Axiomatizing Arithmetic

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 nice (technically: recursive, so that a program can check whether a formula is an axiom) sound and complete axiomatization for arithmetic?

- *Gödel's Incompleteness Theorem:* NO!

Logic: The Big Picture

A typical logic is described in terms of

- *syntax*: what are the legitimate formulas
- *semantics*: under what circumstances is a formula true
- *proof theory/ axiomatization*: rules for proving a formula true Truth and provability are quite different.
- What is provable depends on the axioms and inference rules you use
- Provability is a mechanical, turn-the-crank process
- What is true depends on the semantics

Tautologies and Valid Arguments

When is an argument

A_1
A_2
$\vdots$
A_n

B

valid?

- When the truth of the premises implies the truth of the conclusion

How do you check if an argument is valid?

- Method 1: Take an arbitrary truth assignment v. Show that if $A_1, \ldots, A_n$ are true under v ($v \models A_1$, $\ldots v \models A_n$) then B is true under v.
- Method 2: Show that $A_1 \wedge \ldots \wedge A_n \Rightarrow B$ is a tautology (essentially the same as Method 1)
  - true for every truth assignment
- Method 3: Try to prove $A_1 \wedge \ldots \wedge A_n \Rightarrow B$ using a sound axiomatization

Graphs and Trees

Graphs and trees come up everywhere.

- We can view the internet as a graph (in many ways)
  - who is connected to whom
- Web search views web pages as a graph
  - Who points to whom
- Niche graphs (Ecology):
  - The vertices are species
  - Two vertices are connected by an edge if they compete (use the same food resources, etc.)

  Niche graphs give a visual representation of competitiveness.
- Influence Graphs
  - The vertices are people
  - There is an edge from a to b if a influences b

  Influence graphs give a visual representation of power structure.

There are lots of other examples in all fields ...

Terminology and Notation

A *graph* G is a pair (V, E), where E is a set of *vertices* or *nodes* and E is a set of *edges* or *branches*; an edge is a set $\{v, v'\}$ of two not necessarily distinct vertices (i.e., $v, v' \in V$).

- We sometimes write $G(V, E)$ instead of G
- If $V = \emptyset$, then $E = \emptyset$, and G is called the *null graph*.

We usually represent a graph pictorially.

- A vertex with no edges incident to it is said to be *isolated*
- If $\{v\} \in E$ (the book writes $\{v, v\}$), then there is a *loop* at v
- $G'(V', E')$ is a *subgraph* of $G(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

Directed Graphs

Note that $\{v, u\}$ and $\{u, v\}$ represent the same edge.

In a *directed graph* (*digraph*), the order matters. We denote an edge as (v, v') rather than $\{v, v'\}$. We can identify an undirected graph with the directed graph that has edges (v, v') and (v', v) for every edge $\{v, v'\}$ in the undirected graph.

Two vertices v and v' are *adjacent* if there is an edge between them, i.e., $\{v, v'\} \in E$ in the undirected case, $(v, v') \in E$ or $(v', v) \in E$ in the directed case.

Representing Relations Graphically

Given a relation R on $S \times T$, we can represent it by the directed graph $G(V, E)$, where

- $V = S \cup T$ and
- $E = \{(s, t) : (s, t) \in R\}$

Example: Represent the $<$ relation on $\{1, 2, 3, 4\}$ graphically.

How does the graphical representation show that a graph is

- reflexive?
- symmetric?
- transitive?

Multigraphs

In a *multigraph*, there may be several edges between two vertices.

- There may be several roads between two towns.
- There may be several transformations that can change you from one configuration to another
  - This is particularly important in graphs where edges are labeled

Formally, a multigraph $G(V, E)$ consists of a set V of vertices and a *multiset* E of edges

- The same edge can be in more than once

In this course, all graphs are *simple graphs* (not multigraphs) unless explicitly stated otherwise.

- Most of the results generalize to multigraphs

Degree

In a directed graph $G(V, E)$, the *indegree* of a vertex v is the number of edges coming into it

- indegree$(v) = |\{v' : (v', v) \in E\}|$

The *outdegree* of v is the number of edges going out of it:

- outdegree$(v) = |\{v' : (v, v') \in E\}|$

The *degree* of v, denoted deg(v), is the sum of the indegree and outdegree.

For an undirected graph, it doesn't make sense to talk about indegree and outdegree. The degree of a vertex is the sum of the edges incident to the vertex, except that we double-count all self-loops.

- Why? Because things work out better that way

Theorem: Given a graph $G(V, E)$,

$$2|E| = \sum_{v \in V} \deg(v)$$

Proof: For a directed graph: each edge contributes once to the indegree of some vertex, and once to the outdegree of some vertex. Thus $|E|$ = sum of the indegrees = sum of the outdegrees.

Same argument for an undirected graph without loops. We need to double-count the loops to make this right in general.

Handshaking Theorem

Theorem: The number of people who shake hands with an odd number of people at a party must be even.

Proof: Construct a graph, whose vertices are people at the party, with an edge between two people if they shake hands. The number of people person p shakes hands with is deg(p). Split the set of all people at the party into two subsets:

- A = those that shake hands with an even number of people
- B= those that shake hands with an odd number of people

$$\sum_p \deg(p) = \sum_{p \in A} \deg(p) + \sum_{p \in B} \deg(p)$$

- We know that $\Sigma_p \deg(p) = 2|E|$ is even.
- $\Sigma_{p \in A} \deg(p)$ is even, because for each $p \in A$, deg(p) is even.
- Therefore, $\Sigma_{p \in B} \deg(p)$ is even.
- Therefore $|B|$ is even (because for each $p \in B$, deg(p) is odd, and if $|B|$ were odd, then $\Sigma_{p \in B} \deg(p)$ would be odd).

Paths

Given a graph $G(V, E)$.

- A *path* in G is a sequence of vertices $(v_0, \ldots, v_n)$ such that $\{v_i, v_{i+1}\} \in E$ ((v_i, v_{i+1}) in the directed case).
- If $v_0 = v_n$, the path is a *cycle*
- An *Eulerian* path/cycle is a path/cycle that traverses every every edge in E exactly once
- A *Hamiltonian* path/cycle is a path/cycle that passes through each vertex in V exactly once.
- A graph with no cycles is said to be *acyclic*

Connectivity

- An undirected graph is *connected* if there is for all vertices u, v, $(u \neq v)$ there is a path from u to v.
- A digraph is *strongly connected* if for all vertices u, v $(u \neq v)$ there is a path from u to v and from v to u.
- If a digraph is *weakly connected* if, for every pair u, v, there is an edge from u to v *or* an edge from v to u.
- A *connected component* of an (undirected) graph G is a connected subgraph G' which is not the subgraph of any other connected subgraph of G.

Example: We want the graph describing the interconnection network in a parallel computer:

- the vertices are processors
- there is an edge between two nodes if there is a direct link between them.
  - if links are one-way links, then the graph is directed

We typically want this graph to be connected.

Trees

A *tree* is a digraph such that

(a) with edge directions removed, it is connected and acyclic

(b) every vertex but one, the *root*, has indegree 1

(c) the root has indegree 0

Trees come up everywhere:

- when analyzing games
- representing family relationships

Complete Graphs and Cliques

- An undirected graph $G(V, E)$ is *complete* if it has no loops and for all vertices u v $(u \neq v)$, $\{u, v\} \in E$.
  - How many edges are there in a complete graph with n vertices?

A complete subgraph of a graph is called a *clique*

- The *clique number* of G is the size of the largest clique in G.

The Konigsberg Bridge Problem

This is a classic mathematical problem.

There were seven bridges across the river Pregel at Königsberg.

Is it possible to take a walk in which each bridge is crossed exactly once?

Euler solved this problem in 1736.

- Key insight: represent the problem graphically

Eulerian Paths

Recall that $G(V, E)$ has an Eulerian path if it has a path that goes through every edge exactly once. It has an Eulerian cycle (or Eulerian circuit) if it has an Eulerian path that starts and ends at the same vertex.

How can we tell if a graph has an Eulerian path/circuit?

What's a necessary condition for a graph to have an Eulerian circuit?

Count the edges going into and out of each vertex:

- Each vertex must have even degree!

This condition turns out to be sufficient too.

Theorem: A connected (multi)graph has an Eulerian cycle iff each vertex has even degree.

Proof: The necessity is clear: In the Eulerian cycle, there must be an even number of edges that start or end with any vertex.

To see the condition is sufficient, we provide an algorithm for finding an Eulerian circuit in $G(V, E)$.

First step: Follow your nose to construct a cycle.

Second step: Remove the edges in the cycle from G. Let H be the subgraph that remains.

- every vertex in H has even degree
- H may not be connected; let $H_1, \ldots, H_k$ be its connected components.

Third step: Apply the algorithm recursively to $H_1, \ldots, H_k$, and then splice the pieces together.

Finding cycles

First, find an algorithm for finding a cycle:

Input: $G(V, E)$ [a list of vertices and edges]

procedure Pathgrow(V,E,v)
  [v is first vertex in cycle]
  $P \leftarrow ()$ [P is sequence of edges on cycle]
  $w \leftarrow v$ [w is last vertex in P]
  repeat until $I(w) - P = \emptyset$
    [$I(w)$ is the set of edges incident on w]
    Pick $e \in I(w) - P$
    $w \leftarrow$ other end of e
    $P \leftarrow P \cdot e$ [append e to P]
  endrepeat
  return P
endpro

Claim: If every vertex in V has even degree, then P will be a cycle

- Loop invariant: In the graph $G(V, E - P)$, if the first vertex (v) and last vertex (w) in P are different, they have odd degree; all the other vertices have even degree.

Finding Eulerian Paths

procedure Euler(V,E,v)
  //$G(V, E)$ is a connected undirected graph; $v \in V$ is arbitra
  //output is an Eulerian cycle in G//
Pathgrow(V',E',v') [returns cycle P in G]
if P is not Eulerian
  thendelete the edges in P from E;
    let $G_1(V_1, E_1), \ldots, G_n(V_n, E_n)$ be
      the resulting connected components
    let v_i be a vertex in V_i also on P
      for $i = 1$ **to** n
        Euler(V_i, E_i, v_i) [returns Eulerian cycle C_i
        Attach C_i to P at v_i
      endfor
    return P
  endpro

Corollary: A connected multigraph has an Eulerian path (but not an Eulerian cycle) if it has exactly two vertices of odd degree.

Which of these graphs have Eulerian paths:

Hamiltonian Paths

Recall that $G(V, E)$ has a Hamiltonian path if it has a path that goes through every vertex exactly once. It has a Hamiltonian cycle (or Hamiltonian circuit) if it has a Hamiltonian path that starts and ends at the same vertex.

There is no known easy characterization or algorithm for checking if a graph has a Hamiltonian cycle/path.

Which of these graphs have a Hamiltonian cycle?

Searching Graphs

Suppose we want to process data associated with the vertices of a graph. This means we need a systematic way of searching the graph, so that we don't miss any vertices.

There are two standard methods.

- Breadth-first search
- Depth-first search

It's best to think of these on a tree:

Breadth-first search would visit the nodes in the following order:

$$1, 2, 3, \ldots, 10$$

Depth-first search would visit the nodes in the following order:

$$1, 2, 4, 5, 7, 8, 11, 3, 6, 9, 10$$

Breadth-First Search

Input $G(V, E)$ [a connected graph]
v [start vertex]

Algorithm Breadth-First Search
visit v
$V' \leftarrow \{v\}$ [V' is the vertices already visited]
Put v on Q [Q is a queue]
repeat while $Q \neq \emptyset$
  $u \leftarrow head(Q)$ [$head(Q)$ is the first item on Q]
  for $w \in A(u)$ [$A(u) = \{w | \{u, w\} \in E\}$]
    if $w \notin V'$
    then visit w
      Put w on Q
      $V' \leftarrow V' \cup \{w\}$
    endif
  endfor
  Delete u from Q

The BFS algorithm basically finds a tree embedded in the graph.

- This is called the *BFS search tree*