Or, the qualifiers are on the *top level* of the syntax tree.

Note: the word prenex comes from praenexus, which latin for "bound up in front"
(according to Wikipedia)

PNF example

The formulas
$\forall x.Q(x,x)$
$\forall x \exists y.P(x) \wedge P(y)$
$\exists x \forall y.P(x) \wedge P(y)$
$\forall x \exists y \forall z.P(x) \rightarrow Q(y,z)$
are all in PNF

We don't care about the order of the quantifiers,
just as long as they're outside.

The formula $(\forall x.Q(x)) \rightarrow P$ is NOT in PNF.
$\forall$ is on the left, but it's part of a subexpression of the $\rightarrow$.
If you're not sure if it's in PNF, draw the syntax tree.

PNF, CNF, and DNF

You'll notice that PNF is not related to CNF or DNF.
We can have a formula in PNF where the unquantified part is in CNF/DNF.

suppose we have
$((\forall x.P(x)) \vee (\forall y.P(y))) \wedge (\exists y.P(y))$

In PNF this is
$\forall xz \exists y.(P(x) \vee P(z)) \wedge P(y)$

In PNF-DNF this is
$\forall xz \exists y.(P(x) \wedge P(y)) \vee (P(z) \wedge P(y))$

Examples

Convert the following to PNF

- $(\forall x.P(x)) \wedge (\forall y.Q(y))$
- $(\forall x.P(x) \rightarrow Q(x)) \rightarrow (\forall y.Q(y) \rightarrow P(y))$
- $(\forall x.P(x)) \vee (\forall x.Q(x))$
- $(\forall x.P(x) \rightarrow Q(x)) \wedge (\exists y.Q(y))$
  $\wedge(\exists z.P(z)) \wedge (\exists z.Q(z) \rightarrow R(x))$

Simplification

Simplification works just like it did in propositional logic.

The only difference is that we have about twice as many rules.

A good general rule of Thumb, get the formula in PNF first.
The rules to get a formula in PNF usually make it smaller.
Bonus: once it's in PNF, we know how to work with it.

Examples

Simplify the following.

- $(\forall x.P(x)) \wedge (\exists y.\neg P(y))$
- $(\forall x.P(x) \rightarrow Q(x)) \rightarrow (\forall y.Q(y) \rightarrow P(y))$
- $(\forall x.P(x)) \vee (\forall x.Q(x))$

Models and validity

Just like last time, we want a general way to check if we're done with rules.

So, what's the first order logic equivalent of truth tables?

Well... I've got some bad news.

Models

While statements in FOL are either true or false, they're more complicated.
Specifically statements in first order logic are more general.

Now instead of talking about True and False Statements,
I can talk about numbers, cows, and programs.
This makes it much harder to tell if a statement it True.

Instead, we talk about our statement being true in models.
A **model** interprets logic statements in a domain.

Models

Remember, a domain is just a set. So, given a domain D, we define two sets
$C \subseteq D$ is a set of constant symbols.
$F_n \subseteq D^n \to D$ is a set of functions that take n arguments.
$P_n \subseteq D^n$ is a set fo relation of n elements on D.

Let $\mathcal{F} = \cup_{i=1}^{\infty} F_n$
Let $\mathcal{P} = \cup_{i=1}^{\infty} P_n$
Then $M = \{C, \mathcal{F}, \mathcal{P}\}$ is a model.

This is a bit messy. Basically we take every constant, function, and predicate symbol we could use in a formula, and we map it to a real constant/function/predicate in D.

Models: Arith

Let's see an example of this.
let
$C := \{1, 2, \ldots\}$
$F_2 := \{+, \times\}$
$P_2 := \{=, \neq, <, >, \leq, \geq\}$
Then,
$Arith = \{C, F_2, P_2\}$ is a model.

The following statements are valid
$\forall x, y, z. x \times (y + z) = x \times y + x \times x$
$\forall x, y, z. x < y \wedge y < z \rightarrow x < z$

But these are valid *only* for this model.

Models: Grp

We can actually multiple models for the same system. Well, have a constant e,
functions $*(x, y)$ and $i(x)$
and a binary predicate $=$
We will require that the following formulas be valid.
$\forall x, y, z.(x * y) * z == x * (y * z)$
$\forall x, y, z.x * (y * z) == (x * y) * z$
$\forall x.x * e == x$
$\forall x.x * i(x) == e$
$\forall x.i(x) * x == e$
There are a bunch of models of this.

Models: Grp

If our domain is $\mathbb{Z}$, then
$e := 0$
$*(x, y) := x + y$
$i(x) := -x$
$x = y$ is normal equality.
This is addittion of integers.
It satisfies all of the formulas.

Models: Grp

We could also have our domain is $\mathbb{R}^{n\times n}$, then
e is the 0 matrix
$*(x, y)$ is matrix addition
$i(x)$ is the negative matrix
$x = y$ is matrix equality.
This is addittion of matrices.

Models: Grp

If we pick a specific subset of matrices, we get e is the identity
$*(x, y)$ is matrix multiplication
$i(x)$ is the inverse matrix
$x = y$ is matrix equality.

Now our domain is non-singular matrices ($|A| \neq 0$)
This is multiplication of matrices.

Models: Grp

We can even do permutations.
Our domain is $P : \mathbb{N}_k \to \mathbb{N}_k$, the set of permutations from $1..k$ e doesn't permute anything
$*(x, y)$ is composes permutations
$i(x)$ undoes the permutation x.
$x = y$ if two permutations are the same.

There are actually a lot of models of this system.
Anything that's a model of this system we call a group.

Models and validity

While we can define formulas that are true for some model, in general we want formulas that are true for all models.

A formula is **valid** if it is true for all possible models.
In general it's hard to show anything is valid for every model.

However, it's really easy to show that things are invalid.
If we construct a single model where the formula doesn't hold, then it's invalid.

decidability

In propositional logic we called a valid formula a tautology.

We found out that we can always determine if a formula is a tautology.
We just construct the truth table and see if all rows are $\top$.

But, we can't construct a truth table for predicates.

So, how do we determine if a formula is valid?

theorem:
There is no algorithm for deciding if a propositional logic formula is valid.