Predicate Calculus

There are lots of things that can't be expressed by propositional formulas. In first-order logic, we can:

- Talk about individuals and the properties they have:
  - Bob and Alice are both American
    $American(Bob) \wedge American(Alice)$
- Talk about the relations between individuals
  - Bob loves Alice but Bob doesn't love Anne
    $Loves(Bob, Alice) \wedge \neg Loves(Bob, Anne)$.
- Quantify:
  - Everybody loves somebody
    $\forall x \exists y Loves(x, y)$

First-order logic lets us capture arguments like:

All men are mortal
Socrates is a man
Therefore Socrates is mortal

All prime numbers are integers
7 is a prime number
Therefore 7 is an integer

Syntax of First-Order Logic

We have:

- *constant symbols*: *Alice*, *Bob*
- variables: $x, y, z, \ldots$
- *predicate symbols* of each arity: P, Q, R, ...
  - A *unary* predicate symbol takes one argument: $P(Alice)$, $Q(z)$
  - A *binary* predicate symbol takes two arguments: *Loves(Bob,Alice)*, *Taller(Alice,Bob)*.

An *atomic expression* is a predicate symbol together with the appropriate number of arguments.

- Atomic expressions act like primitive propositions in propositional logic
  - we can apply $\wedge$, $\vee$, $\neg$ to them
  - we can also quantify the variables that appear in them

Typical formula:

$$\forall x \exists y (P(x, y) \Rightarrow \exists z Q(x, z))$$

Semantics

Assume we have some domain D.

- The domain could be finite:
  - $\{1, 2, 3, 4, 5\}$
  - the people in this room
- The domain could be infinite
  - N, R, ...

A statement like $\forall x P(x)$, means that $P(d)$ is true for each d in the domain.

- If the domain is N, then $\forall x P(x)$ is equivalent to

$$P(1) \wedge P(2) \wedge \ldots$$

Similarly, $\exists x P(x)$ means that $P(d)$ is true for some d in the domain.

- If the domain is N, then $\exists x P(x)$ is equivalent to

$$P(1) \vee P(2) \vee \ldots$$

Is $\exists x(x^2 = 2)$ true?

Yes if the domain is R; no if the domain is N.

How about $\forall x \forall y((x < y) \Rightarrow \exists z(x < z < y))$?

Translating from English to First-Order Logic

All men are mortal
Socrates is a man
Therefore Socrates is mortal

There is two unary predicates: $Mortal$ and Man
There is one constant: $Socrates$
The domain is the set of all people

$\forall x(Man(x) \Rightarrow Mortal(x))$
$Man(Socrates)$

$Mortal(Socrates)$

More on Quantifiers

$\forall x \forall y P(x, y)$ is equivalent to $\forall y \forall x P(x, y)$

- P is true for every choice of x and y

Similarly $\exists x \exists y P(x, y)$ is equivalent to $\exists y \exists x P(x, y)$

- P is true for some choice of (x, y).

What about $\forall x \exists y P(x, y)$? Is it equivalent to $\exists y \forall x P(x, y)$?

- Suppose the domain is the natural numbers. Compare:
  - $\forall x \exists y (y \geq x)$
  - $\exists y \forall x (y \geq x)$

In general, $\exists y \forall x P(x, y) \Rightarrow \forall x \exists y P(x, y)$ is *logically valid.*

- A logically valid formula in first-order logic is the analogue of a tautology in propositional logic.
- A formula is logically valid if it's true in every domain and for every *interpretation* of the predicate symbols.

More valid formulas involving quantifiers:

- $\neg\forall x P(x) \Leftrightarrow \exists x \neg P(x)$
- Replacing P by $\neg P$, we get:

$$\neg\forall x \neg P(x) \Leftrightarrow \exists x \neg\neg P(x)$$

- Therefore

$$\neg\forall x \neg P(x) \Leftrightarrow \exists x P(x)$$

- Similarly, we have

$$\neg\exists x P(x) \Leftrightarrow \forall x \neg P(x)$$

$$\neg\exists x \neg P(x) \Leftrightarrow \forall x P(x)$$

Bound and Free Variables

$\forall i(i^2 > i)$ is equivalent to $\forall j(j^2 > j)$:

- the i and j are *bound* variables, just like the i, j in

$$\sum_{i=1}^{n} i^2 \text{ or } \sum_{j=1}^{n} j^2$$

What about $\exists i(i^2 = j)$:

- the i is bound by $\exists i$; the j is *free*. Its value is unconstrained.
- if the domain is the natural numbers, the truth of this formula depends on the value of j.

Theorems and Proofs

Just as in propositional logic, there are axioms and proof rules that provide a complete axiomatization for first-order logic, independent of the domain.

A typical axiom:

- $\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$.

Suppose we restrict the domain to the natural numbers, and allow only the standard symbols of arithmetic ($+$, $\times$, $=$, $>$, 0, 1). Typical true formulas include:

- $\forall x \exists y(x \times y = x)$
- $\forall x \exists y(x = y + y \vee x = y + y + 1)$

Let $Prime(x)$ be an abbreviation for

$$\forall y \forall z((x = y \times z) \Rightarrow ((y = 1) \vee (y = x)))$$

- $Prime(x)$ is true if x is prime

What does the following formula say:

- $\forall x(\exists y(y > 1 \wedge x = y + y) \Rightarrow$
  $\exists z_1 \exists z_2(Prime(z_1) \wedge Prime(z_2) \wedge x = z_1 + z_2))$
- This is *Goldbach's conjecture*: every even number other than 2 is the sum of two primes.
  - Is it true? We don't know.

Is there a sound and complete axiomatization for arithmetic?

- A small collection of axioms and inference rules such that every true formula of arithmetic can be proved from them
- *Gödel's Theorem:* NO!

A Complete Axiomatization for Propositional Logic

All you need are two axioms *schemes*:

Ax1. $A \Rightarrow (B \Rightarrow A)$

Ax2. $(A \Rightarrow (B \Rightarrow C) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$

and one inference rule: Modus Ponens:

- From $A \Rightarrow B$ and A infer B

Ax1 and Ax2 are axioms schemes:

- each one encodes an infinite set of axioms (obtained by plugging in arbitrary formulas for A, B, C

A *proof* is a sequence of formulas $A_1, A_2, A_3, \ldots$ such that each A_i is either

1. An instance of Ax1 and Ax2
2. Follows from previous formulas by applying MP
   - that is, there exist A_j, A_k with $j, k < i$ such that A_j has the form $A \Rightarrow B$, A_k is A and A_i is B.

Axiomatizing First-Order Logic

There's also an elegant complete axiomatization for first-order logic.

- Again, the only inference rule is Modus Ponens
- Typical axiom:

$$\forall x(P(x) \Rightarrow Q(x)) \Rightarrow (\forall x P(x) \Rightarrow \forall x Q(x))$$

- Completeness was proved by Gödel in 1930

Coverage of Final

- everything covered by the first prelim
  - slight emphasis on more recent material
- Chapter 3: Graphs and Trees
  - Knowing when Eulerian cycles exist and how to find them
  - Dijkstra's algorithm
  - Breadth-first search and depth-first search
  - Minimum spanning trees (Prim's algorithm)
- Chapter 4: Fundamental Counting Methods
  - Basic methods: sum rule, product rule, division rule
  - Permutations and combinations
  - Combinatorial identities (know Theorems 1–4 on pp. 310–314)
  - Pascal's triangle
  - Binomial Theorem (but not multinomial theorem)
  - Balls and urns
  - Inclusion-exclusion

- Pigeonhole principle

- Chapter 6: Probability:
  - 6.1–6.5 (but not Poisson and inverse binomial distribution)
  - basic definitions: probability space, events
  - conditional probability, independence, Bayes Thm.
  - random variables, uniform + binomial distribution
  - expected value and variance

- Chapter 7: Logic:
  - 7.1–7.4, 7.6; *not* 7.5
  - translating from English to propositional (or first-order) logic
  - truth tables and axiomatic proofs
  - algorithm verification
  - first-order logic

(A Little Bit on) NP

(No details here; just a rough sketch of the ideas. Take CS 381/481 if you want more.)

NP = nondeterministic polynomial time

- a problem is in NP if you can guess a solution and quickly (in polynomial time) verify that it's correct.
- Examples:
  - Does a graph have a Hamiltonian path?
  - Is a formula satisfiable?
  - Is there a schedule that satisfies certain constraints?
  - ...
- A problem is NP-hard if every NP problem can be *reduced* to it.
  - I'll give an example of reduction next

NP-completeness

A problem is NP-complete if it is in NP and NP-hard

- Intuitively, if it is one of the hardest problems in NP.

There are *lots* of problems known to be NP-complete

- If any NP complete problem is doable in polynomial time, then they all are.
  - Hamiltonian path
  - satisfiability
  - scheduling
  - ...
- If you can prove P = NP, you'll get a Turing award.

The Clique Problem

Remember: a *clique* in a graph is a completely connected subgraph

- a set of nodes such that every pair is connected by an edge

The CLIQUE problem: Given graph G and a number k, does G have a clique of size k?
Theorem: CLIQUE is NP-complete.

Proof: Clearly, CLIQUE is in NP: just guess the nodes in the clique and verify that they're all connected to each other.

To show CLIQUE is NP-hard, reduce 3-CNF to CLIQUE:

- Given a 3-CNF formula φ with k clauses, find a graph G such that G has a clique of size k iff φ is satisfiable.

Idea of proof:
Suppose $\varphi = C_1 \wedge \ldots \wedge C_k$;

- $C_i = l_{1k} \vee l_{2k} \vee l_{3k}$, where l_{ij} is a literal

Construct a graph $G_\varphi = (V_\varphi, E_\varphi)$ as follows:

- put a vertex v_{ij} in V_φ for each literal l_{ij}, $i = 1, 2, 3$, $j = 1, \ldots, k$
  - That means that V_φ has $3k$ vertices
- Put an edge between v_{ij} and $v_{i'j'}$ if
  - $j \neq j'$ (so that l_{ij} and $l_{i'j'}$ are indifferent clauses)
  - l_{ij} is not equivalent to $\neg l_{i'j'}$

Claim: φ is satisfiable iff G_φ has a k-clique

- If φ is satisfiable, there is a truth assignment such that for each j, (at least) one of l_{1j}, l_{2j}, and l_{3j} is true.
  - Pick one: the set of literals you picked forms a k-clique
- if there is a k-clique, define a truth assignment that makes each literal in the k-clique true
  - This gives an assignment that satisfies φ

Ten Powerful Ideas

- **Counting**: Count without counting (*combinatorics*)
- **Induction**: Recognize it in all its guises.
- **Exemplification**: Find a sense in which you can try out a problem or solution on small examples.
- **Abstraction**: Abstract away the inessential features of a problem.
  - One possible way: represent it as a graph
- **Modularity**: Decompose a complex problem into simpler subproblems.
- **Representation**: Understand the relationships between different possible representations of the same information or idea.
  - Graphs vs. matrices vs. relations
- **Refinement**: The best solutions come from a process of repeatedly refining and inventing alternative solutions.
- **Toolbox**: Build up your vocabulary of abstract structures.

- **Optimization**: Understand which improvements are worth it.
- **Probabilistic methods**: Flipping a coin can be surprisingly helpful!

Connections: Random Graphs

Suppose we have a random graph with n vertices. How likely is it to be connected?

- What is a *random* graph?
  - If it has n vertices, there are $C(n, 2)$ possible edges, and $2^{C(n,2)}$ possible graphs. What fraction of them is connected?
  - One way of thinking about this. Build a graph using a random process, that puts each edge in with probability 1/2.

- Given three vertices a, b, and c, what's the probability that there is an edge between a and b and between b and c? 1/4
- What is the probability that there is no path of length 2 between a and c? $(3/4)^{n-2}$
- What is the probability that there is a path of length 2 between a and c? $1 - (3/4)^{n-2}$
- What is the probability that there is a path of length 2 between a and every other vertex? $> (1-(3/4)^{n-2})^{n-1}$

Now use the binomial theorem to compute $(1-(3/4)^{n-2})^{n-1}$

$$\begin{aligned} & (1-(3/4)^{n-2})^{n-1} \\ = \; & 1-(n-1)(3/4)^{n-2}+C(n-1,2)(3/4)^{2(n-2)}+\cdots \end{aligned}$$

For sufficiently large n, this will be (just about) 1.

Bottom line: If n is large, then it is almost certain that a random graph will be connected.

Theorem: [Fagin, 1976] If P is *any* property expressible in first-order logic, it is either true in almost all graphs, or false in almost all graphs.

This is called a *0-1 law*.

Connection: First-order Logic

Suppose you wanted to query a database. How do you do it?

Modern database query language date back to SQL (structured query language), and are all based on first-order logic.

- The idea goes back to Ted Codd, who invented the notion of relational databases.

Suppose you're a travel agent and want to query the airline database about whether there are flights from Ithaca to Santa Fe.

- How are cities and flights between them represented?
- How do we form this query?

You're actually asking whether there is a path from Ithaca to Santa Fe in the graph.

- This fact cannot be expressed in first-order logic!