From Mathematics to Generic Programming

Course Slides – Part 2 of 3

Version 1.0

October 5, 2015

Lecture 9

Introduction to Abstract Algebra

(Covers material from Sec. 6.1-6.4)

Abstract Algebra

- Branch of mathematics that deals with abstract entities called *algebraic structures*
  - Collections of objects that follow certain rules

Groups

A *group* is a set on which the following are defined:

- Operations: $x \circ y,\ x^{-1}$
- Constant: e

and for which the following axioms hold:

- Associativity $x \circ (y \circ z) = (x \circ y) \circ z$
- Identity $x \circ e = e \circ x = x$
- Cancellation $x \circ x^{-1} = x^{-1} \circ x = e$

Group Notes

- The constant e is the *identity element*
  - It is often written as "1" in multiplicative contexts
- The operation x^{-1} is the inverse
  - Applying group operation to an item and its inverse results in the identity element

Group Operation

- The group operation $\circ$ is binary, meaning it takes 2 arguments
- We often treat the operation as multiplication, writing $x \circ y$ as xy and $x \circ x$ as x^2
- The group operation is not necessarily commutative, i.e. it is not necessarily true that $x \circ y = y \circ x$.

Groups with Commutativity

- An *abelian group* is a group whose operation is commutative.
- An *additive group* is an abelian group where the group operation is addition.
- Additive groups are assumed to be commutative, even though the name doesn't explicitly say this.

With additive groups, we use different notation:

"+" for the group operation
"0" for identity element

Closure

- Groups are *closed* under the group operation.
  - This means if you apply the group operation to any two elements of the group, the result is another element of the group.
- Groups are closed under the inverse operation.
  - If you apply the inverse operation to any element of the group, the result is another element of the group.

Examples of Groups

- Additive group of integers
- Multiplicative group of nonzero remainders modulo 7
- Group of rearrangements of a deck of cards
- Multiplicative group of invertible matrices with real coefficients
- Group of rotations of the plane

Multiplicative Group of Integers?

- No, because the multiplicative inverse of most integers is not an integer.
- In other words, integers are not closed under multiplicative inverse.

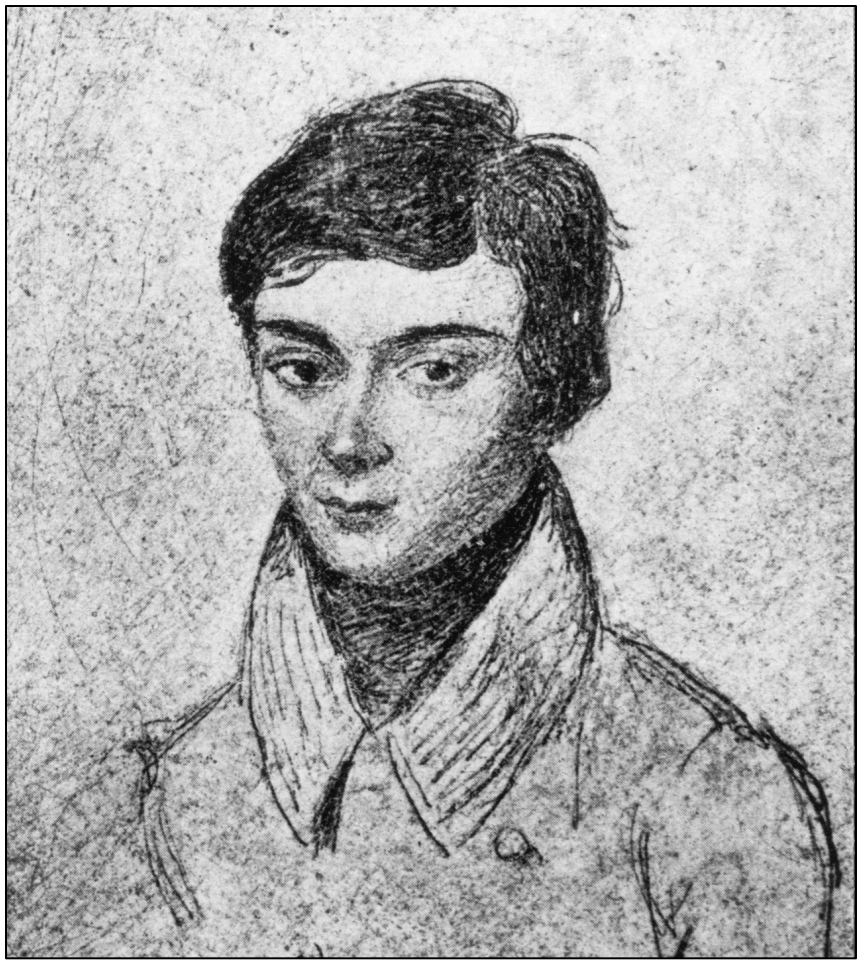
Multiplicative group of nonzero remainders mod 7 – what makes it a group?

×	1	2	3	4	5	6	
1	**1**	2	3	4	5	6	1
2	2	4	6	**1**	3	5	4
3	3	6	2	5	**1**	4	5
4	4	**1**	5	2	6	3	2
5	5	3	**1**	6	4	2	3
6	6	5	4	3	2	**1**	6

$\{1, 2, 3, 4, 5, 6\}$

- Has binary group operation (multiplication)
- Has an identity element (product of x and 1 is x)
- Closed under multiplication, e.g.
  $4 \circ 3 = (4 \text{ x } 3) \bmod 7 = 5$
- Closed under inverse, e.g.
  $5^{-1} = 3 \bmod 7$

Évariste Galois (1811-1832)



- Would-be revolutionary, expelled from school
- Studied math on his own
- Died in a pointless duel
- Wrote a letter that planted the seeds of group theory the night before his death

Monoids:
When we don't need inverse…

A *monoid* is a set on which the following are defined:

- Operations: $x \circ y$
- Constant: e

and for which the following axioms hold:

- Associativity $x \circ (y \circ z) = (x \circ y) \circ z$
- Identity $x \circ e = e \circ x = x$

Examples of Monoids

- Monoid of finite strings (free monoid)
  - Elements: strings
  - Operation: concatenation
  - Identity: empty string
- Multiplicative monoid of integers
  - Elements: integers
  - Operation: multiplication
  - Identity: 1

Semigroups:
When we don't need identity…

A *semigroup* is a set on which the following is defined:

- Operation: $x \circ y$

and for which the following axioms hold:

- Associativity $x \circ (y \circ z) = (x \circ y) \circ z$

Examples of Semigroups

- Additive semigroup of positive integers
  - Elements: positive integers
  - Operation: addition
- Multiplicative semigroup of even integers
  - Elements: even integers
  - Operation: multiplication

Raising Semigroup Element to a Power

Formal definition:

$$x^n = \begin{cases} x & \text{if } n = 1 \\ xx^{n-1} & \text{otherwise} \end{cases}$$

For $n > 2$, $xx^{n-1} = x^{n-1}x$

- Basis: $n = 2$. It's obviously true, because

$$xx^1 = xx = x^1x$$

- Induction hypothesis: Assume true for $n = k - 1$.

$$xx^{(k-1)-1} = xx^{k-2} = x^{k-2}x = x^{(k-1)-1}x$$

- Derive result for $n = k$:

$$\begin{aligned}
xx^{k-1} &= x(xx^{k-2}) && \text{by definition of power} \\
&= x(x^{k-2}x) && \text{by induction hypothesis} \\
&= (xx^{k-2})x && \text{by associativity of semigroup operation} \\
&= (x^{k-1})x && \text{by definition of power}
\end{aligned}$$

Commutativity of Powers:

$$x^n x^m = x^m x^n = x^{n+m}$$

- Basis $m = 1$:

$$\begin{aligned} x^n x &= x x^n && \text{by previous lemma} \\ &= x^{n+1} && \text{by definition of power} \end{aligned}$$

- Inductive step. Assume for $m = k$. Show for $m = k+1$:

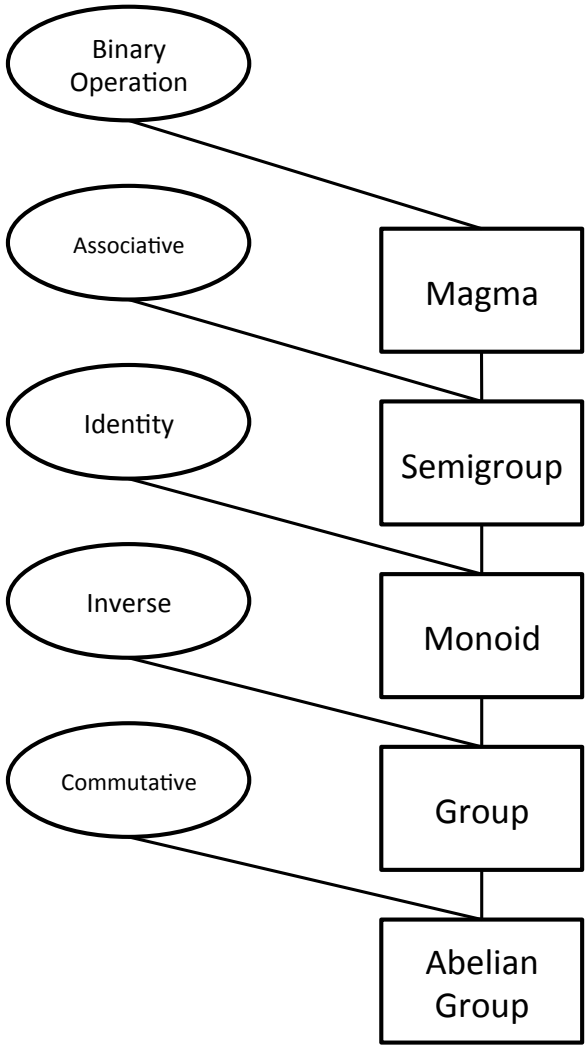
$$\begin{aligned} x^n x^{k+1} &= x^n (x x^k) && \text{by definition of power} \\ &= (x^n x) x^k && \text{by associativity of semigroup operation} \\ &= x^{n+1} x^k && \text{by previous lemma and definition of power} \\ &= x^{n+1+k} && \text{by inductive hypothesis} \\ &= x^{n+k+1} && \text{by commutativity of integer addition} \end{aligned}$$

- We have shown $x^n x^m = x^{n+m}$, so also $x^m x^n = x^{m+n}$.
- By commutativity of addition, $x^{n+m} = x^{m+n}$, so $x^n x^m = x^m x^n$.

Is there any algebraic structure weaker than semigroups?

- The only requirement left to relax is the associativity axiom.
- An algebraic structure called *magma* drops this axiom, but it's not very useful.
  - Without axioms, you can't derive any theorems.

Recap of Algebraic Structures



All groups are *transformation groups*

- Every element a of the group G defines a transformation of G onto itself:

$$x \longrightarrow ax$$

  For example, with the additive group of integers, if $a = 5$, then this acts as a "$+5$" operation, transforming each element x to $x + 5$.

- The transformations are one-to-one because of the invertibility axiom

$$a^{-1}(ax) \longrightarrow x$$

A group transformation is a one-to-one correspondence

Equivalently:

- For any finite set *S* of elements of group *G* and any element *a* of *G*, a set of elements *aS* has the same number of elements as *S*.

A group transformation is a one-to-one correspondence -- Proof

- If $S = \{s_1, \ldots, s_n\}$ then $aS = \{as_1, \ldots, as_n\}$.
- Suppose two elements of aS were the same: $as_i = as_j$. Then

$$\begin{aligned} a^{-1}(as_i) &= a^{-1}(as_j) \\ (a^{-1}a)s_i &= (a^{-1}a)s_j && \text{by associativity} \\ es_i &= es_j && \text{by cancellation} \\ s_i &= s_j && \text{by identity} \end{aligned}$$

- So if results of transformation as_k are equal, inupts s_k are equal. Equivalently, if inputs not equal, results not equal.
- Since we started with n distinct arguments, we have n distinct results. So $|aS| = |S|$.

There is a unique inverse for every element of a group.

$$ab = e \implies b = a^{-1}$$

Proof: Suppose there is some b that cancels a, i.e.

$$ab = e$$

Then we can multiply both sides by a^{-1} on the left:

$$ab = e$$

$$a^{-1}(ab) = a^{-1}e$$

$$(a^{-1}a)b = a^{-1}$$

$$eb = a^{-1}$$

$$b = a^{-1}$$

The inverse of a product is the reversed product of the inverses.

$$(ab)^{-1} = b^{-1}a^{-1}$$

Proof: The two expressions are equal iff multiplying one by the inverse of the other yields the identity.

We'll use the inverse of $(ab)^{-1}$, i.e. ab, and multiply it by $b^{-1}a^{-1}$:

$$\begin{aligned}(ab)(b^{-1}a^{-1}) &= a(bb^{-1})a^{-1} \\ &= aa^{-1} \\ &= e\end{aligned}$$

Power of inverse is inverse of power

$$(x^{-1})^n = (x^n)^{-1}$$

- Basis n = 1: $(x^{-1})^1 = x^{-1} = (x^1)^{-1}$
- Inductive step: Assume true for $n = k - 1$, that is, $(x^{-1})^{k-1} = (x^{k-1})^{-1}$. Show for $n = k$.

$$\begin{aligned} x^k(x^{-1})^k &= (xx^{k-1})(x^{-1}(x^{-1})^{k-1}) \\ &= (x^{k-1}x)(x^{-1}(x^{-1})^{k-1}) \\ &= x^{k-1}(xx^{-1})(x^{-1})^{k-1} \\ &= x^{k-1}(x^{-1})^{k-1} \\ &= x^{k-1}(x^{k-1})^{-1} \\ &= e \end{aligned}$$

- Therefore $(x^{-1})^n = (x^n)^{-1}$. QED.

Order of a group

- If a group has $n > 0$ elements, n is called the group's *order*.

- If a group has infinitely many elements, its order is infinite.

Order of an element

An element a has an *order* $n > 0$ if $a^n = e$ and for any $0 < k < n$, $a^k \neq e$.

In other words, the order of a is the smallest power of a that produces e.

If such n does not exist, a has an infinite order.

Every element of a finite group has a finite order

Proof: If n is order of the group, then for any element a, $\{a, a^2, a^3, \ldots, a^{n+1}\}$ has at least one repetition a_i and a_j.

Let us assume that $1 \leq i < j \leq n+1$, a_i is the first repeated element and a_j its first repetition. Then

$$a^j = a^i$$

$$a^j a^{-i} = a^i a^{-i} = e$$

$$a^{j-i} = e$$

and $j - i > 0$ is the order of a.

Subgroups

- A subset H of a group G is called a *subgroup* of G if:

$$e \in H$$

$$a \in H \implies a^{-1} \in H$$

$$a, b \in H \implies a \circ b \in H$$

- In other words, to be a subgroup, H must be a subset and a group.
- Example: The additive group of even numbers is a subgroup of the additive group of integers.

How many subgroups?

- Some groups have many subgroups.
- Almost all groups have at least two subgroups:
  - The group itself and the group consisting of only e.
  - These two are called *trivial* subgroups.
- The only group with a single subgroup is the one containing just the identity element.

Multiplicative group $\{1, 2, 3, 4, 5, 6\}$ of nonzero remainders modulo 7

$\times$	1	2	3	4	5	6	
1	**1**	2	3	4	5	6	1
2	2	4	6	**1**	3	5	4
3	3	6	2	5	**1**	4	5
4	4	**1**	5	2	6	3	2
5	5	3	**1**	6	4	2	3
6	6	5	4	3	2	**1**	6

- Four multiplicative subgroups:

$$\{1\},\ \{1, 6\},\ \{1, 2, 4\},\ \{1, 2, 3, 4, 5, 6\}$$

- How do we know? Each is a subset, and each is a group.

Cyclic Groups

A finite group is called *cyclic* if it has an element a such that for any element b there is an integer n where

$$b = a^n$$

In other words, every element can be generated by raising one particular element to different powers.

Such an element is called a *generator* of the group.

Multiplicative group of nonzero remainders modulo 7 is a cyclic group

- Recall that the subgroups of this group are:

$$\{1\}, \{1, 6\}, \{1, 2, 4\}, \{1, 2, 3, 4, 5, 6\}$$

- Generators of the group are 3 and 5, because they are not in any nontrivial subgroup.

Powers of a given element in a finite group form a subgroup

Proof: To be subgroup, must be nonempty subset and group.

- To be nonempty subset, must be closed under group operation. Yes, because product of two powers is a power.
- To be group, operation must be associative, must contain identity element, and must have an inverse operation.
  - Associative: Yes, because it's the same as the original operation.
  - Identity: Yes, because we showed earlier that every element of a finite group has a finite order.
  - Inverse: Yes, because for every power a^k, a^{n-k} is its inverse, where n is the order of a.

Things to Consider

- Organizing algebraic structures as a taxonomy makes their relationships clear, and enables the discovery of new structures
- The act of constructing a taxonomy forces you to develop a conceptual framework for the domain
- Taxonomies have been a source of insights in science since the time of Aristotle, and are still useful in programming

Lecture 10

Lagrange's Theorem

(Covers material from Sec. 6.5-6.8)

Abstract Algebra

Abstract algebra lets us prove results for structures such as groups *without knowing anything about either the items in the group or the operation*.

Cosets

If G is a group and $H \subset G$ is a subgroup of G, then for any $a \in G$ the **(left) coset** of a by H is a set

$$aH = \{g \in G \mid \exists h \in H \ : \ g = ah\}$$

In other words, a coset aH is a set of all elements in G obtainable by multiplying elements of H by a.

Examples of Cosets

Consider the additive group of integers $\mathbb{Z}$ and its subgroup, integers divisible by 4, $4\mathbb{Z}$.
It has four distinct cosets:

- 4n
- 4n+1
- 4n+2
- 4n+3

Adding other integers will result only in values already in one of these cosets.

Size of Cosets

In a finite group G, for any of its subgroups H,
the number of elements in a coset aH is the same
as the number of elements in the subgroup H.

Proof:

- Previously proved one-to-one correspondence of transformations aS when S is a subset of G.
- Since a subgroup is also a subset, the mapping from H to aH is also a one-to-one correspondence.
- If there is a one-to-one correspondence between two finite sets, they are the same size.

Complete Coverage by Cosets

Every element a of a group G
belongs to some coset of subgroup H.

Proof:

Every element a belongs to the coset aH generated by itself, because H, being a subgroup, contains the identity element.

Cosets are either disjoint or identical

If two cosets aH and bH in a group G have a common element c, then $aH = bH$.

Proof (start):

- Suppose the common element c is ah_a in one coset and bh_b in the other coset, i.e. $ah_a = bh_b$.
- Multiplying both sides on the right by h_a^{-1}, we get:

$$ah_a h_a^{-1} = bh_b h_a^{-1}$$

$$a = bh_b h_a^{-1}$$

$$a = b(h_b h_a^{-1})$$

Cosets are either disjoint or identical (proof continued)

$$a = b(h_b h_a^{-1})$$

- The term on the right is b times something from H.
- Multiply both sides on the right by x, an element of H:

$$ax = b(h_b h_a^{-1})x$$

- We know that ax is in the coset aH.
  We know that term on right is also in coset bH.
  Since we can do this for any x in H, $aH \subseteq bH$.
- Repeat from beginning, using h_b^{-1} to show $bH \subseteq aH$.
- So $aH = bH$.

Lagrange's Theorem

The order of a subgroup H of a finite group G divides the order of the group.

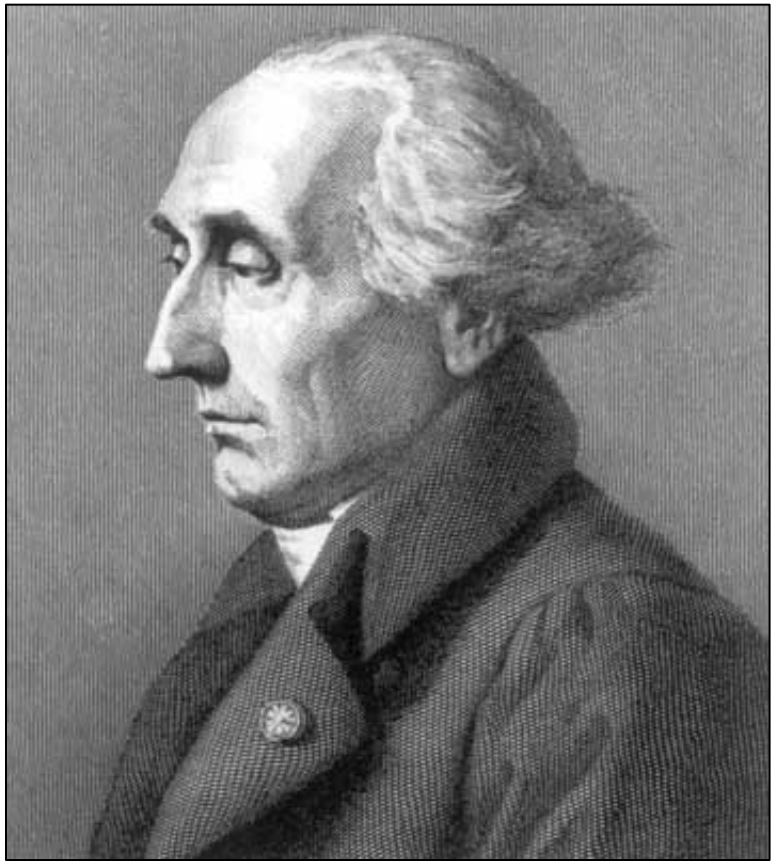
Proof:

- We already proved that:
  - The group G is covered by cosets of H.
  - Different cosets are disjoint.
  - They are the same size n, where n is order of H.
- Therefore the order of G is nm, where m is the number of distinct cosets, so order of G is a multiple of order of H.
- In other words, the order of H divides the order of G.

Lagrange's Theorem Example

- Suppose a group G has two distinct cosets of its subgroup H
- Then every element of G must be covered by one or the other (but not both) of the cosets
- So the order of H must be $|G|/2$.

Joseph-Louis Lagrange (1736-1813)



- Leading mathematician in Europe at the time
- Contributed to many areas of mathematics and physics
- Lived in Italy, Germany, and France
- Involved in creation of the metric system

Corollary 1 to Lagrange's Theorem

The order of any element in a finite group divides the order of the group.

Proof:

- The powers of an element of G form a subgroup of G.
- Since the order of an element is the order of the subgroup, and since the order of the subgroup must divide the order of the group, then the order of the element must divide the order of the group.

Corollary 2 to Lagrange's Theorem

Given a group G of order n, if a is an element of G, then $a^n = e$.

Proof:

- If a has an order m, then m divides n (by Corollary 1),
- So $n = qm$.
- $a^m = e$ (by definition of order of an element).
- Therefore $(a^m)^q = e$ and $a^n = e$.

Proving Fermat's Little Theorem with Lagrange's Theorem

If p prime, $a^{p-1} - 1$ is divisible by p for any $0 < a < p$

Proof:

- Take the multiplicative group of remainders mod p.
- It contains $p-1$ nonzero remainders, i.e. has order $p-1$, so from corollary 2:

$$a^{p-1} = e$$

- Since the identity element is 1:

$$a^{p-1} = 1 \bmod p$$

$$a^{p-1} - 1 = 0 \bmod p$$

which is what it means to be divisible by p.

Proving Euler's Theorem with Lagrange's Theorem

For $0 < a < n$, a and n coprime, $a^{\phi(n)} - 1$ is divisible by n

Proof:

- Take the multiplicative group of invertible remainders mod n.
- Since $\phi(n)$ is number of coprimes, and every coprime is invertible, $\phi(n)$ gives the order of the group.
- It follows from Corollary 2 that:

$$a^{\phi(n)} = e$$

$$a^{\phi(n)} = 1 \bmod n$$

- Or equivalently: $a^{\phi(n)} - 1 = 0 \bmod n$
  which is what it means to be divisible by n.

Abstract Algebra and Model Theory

- When we apply abstraction to things like remainders and modular arithmetic, we get *groups*
- We can also apply abstraction to things like groups, and get *theories*.

Theories

Definition: A *theory* is a set of true propositions.

- Can be generated by axioms + inference rules.
- A theory is *finitely axiomatizable* if it can be generated from a finite set of axioms.
- A set of axioms is *independent* if removing one will decrease the set of true propositions.
- A theory is *complete* if for any propositions, either that proposition or its negation is in the theory.
- A theory is *consistent* if for no proposition it contains both that proposition and its negation.

Groups as Theories

- Given operations $x \circ y$ and x^{-1} and the identity element e, the theory of a group has axioms:

$$x \circ (y \circ z) = (x \circ y) \circ z$$

$$x \circ e = e \circ x = x$$

$$x \circ x^{-1} = x^{-1} \circ x = e$$

- Starting with these, we can derive true propositions (theorems) such as:

$$x \circ y = x \implies y = e$$

$$(x \circ y)^{-1} = y^{-1} \circ x^{-1}$$

Observations about Theories

- Just because a theory is a set of true propositions doesn't mean we have to enumerate them.
  - We can generate them from axioms and previously proven propositions.
- Axioms form a basis for a theory
  - (like basis vectors in linear algebra)
- We can have more than one basis for the same theory.

Models

A set of elements where all the operations in the theory are defined and all the propositions in the theory are true is called its *model*.

- A model is like an implementation of a theory.
- A theory can have multiple models.
  - Example: the additive group of integers and the multiplicative group of remainders modulo 7 are both models of the theory of abelian groups.

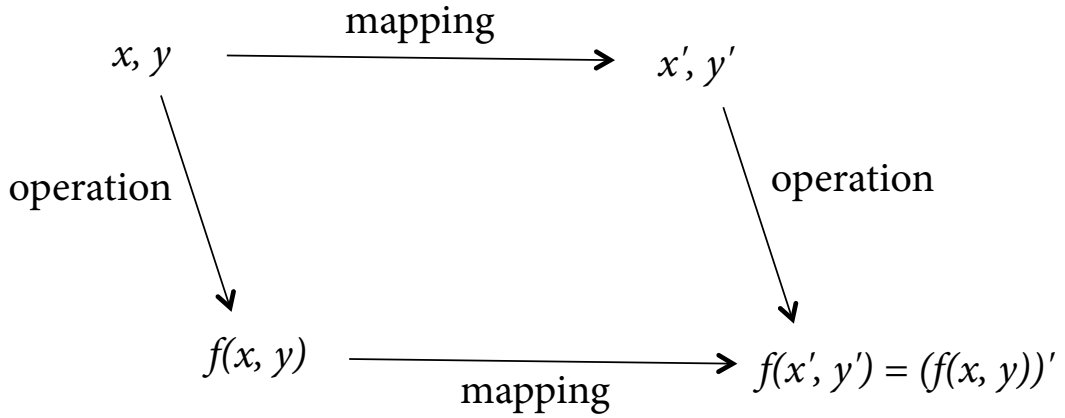
Number of Models for a Theory

- The more propositions there are in a theory, the fewer different models there are.
- Axioms and propositions are constraints on a theory
  - The more you have, the harder it is to satisfy all of them, so the fewer models there will be that do so.
  - Conversely, the more models there are for a theory, the fewer propositions there are.

Isomorphic Models

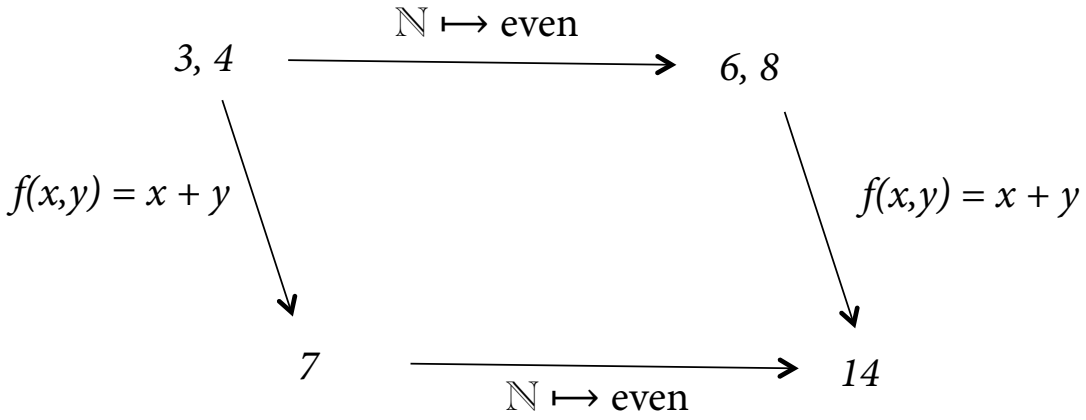
Two models are *isomorphic* if there is a one-to-one correspondence between them that preserves their operations.

This means we can apply the mapping (or its inverse) before or after the operation and we'll get the same result.



Isomorphic Models Example

- We can map natural numbers to even natural numbers with the mapping "multiply by 2" and use addition as our operation.
- If we add two numbers and then apply the mapping (multiply by 2), we get the same result as if we multiply by 2 and then add:



More About Theories and Models

- An isomorphism of a model with itself is called an *automorphism*.

- A (consistent) theory is called *categorical* or *univalent* if all its models are isomorphic.

- An inconsistent theory is one that has no models.
  - There's no way to satisfy all the propositions without a contradiction.

Categorical Theories vs. STL

- People used to believe that only categorical theories made sense for programming.
- When STL was proposed, some objected that some of its concepts, such as **Iterator**, were underspecified (i.e. not categorical).
- But STL's power comes from this generality:
  - E.g. linked lists and arrays are not isomorphic, but many STL algorithms work on both data structures.

Categorical Theory with 2 Isomorphic Models: Cyclic Groups of Order 4

	0	**1**	**2**	**3**
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

$\mathbb{Z}_4$

Additive group of remainders modulo 4

	1	**2**	**3**	**4**
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

$(\mathbb{Z}_5', \times)$

Multiplicative group of nonzero remainders modulo 5

Constraints on Mapping Models

- In theory, 4! = 24 possible mappings.
  - 0 in first model could map to 1, 2, 3, 4 in second, etc.
- An element is a *generator* if you can raise it to powers and get all other elements.
  - 1 and 3 are generators in first model
  - 2 and 3 are generators in second model
  - A generator in one model must map to generator in other.
- An identity element must map to an identity element
- 2 must map to 4, since it's the only non-identity element that gives identity when applied to itself.

Constraints Leave Two Possible Mappings

Value in $\mathbb{Z}_4$	Value in $\left(\mathbb{Z}_5', \times\right)$
0	1
1	2
2	4
3	3

Value in $\mathbb{Z}_4$	Value in $\left(\mathbb{Z}_5', \times\right)$
0	1
1	3
2	4
3	2

How do we know we can use these mappings to get our second model from our first?

Transforming Multiplication Table

1. Replace original values with mapped values:

	1	**3**	**4**	**2**
1	1	3	4	2
3	3	4	2	1
4	4	2	1	3
2	2	1	3	4

2. Swap last two columns and last two rows:

	1	**3**	**2**	**4**
1	1	3	2	4
3	3	4	1	2
2	2	1	4	3
4	4	2	3	1

Transforming Multiplication Table

3. Swap middle two rows and columns:

	1	**2**	**3**	**4**
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Non-Categorical Theory

- A non-categorical theory has multiple models that are not isomorphic.
- Example: Groups of order 4
- It has two non-isomorphic models
  - Cyclic group of order 4
  - Klein group

Klein Group

Two important models:

- Multiplicative group of units modulo 8
  - Consists of {1, 3, 5, 7}
- Group of isometries transforming a rectangle into itself
  - Consists of identity transform, vertical symmetry, horizontal symmetry, 180° rotation.

Non-Categorical Theory: All Groups of Order 4

	e	**a**	**b**	**c**
e	e	a	b	c
a	a	b	c	e
b	b	c	e	a
c	c	e	a	b

Cyclic group $\mathbb{Z}_4$

	e	**a**	**b**	**c**
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

Klein group

How do we know these are not isomorphic?

- There is a distinguishing proposition (differentia) that is true for the Klein group but false for Z_4:

$$\forall x \in G \ : \ x^2 = e$$

- Equivalently, the diagonals of the multiplication tables are different.
- The cyclic group contains two generators, while the Klein group contains none.

Things to Consider

- The multiplication table, a simple mathematical tool, continues to be a useful source of insights
  - You used it in 3rd grade for basic arithmetic
  - We used it to understand modular arithmetic
  - We used it to see how Euler extended Fermat's results
  - We used it to distinguish categorical from non-categorical theories

Lecture 11

Requirements for an Algorithm

(Covers material from Sec. 7.1-7.5)

Egyptian Multiplication Revisited: Coloring `multiply-accumulate`

```
int mult_acc4(int r, int n, int a) {
  while (true) {
    if (odd(n)) {
      r = r + a;
      if (n == 1) return r;
    }
    n = half(n);
    a = a + a;
  }
}
```

Observation:

No operations involve both blue and red variables.

Requirements on Types

Blue Variables *r* and *a*:

- Must support addition
  - "plusable"

We'll use template typename **A** for these variables.

Red Variable *n*

- Must be able to check for oddness
- Must be able to compare with 1
- Must support division by 2
  - "halvable"

We'll use template typename **N** for this variable.

More Generic Form

```
template <typename A, typename N>
A multiply_accumulate0(A r, N n, A a) {
    while (true) {
        if (odd(n)) {
            r = r + a;
            if (n == 1) return r;
        }
        n = half(n);
        a = a + a;
    }
}
```

Syntactic Requirements on **A**

```
template <typename A, typename N>
A multiply_accumulate0(A r, N n, A a) {
    while (true) {
        if (odd(n)) {
            r = r + a;
            if (n == 1) return r;
        }
        n = half(n);
        a = a + a;                    } }
```

- added (in C++, must have `operator+`)
- passed by value (in C++, must have copy constructor)
- assigned (in C++, must have `operator=`)

Semantic Requirements on **A**

The addition operator + must be associative:

$$A(T) \implies \forall a, b, c \in T : a + (b + c) = (a + b) + c$$

Isn't + always associative?

- Not necessarily, when on computers…

```
w = (x + y) + z;
w = x + (y + z);
```

- Suppose x, y, and z are ints, and z is negative.
- Then there are large values where the first computation would overflow, but the second wouldn't.
- We say that **+** is a partial function
  - Not defined for all values.

Solution to Partial Function Problem

- We require that our axioms hold only inside the *domain of definition* – the set of values for which the function is defined.

More Syntactic Requirements on **A**

Some of the requirements we already listed imply additional requirements.

- Copy construction means making a copy that is equal to the original, so we need to have a notion of what it means for items of type **A** to be equal:
  - They can be compared for *equality* (in C++, they must support `operator==`)
  - They can be compared for *inequality* (in C++, they must support `operator!=`)

More Semantic Requirements on **A**: Equational Reasoning

New semantic requirements come with the new syntactic requirements:

- Inequality is the negation of equality:

$$(a \neq b) \iff \neg(a = b)$$

- Equality is reflexive, symmetric, and transitive:

$$a = a$$

$$a = b \implies b = a$$

$$a = b \wedge b = c \implies a = c$$

- Equality requires substitutibility:

$$\text{for any function } f \text{ on T}, \quad a = b \implies f(a) = f(b)$$

Regular Types

Regular Types are those that behave in the "usual way."

Specifically, a regular type T is a type where the relationships between construction, assignment, and equality are the same as for built-in types.

Regular Types in STL

STL was designed for regular types:

- Regular types can be stored in STL containers
- STL containers are themselves regular types
- Regular types can be used in STL algorithms

Examples of Regular Type Behavior

- `T a(b); assert(a == b); unchanged(b);`
- `a = b;  assert(a == b); unchanged(b);`
- `T a(b);` is equivalent to `T a; a = b;`

All types that we use in this course are regular.

Formal Requirements on **A**

- Regular type
- Provides associative +

A is a *Noncommutative Additive Semigroup*

- Semigroup – because it has an associative binary operation
- Additive – because the operation is +
- Noncommutative – because we don't *require* that the operation be commutative
  - Nothing in our algorithm needs commutativity

Examples of Noncommutative Additive Semigroups

- Positive even integers
- Negative integers
- Real numbers
- Polynomials
- Planar vectors
- Boolean functions
- Line segments

Mathematical Naming Conventions

- If a set has one binary operation that is both associative and commutative, call it +
- If a set has one binary operation that is associative but not commutative, call it *
  - Often write a * b as ab

Naming Conflict

- String concatenation is not commutative
- Kleene introduced notation *ab* to indicate string concatenation
  - Follows mathematical naming convention
- Many programming languages use notation a + b to indicate string concatenation
  - Violates mathematical naming convention

Bad result: Now we don't know if + implies commutativity or not.

The Naming Principle

If we are coming up with a name for something, or overloading an existing name:

1. If there is an established term, use it.
2. Do not use an established term inconsistently with its accepted meaning.
3. If there are conflicting usages, the much more established one wins.

Another Unfortunate Violation of the Principle

- The name `vector` in STL was derived from a similar use in the earlier programming languages Scheme and Common Lisp.
- Unfortunately, this is inconsistent with the much older mathematical meaning of *vector*.
- The STL data structure should have been called `array`.

Naming Compromise

- Term “noncommutative additive semigroup” is a solution to this naming conflict.
- We normally assume + is commutative, but here we explicitly say we don’t need that requirement.

Specifying Type Requirements for **A**

```
template <NoncommutativeAdditiveSemigroup A,
          typename N>
A multiply_accumulate(A r, N n, A a) {
    while (true) {
        if (odd(n)) {
            r = r + a;
            if (n == 1) return r;
        }
        n = half(n);
        a = a + a;
    }
}
```

Concepts

- A set of requirements for a type is called a *concept*.
- Current versions of C++ do not support declaring concepts, but we can fake them like this:

```
#define NoncommutativeAdditiveSemigroup typename
```

Syntactic Requirements on **N**

```
template <typename A, typename N>
A multiply_accumulate0(A r, N n, A a) {
    while (true) {
        if (odd(n)) {
            r = r + a;
            if (n == 1) return r;
        }
        n = half(n);
        a = a + a;                    } }
```

- **N** must be regular and also implement:
  - `half`
  - `odd`
  - `== 1`
  - `== 0` (we don't need now, but we will in a minute)

Semantic Requirements on **N**

$$\text{even}(n) \implies \text{half}(n) + \text{half}(n) = n$$
$$\text{odd}(n) \implies \text{even}(n-1)$$
$$\text{odd}(n) \implies \text{half}(n-1) = \text{half}(n)$$
$$\text{Axiom:} \quad n \leq 1 \quad \vee \quad \text{half}(n) = 1 \quad \vee \quad \text{half}(\text{half}(n)) = 1 \vee \ldots$$

- What C++ types satisfy these requirements?
  - `uint8_t`, `int8_t`, `uint64_t`, etc.
- What's the concept they represent?
  - **Integer**

Fully Generic Version of Function

```
template <NoncommutativeAdditiveSemigroup A,
          Integer N>
A multiply_accumulate_semigroup(A r, N n, A a) {
    // precondition(n >= 0);
    if (n == 0) return r;
    while (true) {
        if (odd(n)) {
            r = r + a;
            if (n == 1) return r;
        }
        n = half(n);
        a = a + a;
    }
}
```

Fully Generic Version of Multiply

```
template <NoncommutativeAdditiveSemigroup A,
          Integer N>
A multiply_semigroup(N n, A a) {
    // precondition(n > 0);
    while (!odd(n)) {
        a = a + a;
        n = half(n);
    }
    if (n == 1) return a;
    return multiply_accumulate_semigroup(a,
                half(n - 1), a + a);
}
```

Why can't *n* be zero?

- We started with the assumption that we had only positive integers (since ancient Greeks had no zero)
- What should an additive semigroup multiplication function return when $n = 0$?
  - The value that doesn't change the result when the semigroup operation (addition) is applied.
  - But semigroups don't require an identity element, so there is no notion of "adding zero."

What if we want to use zero?

- We can add a requirement that there be an identity element e and an identity axiom:

$$x \circ e = e \circ x = x$$

- What structure adds identity to semigroup?
  - Monoid
- So we'll use a *noncommutative additive monoid*, where identity element is called "0" and identity axiom is:

$$x + 0 = 0 + x = x$$

Multiply For Monoids

```
template <NoncommutativeAdditiveMonoid A,
          Integer N>
A multiply_monoid(N n, A a) {
    // precondition(n >= 0);
    if (n == 0) return A(0);
    return multiply_semigroup(n, a);
}
```

What if we want to use negatives?

- "Multiplying by a negative" corresponds to *inverse*.
- We can add a requirement that there be an inverse operation x^{-1} and an cancellation axiom:

$$x \circ x^{-1} = x^{-1} \circ x = e$$

- What structure adds inverse to monoid?
  - Group
- So we'll use a *noncommutative additive group*, where inverse operation is unary minus and cancellation axiom is:

$$x + -x = -x + x = 0$$

Multiply For Groups

```
template <NoncommutativeAdditiveGroup A,
          Integer N>
A multiply_group(N n, A a) {
    if (n < 0) {
        n = -n;
        a = -a;
    }
    return multiply_monoid(n, a);
}
```

Turning Multiply into Power

Observation:

If we replace + with $*$
(thereby replacing doubling with squaring),
we can use our existing algorithm
to compute a^n instead of $n \cdot a$.

power_accumulate for Semigroups

```
template <MultiplicativeSemigroup A,
          Integer N>
A power_accumulate_semigroup(A r, A a, N n) {
    // precondition(n >= 0);
    if (n == 0) return r;
    while (true) {
        if (odd(n)) {
            r = r * a;
            if (n == 1) return r;
        }
        n = half(n);
        a = a * a;
    }
}
```

Power for Semigroups

```
template <MultiplicativeSemigroup A,
          Integer N>
A power_semigroup(A a, N n) {
    // precondition(n > 0);
    while (!odd(n)) {
        a = a * a;
        n = half(n);
    }
    if (n == 1) return a;
    return power_accumulate_semigroup(a,
                a * a, half(n - 1));
}
```

Power for Monoids

```
template <MultiplicativeMonoid A, Integer N>
A power_monoid(A a, N n) {
    // precondition(n >= 0);
    if (n == 0) return A(1);
    return power_semigroup(a, n);
}
```

Power for Groups

```
template <MultiplicativeGroup A, Integer N>
A power_group(A a, N n) {
    if (n < 0) {
        n = -n;
        a = multiplicative_inverse(a);
    }
    return power_monoid(a, n);
}
```

Identities and Inverses

- Just as we needed the identity 0 for our additive monoid, we need the identity 1 for our multiplicative monoid.
- Just as we needed the additive inverse (*unary minus)* for our additive group, we need the multiplicative inverse (*reciprocal)* for our multiplicative group.

```
template <MultiplicativeGroup A>
A multiplicative_inverse(A a) {
    return A(1) / a;
}
```

Things to Consider

- By analyzing the operations an algorithm needs to perform on its arguments, we can make it as general as possible
- By replacing just a single operation, we can transform an algorithm designed for one task to work on another

Lecture 12

Generalizing an Algorithm

(Covers material from Sec. 7.6-7.8)

Two Operations, Two Versions of Code

```
A multiply_semigroup(N n, A a) {
    // precondition(n > 0);
    while (!odd(n)) {
        a = a + a;
        n = half(n);
    }
   ...
```

```
A power_semigroup(A a, N n) {
    // precondition(n > 0);
    while (!odd(n)) {
        a = a * a;
        n = half(n);
    }
   ...
```

Do we need to rewrite the function every time we want to use the algorithm with another operation?

There must be a better way…

Alternative: Generalize the Operation

- We'll pass the operation itself as an argument to the function
- This is a common pattern in STL
- We'll still call the result "power" because it computes the semigroup operation repeatedly (even though operation might not be multiply)

e.g. $x \circ x \circ x \circ x \circ x = x^5$

Changes We Need

- Pass in a **SemigroupOperation** as an argument.
- Add a requirement that the domain of the operation be **A**.
- Change requirement on **A** to be **Regular** instead of **Semigroup**.
  - Because we don't know what kind of semigroup (additive, multiplicative, etc.) to make A anymore; it depends on the operation.

power_accumulate with Generic Operation

```
template <Regular A, Integer N, SemigroupOperation Op>
// requires (Domain<Op, A>)
A power_accumulate_semigroup(A r, A a, N n, Op op) {
    // precondition(n >= 0);
    if (n == 0) return r;
    while (true) {
        if (odd(n)) {
            r = op(r, a);
            if (n == 1) return r;
        }
        n = half(n);
        a = op(a, a);
    }
}
```

Power with Generic Semigroup Operation

```
template <Regular A, Integer N,
          SemigroupOperation Op>
// requires (Domain<Op, A>)
A power_semigroup(A a, N n, Op op) {
    // precondition(n > 0);
    while (!odd(n)) {
        a = op(a, a);
        n = half(n);
    }
    if (n == 1) return a;
    return power_accumulate_semigroup(a,
                op(a, a), half(n - 1), op);
}
```

What if we want a monoid operation?

- We don't know what identity element to use (e.g. 0, 1, or something else) because we don't know in advance what the operation will be.
- Solution:
  Obtain the identity from the operation.

Power with Generic Monoid Operation

```
template <Regular A, Integer N,
          MonoidOperation Op>
// requires(Domain<Op, A>)
A power_monoid(A a, N n, Op op) {
    // precondition(n >= 0);
    if (n == 0) return identity_element(op);
    return power_semigroup(a, n, op);
}
```

Examples of Identity Element Functions

Identity element function for +:

```
template <NoncommutativeAdditiveMonoid T>
T identity_element(std::plus<T>) {
    return T(0);
}
```

Identity element function for *:

```
template <MultiplicativeMonoid T>
T identity_element(std::multiplies<T>) {
    return T(1);
}
```

What if we want a group operation?

- We don't know what inverse operation to use (e.g. unary minus, reciprocal, or something else) because we don't know in advance what the operation will be.
- Solution:
  Obtain the inverse from the operation.

Power with Generic Group Operation

```
template <Regular A, Integer N,
          GroupOperation Op>
// requires(Domain<Op, A>)
A power_group(A a, N n, Op op) {
    if (n < 0) {
        n = -n;
        a = inverse_operation(op)(a);
    }
    return power_monoid(a, n, op);
}
```

Examples of Inverse Operation Functions

Inverse operation function for +:

```
template <AdditiveGroup T>
std::negate<T> inverse_operation(std::plus<T>) {
    return std::negate<T>();
}
```

Inverse operation function for *:

```
template <MultiplicativeGroup T>
reciprocal<T>
inverse_operation(std::multiplies<T>) {
    return reciprocal<T>();
}
```

Reciprocal

```
template <MultiplicativeGroup T>
struct reciprocal {
    T operator()(const T& x) const {
        return T(1) / x;
    }
};
```

This is our first example of a *function object:*

a C++ object that provides a function

- declared by `operator()`
- invoked by a function call
- using the name of the object as function name

Reduction

Power is one generic algorithm; *reduction* is another.

- In reduction, a binary operation is applied successively to each element of a sequence and its previous result.

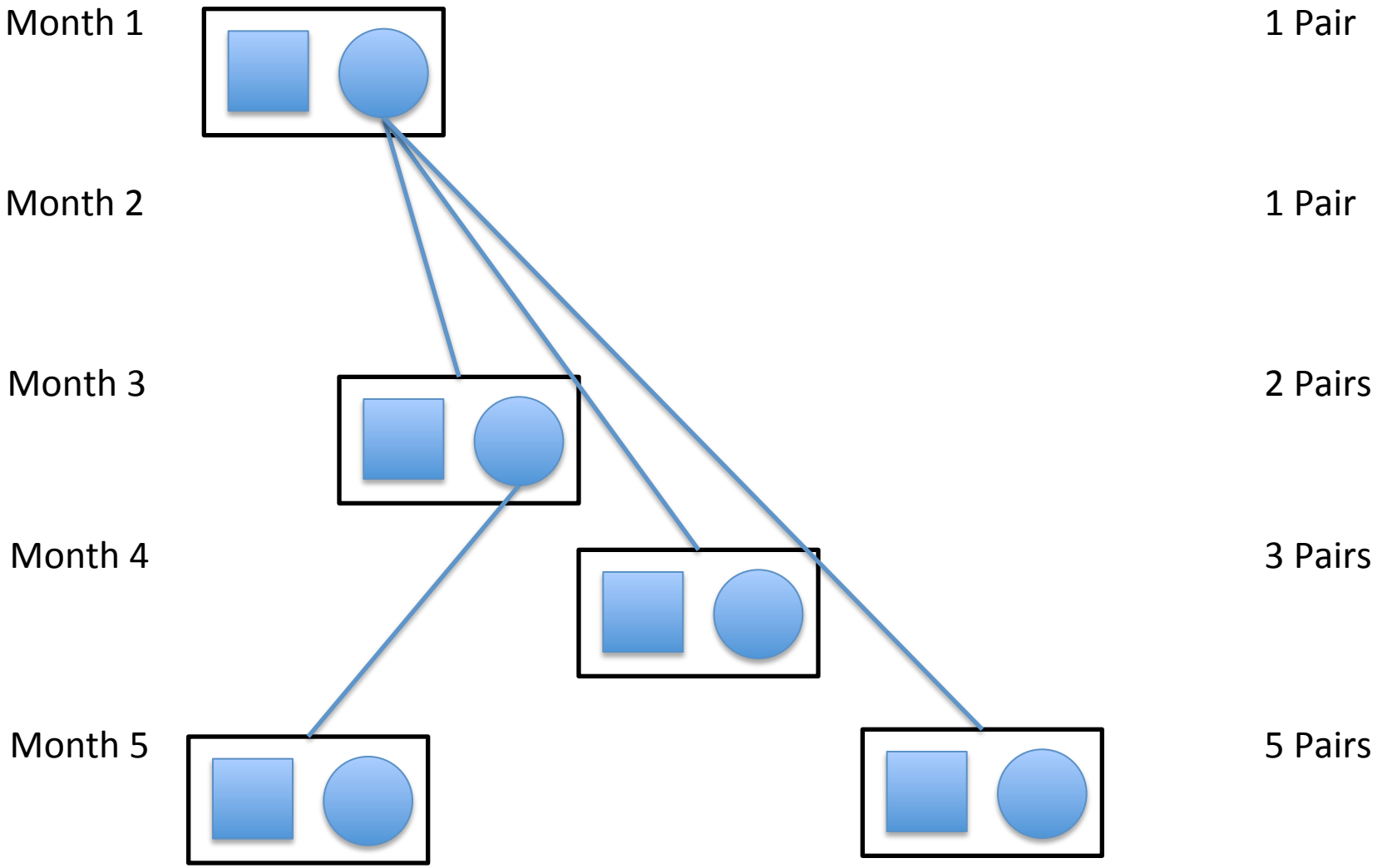
History of Reduction

- Mathematical sum (∑) and product (∏) operators
- 1962: APL "/" reduction operator
  - e.g. + / 1 2 3
- 1973: FP insert operator
- 1981: Tecton parallel reduction
- 1984: Common Lisp reduce function (as well as map)
- 2004: Google Map-Reduce paper
- 2005: Hadoop Map-Reduce

How Many Rabbits?

- Leonardo Pisano, aka Fibonacci, posed this question: If we start with one pair of rabbits, how many pairs will we have after *n* months?
- Simplifying assumptions:
  - Each pair has one male and one female
  - Females take one month to reach sexual maturity and have one litter per month after that
  - Rabbits live forever

Fibonacci's Rabbits



Fibonacci Numbers

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…
- Observation: Each value (each month's rabbit population) can be obtained by adding the previous two values.
- Formally:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_i = F_{i-1} + F_{i-2}$$

Naïve Fibonacci Function

```
int fib0(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fib0(n - 1) + fib0(n - 2);
}
```

Lots of Wasted Work

$$
\begin{aligned}
F_5 &= F_4 + F_3 \\
&= (F_3 + F_2) + (F_2 + F_1) \\
&= ((F_2 + F_1) + (F_1 + F_0)) + ((F_1 + F_0) + F_1) \\
&= (((F_1 + F_0) + F_1) + (F_1 + F_0)) + ((F_1 + F_0) + F_1)
\end{aligned}
$$

- 7 additions for this little example
- $F_1 + F_0$ is computed 3 times

Better: Keep a Running Total of Previous Two Results – O(*n*)

```
int fibonacci_iterative(int n) {
    if (n == 0) return 0;
    std::pair<int, int> v = {0, 1};
    for (int i = 1; i < n; ++i) {
        v = {v.second, v.first + v.second};
    }
    return v.second;
}
```

Another Approach: Use Matrix

- This matrix equation goes from one Fibonacci number to the next:

$$\begin{bmatrix} v_{i+1} \\ v_i \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} v_i \\ v_{i-1} \end{bmatrix}$$

- So the *n*th Fibonacci number is obtained by :

$$\begin{bmatrix} v_n \\ v_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Fibonacci Computation = Raising Matrix to Power

- We already have a generic function to raise something to a power
- And it's O(log *n*)

Generic Matrix-to-Power Methods

- A *linear recurrence* function of order k is a function f such that:

$$f(y_0, \ldots, y_{k-1}) = \sum_{i=0}^{k-1} a_i y_i$$

- A *linear recurrence sequence* is a sequence generated by a linear recurrence from initial k values.
  - The Fibonacci sequence is a linear recurrence sequence of order 2.
- Can replace "+" in the original definition of the Fibonacci sequence with an arbitrary linear recurrence function, to compute any linear recurrence.

Computing Arbitrary Linear Recurrence Sequence Using Power

$$\begin{bmatrix} x_n \\ x_{n-1} \\ x_{n-2} \\ \vdots \\ x_{n-k+1} \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{k-2} & a_{k-1} \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}^{n-k+1} \begin{bmatrix} x_{k-1} \\ x_{k-2} \\ x_{k-3} \\ \vdots \\ x_0 \end{bmatrix}$$

The line of 1s just below the diagonal provides the "shifting" behavior, so that each value in the sequence depends on the previous k values.

Things to Consider

- An algorithm can be generalized by passing in a key operation as an argument, rather than hardcoding it.
- An efficient general algorithm can be used to solve a variety of seemingly unrelated problems.

Lecture 13

Extending the Notion of Numbers

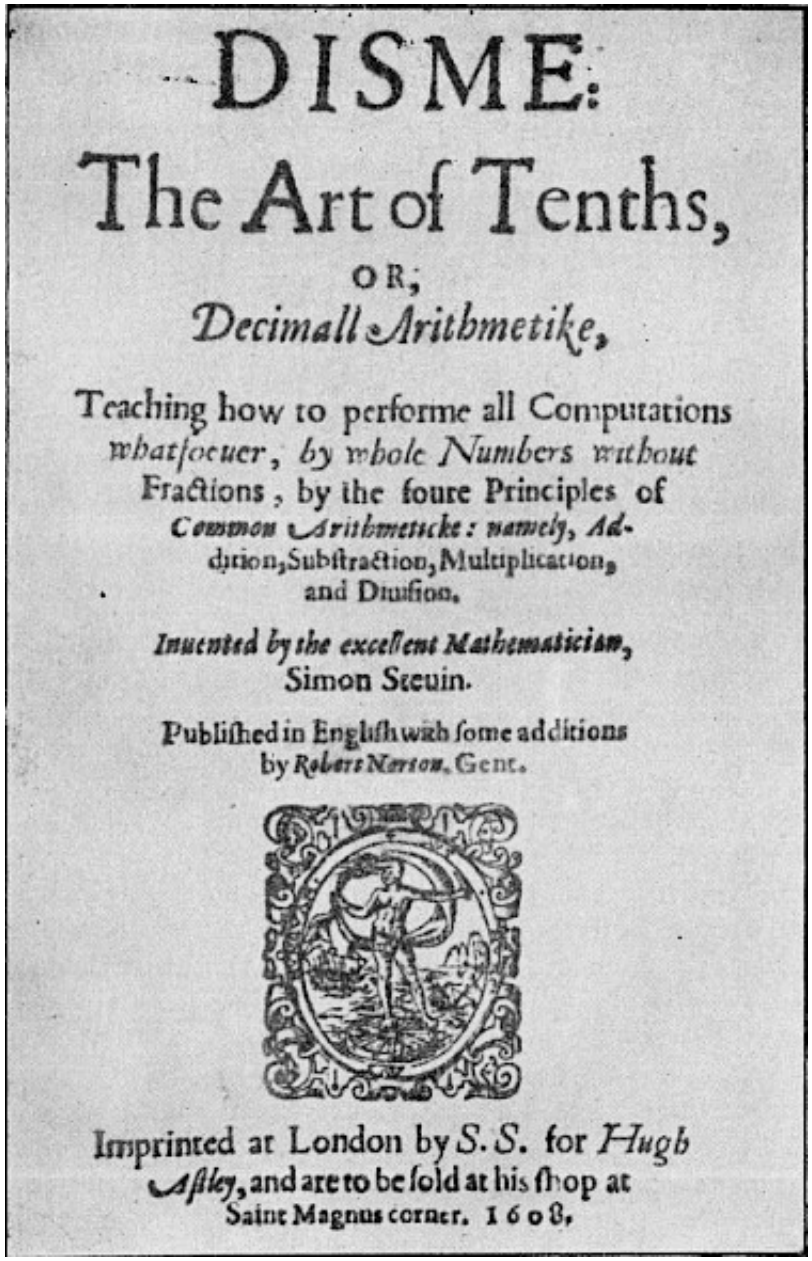
(Covers material from Sec. 8.1-8.2)

Euclid's GCD Algorithm, Revisited

- The original algorithm was for greatest common measure of line segments
- We extended it to greatest common divisor for integers
- Can it be extended further?

Simon Stevin (1548 – 1620)

- Introduced decimals in his book *De Thiende* ("The Tenth")
- Proposed parallelogram of forces in physics
- Discovered frequency ratio of adjacent notes in a scale
- A Dutch patriot who defended the country by selectively flooding lands with sluices



DISME:

The Art of Tenths,

OR,

Decimall Arithmetike,

Teaching how to performe all Computations *whatsoeuer, by whole Numbers without* Fractions, by the foure Principles of *Common Arithmeticke: namely,* Addition, Substraction, Multiplication, and Diuision.

Inuented by the excellent Mathematician, Simon Steuin.

Published in English with some additions by *Robert Norton*, Gent.

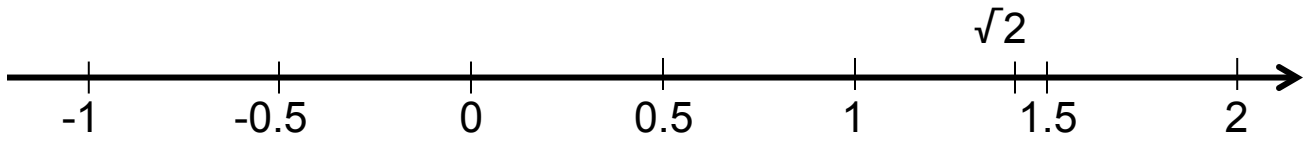
Imprinted at London by *S. S.* for *Hugh Astley*, and are to be sold at his shop at Saint Magnus corner. 1608.

Stevin expanded the notion of numbers from integers and fractions to "*that which expresseth the quantitie of each thing*"

Stevin Invented the Number Line

Any quantity could go on the number line, including:

- negative numbers
- irrational numbers
- “inexplicable” numbers (perhaps transcendental)



Advantages and Disadvantages

- Stevin was able to solve previously unsolvable problems, such as computing cube roots.
- He realized that results could be computed to whatever level of accuracy was desired.

- But decimal notation can take an infinite number of digits to express a simple value:

$$\frac{1}{7} = 0.142857142857142857142857142857...$$

Stevin's Next Breakthrough: Polynomials (*Arithmetique*, 1585)

$$4x^4 + 7x^3 - x^2 + 27x - 3$$

- Before Stevin, constructing a polynomial meant executing an algorithm:
  - Take a number, raise it to the 4th power, multiply it by 4, etc.
- Steven realized it's simply a finite sequence of numbers:

$$\{4, 7, -1, 27, -3\}$$

Horner's Rule

Use associativity to ensure that we never have to multiply powers of x higher than 1.

$$4x^4 + 7x^3 - x^2 + 27x - 3 = (((4x + 7)x - 1)x + 27)x - 3$$

For a polynomial of degree n,
we need n multiplications and $n - m$ additions,
where m is the number of coefficients equal to zero.

Polynomial Evaluation Function Using Horner's Rule

```
template <InputIterator I, Semiring R>
R polynomial_value(I first, I last, R x) {
    if (first == last) return R(0);
    R sum(*first);
    while (++first != last) {
        sum *= x;
        sum += *first;
    }
    return sum;
}
```

Requirements for Variables in Polynomial Evaluation Function

- **R** is a semiring
  - the type of variable x in the polynomial
  - a semiring is a structure where multiplication distributes over addition
- **I** is an Iterator
  - a generalized pointer
  - we want to iterate over the sequence of coefficients
- Value type of Iterator can be an entirely different type, such as a matrix.

Stevin's Polynomials and Arithmetic Operations

- Addition and subtraction:
  Add or subtract corresponding coefficients.
- Multiplication:
  Multiply every combination of elements, i.e. if a_i and b_i are ith coefficients of multiplicands and c_i is the ith coefficient of product, then:

$$
\begin{aligned}
c_0 &= a_0 b_0 \\
c_1 &= a_0 b_1 + a_1 b_0 \\
c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \\
&\vdots \\
c_k &= \sum_{k=i+j} a_i b_j
\end{aligned}
$$

Degree of Polynomials

- The *degree* of a polynomial deg(p) is the index of the highest nonzero coefficient
  - Equivalently, the highest power of the variable

- Examples:

$$\deg(5) = 0$$
$$\deg(x + 3) = 1$$
$$\deg(x^3 + x - 7) = 3$$

Polynomial Division

Polynomial a is divisible by polynomial b with remainder if there are polynomials q and r such that:

$$a = bq + r \quad \wedge \quad \deg(r) < \deg(b)$$

where q represents the quotient of $a \div b$.

Polynomial Division Example

- Procedure is just like traditional long division:

$$
\begin{array}{r}
 & 3x^2 & +2x & -2 \\
\cline{2-5}
x-2 \;\big|\; & 3x^3 & -4x^2 & -6x & +10 \\
 & 3x^3 & -6x^2 & & \\
\cline{2-3}
 & & 2x^2 & -6x & \\
 & & 2x^2 & -4x & \\
\cline{3-4}
 & & & -2x & +10 \\
 & & & -2x & +4 \\
\cline{4-5}
 & & & & 6
\end{array}
$$

Stevin's Observation

An algorithm in one domain can be applied in another similar domain.

- This is one of the core principles of generic programming.
- In particular, Stevin realized that Euclid's GCD algorithm would work for polynomials.

GCD for Polynomials

```
polynomial<real> gcd(polynomial<real> a,
                     polynomial<real> b) {
    while (b != polynomial<real>(0)) {
        a = remainder(a, b);
        std::swap(a, b);
    }
    return a;
}
```

How Do We Know Polynomial GCD Algorithm Works?

1. Must show it terminates.
   - i.e. computes GCD of polynomial in finite number of steps
   - Since it performs polynomial remainder, we know by definition of degree that

$$\deg(r) < \deg(b)$$

   - So degree is reduced at every step. Since degree is a non-negative integer, sequence is finite.
2. Must show it computes desired function.
   - Use same logic from Chapter 4.

Further Generalizations

- We've extended Euclid's algorithm from line segments to integers to polynomials to complex numbers.
- Can we go further?
  - To find out, we need to visit the center of mathematical innovation in the 18^{th} and 19^{th} centuries…

Golden Age of German Culture (18^{th} and 19^{th} centuries)

- Music
  - Bach, Händel, Haydn, Mozart, Beethoven, Schubert, Schumann, Brahms, Wagner, Mahler, Richard Strauss
- Literature
  - Goethe, Schiller
- Philosophy
  - Leibniz, Kant, Hegel, Schopenhauer, Marx, Nietzsche

Golden Age of the German Professor

- Commitment to the truth
- Scientist as a professional civil servant
- Collegial spirit, helping other colleagues
- Professor-student bond
- Continuity of research

Wir müssen wissen — wir werden wissen!

We must know — we will know!

Golden Age of the University of Göttingen

- Mathematics:
  - Gauss
  - Riemann
  - Dirichlet
  - Dedekind
  - Klein
  - Hilbert
  - Minkowski
  - Courant
  - Noether
- Physics:
  - Max Born
  - Heisenberg
  - Oppenheimer (visited)

Carl Friedrich Gauss (1777-1855)

- Child prodigy
- At 24, wrote foundational number theory treatise, *Disquisitiones Arithmeticae*
- Contributions to mathematics, physics, astronomy, and statistics
- Known as “Prince of Mathematicians”

Complex Numbers

- Imaginary numbers (xi where $i^2 = -1$) used since the 16th century, but never understood.
- In 1831 Gauss realized that numbers of form
$$z = x + yi$$
could be viewed as points (x,y) on Cartesian plane.
- These complex numbers were as valid as any other type of number.

Complex Number Definitions

complex number:	$z = x + yi$
complex conjugate:	$\bar{z} = x - yi$
real part:	$\mathrm{Re}(z) = \frac{1}{2}(z + \bar{z}) = x$
imaginary part:	$\mathrm{Im}(z) = \frac{1}{2i}(z - \bar{z}) = y$
norm:	$\|z\| = z\bar{z} = x^2 + y^2$
absolute value:	$\lvert z\rvert = \sqrt{\|z\|} = \sqrt{x^2 + y^2}$
argument:	$\arg(z) = \phi$ such that $0 \le \phi < 2\pi$ and $\frac{z}{\lvert z\rvert} = \cos(\phi) + i\sin(\phi)$

- Absolute value of z is length of vector z on complex plane.
- Argument is angle between real axis and vector z.
- Example: $\lvert i \rvert = 1$ and $\arg(i) = 90^\circ$

Arithmetic for Complex Numbers

addition: $z_1 + z_2 = (x_1 + x_2) + (y_1 + y_2)i$

subtraction: $z_1 - z_2 = (x_1 - x_2) + (y_1 - y_2)i$

multiplication: $z_1 z_2 = (x_1 x_2 - y_1 y_2) + (x_2 y_1 + x_1 y_2)i$

reciprocal: $\frac{1}{z} = \frac{\bar{z}}{\|z\|} = \frac{x}{x^2+y^2} - \frac{y}{x^2+y^2}i$

- Multiplication can also be done by adding arguments and multiplying absolute values.
  - Example: To find $\sqrt{i}$, we know it will have an absolute value of 1 and an argument of 45° (since 1 * 1 = 1 and 45 + 45 = 90°)

Gaussian Integers

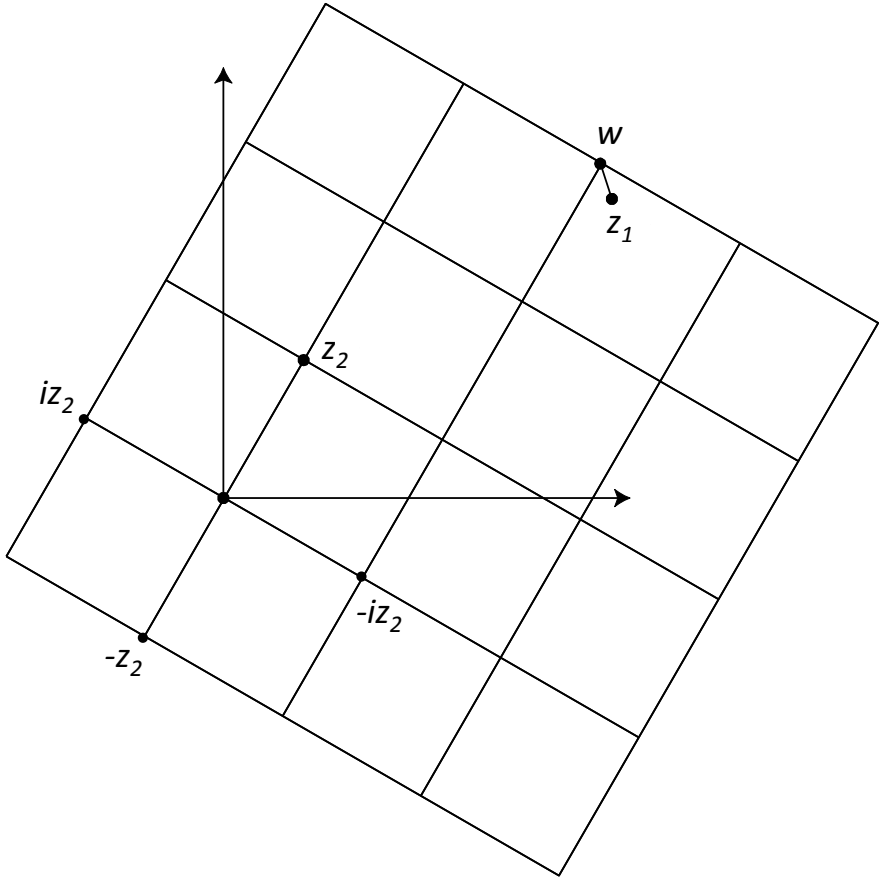
Complex numbers with integer coefficients

Interesting properties:

- Gaussian integer 2 is not prime, since it is the product of two other Gaussian integers, $1+i$ and $1-i$
- We can't do full division, but we can do division with remainder.

Remainder of z_1 and z_2

1. Construct a grid on the complex plane generated by z_2, iz_2, $-iz_2$ and $-z_2$
2. Find a square in the grid containing z_1
3. Find a vertex w of the square closest to z_1
4. $z_1 - w$ is the remainder.



GCD for Gaussian Integers

```
complex<integer> gcd(complex<integer> a,
                     complex<integer> b) {
    while (b != complex<integer>(0)) {
        a = remainder(a, b);
        std::swap(a, b);
    }
    return a;
}
```

Dirichlet's Extensions

- Peter Gustav Lejeune-Dirichlet (another Göttingen professor) realized that Gauss's complex numbers

$$t + n\sqrt{-1}$$

were a special case of

$$t + n\sqrt{-a}$$

- GCD works when $a = 1$, but fails when $a = 5$, since some numbers no longer have unique factorization, e.g.

$$21 = 3 \cdot 7 = (1 + 2\sqrt{-5}) \cdot (1 - 2\sqrt{-5})$$

Result about Euclid's GCD Algorithm

If Euclid's algorithm works in a certain domain, then all nonzero elements of the domain have a unique factorization.

Dedekind's Contributions

- Dirichlet's work was written up by his student Richard Dedekind, in a book called *Vorlesungen über Zahlentheorie* ("Lectures on Number Theory").
- Dedekind published the book under Dirichlet's name, even though Dedekind added his own results.

Dedekind's Generalization

Both Gaussian integers and Dirichlet's extensions are special cases of *algebraic integers*:

- linear integral combinations of roots of *monic polynomials*

$x^2 + 1$ generates Gaussian integers $a + b\sqrt{-1}$
$x^3 - 1$ generates *Eisenstein integers* $a + b\frac{-1+i\sqrt{3}}{2}$
$x^2 + 5$ generates integers $a + b\sqrt{-5}$

Things to Consider

"…the whole structure of number theory rests on a single foundation, namely the algorithm for finding the greatest common divisor of two numbers.

All the subsequent theorems … are still only simple consequences of the result of this initial investigation…"

– Dedekind, *Lectures on Number Theory*

Lecture 14

More Algebraic Structures

(Covers material from Sec. 8.3-8.9)

Generalizing Requirements for Euclid's Algorithm

- We keep finding more and more types where we can use Euclid's algorithm.
- Can we characterize the most general type for which the algorithm still works?

Emmy Noether (1882-1935)

- One of the greatest mathematicians of the 20th century
- A wonderful mentor to many students
- Taught courses at Göttingen, but originally – since she was a woman – under Hilbert's name
- Fled Nazi Germany, but no major U.S. university hired her

Noether's Insight

It is possible to derive results about certain kinds of mathematical entities without knowing anything about the entities themselves.

Noether taught mathematicians to always look for the most general setting of any theorem.

Noether and Generic Programming

- In programming terms, Noether's insight says that we can write algorithms and data structures in terms of concepts, without caring what types will be used.
- The generic programming approach is a direct result of her work.

Modern Algebra

- A book written by Noether's disciple Bartel van der Waerden, based on (and crediting) her lectures.
- Described Noether's abstract approach.
- Revolutionized the way modern math is studied and presented.

Rings

A *ring* is a set on which the following are defined:

- Operations: $x + y, -x, xy$
- Constants: $0_R, 1_R$

and for which the following axioms hold:

$$x + (y + z) = (x + y) + z$$
$$x + 0 = 0 + x = x$$
$$x + -x = -x + x = 0$$
$$x + y = y + x$$
$$x(yz) = (xy)z$$
$$1 \neq 0$$
$$1x = x1 = x \qquad 0x = x0 = 0$$
$$x(y + z) = xy + xz \qquad (y + z)x = yx + zx$$

Rings Are an Abstraction of Integers

They have operators that *act* like addition and multiplication:

- "addition" operator is commutative
- "multiplication" distributes over addition
- "addition" must have an inverse, but "multiplication" does not

Other Rings Besides Integers

- $n \times n$ matrices with real coefficients
- Gaussian integers
- Polynomials with integer coefficients
- …

Commutativity of Rings

- We say that a ring is commutative if

$$xy = yx$$

- Noncommutative rings come from linear algebra (where matrix multiplication does not commute)
- Polynomial rings and rings of algebraic integers do commute
- Two branches of abstract algebra, commutative algebra and noncommutative algebra, depending on type of ring

Invertibility

An element x of a ring is called *invertible* if there is an element x^{-1} such that

$$xx^{-1} = x^{-1}x = 1$$

An invertible element of a ring is called a *unit* of that ring.

- Every ring contains at least one invertible element (1).
- There may be more than one invertible element.
  - In ring of integers, both 1 and –1 are invertible

Units Are Closed Under Multiplication i.e. a product of units is a unit

Proof: Suppose a is a unit and b is a unit.

- Then (by definition of units) $aa^{-1} = 1$ and $bb^{-1} = 1$.
- So:

$$1 = aa^{-1} = a \cdot 1 \cdot a^{-1} = a(bb^{-1})a^{-1} = (ab)(b^{-1}a^{-1})$$

- Similarly, $a^{-1}a = 1$ and $b^{-1}b = 1$, so:

$$1 = b^{-1}b = b^{-1} \cdot 1 \cdot b = b^{-1}(a^{-1}a)b = (b^{-1}a^{-1})(ab)$$

- We now have a term that, when multiplied by ab from either side, gives 1.
- That term is the inverse of ab:

$$(ab)^{-1} = b^{-1}a^{-1}$$

- So ab is a unit.

Zero Divisor

An element x of a ring is called a *zero divisor* if:

1. $x \neq 0$
2. There exists a $y \neq 0,\ xy = 0.$

Example:

- In the ring of remainders modulo 6,
  2 and 3 are zero divisors.

Integral Domain

A commutative ring is called an *integral domain* if it has no zero divisors.

Examples:

- Integers
- Gaussian integers
- Polynomials over integers
- Rational functions over integers, such as $\frac{x^2+1}{x^3-2}$

Matrix Multiplication and Semirings

- Earlier we saw how to use our power algorithm with matrix multiplication to create an O(log n) Fibonacci function.
- Now we're going to see how to generalize this idea further.

First, a quick review of linear algebra…

Inner Product of Two Vectors

- Sum of the products of all the corresponding elements
- Always a scalar (a single number)

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i$$

Matrix-Vector Product

$$\vec{w} = \left[x_{ij}\right] \vec{v}$$

$$w_i = \sum_{j=1}^{n} x_{ij} v_j$$

- Multiplying an $n \times m$ matrix with an m-length vector results in an n-length vector
- The ith element of the result is the inner product of the ith row of the matrix with the original vector

Matrix-Matrix Product

$$[z_{ij}] = [x_{ij}]\ [y_{ij}]$$

$$z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}$$

- In the matrix product AB = C, if A is a $k \times m$ matrix and B is a $m \times n$ matrix, then C will be a $k \times n$ matrix.
- The element in row i and column j of C is the inner product of the ith row of A and the jth column of B.
- Matrix multiplication is not commutative
  - There is no guarantee that AB = BA
  - Often only one of the two products is well-defined

Generalizing Matrix Multiplication

Normally think of as product of sums, but really just need two operations that behave a certain way:

- "plus-like" operation $\oplus$
  - associative and commutative
- "times-like" operation $\otimes$
  - associative
  - distributes over the first operation:

$$a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$$

$$(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$$

Rings Are More Than We Need

- Rings satisfy the requirements
  - They have plus-like and times-like operators where one distributes over the other
- But they also have things we don't need
  - Additive inverse

Semiring: A Ring without Minus

A *semiring* is a set on which the following are defined:

- Operations: $x + y$, xy
- Constants: 0_R, 1_R

and for which the following axioms hold:

$$x + (y + z) = (x + y) + z$$

$$x + 0 = 0 + x = x$$

$$x + y = y + x$$

$$x(yz) = (xy)z$$

$$1 \neq 0$$

$$1x = x1 = x \qquad 0x = x0 = 0$$

$$x(y + z) = xy + xz \qquad (y + z)x = yx + zx$$

Canonical Semiring: Natural Numbers

- Natural numbers do not have additive inverses
- Matrix multiplication on matrices with natural number coefficients makes perfect sense

Sample Graph Problem: Social Network

- If you're friends with X, X is friends with you
- Mathematically, network is an undirected graph
- Represent as a connectivity matrix

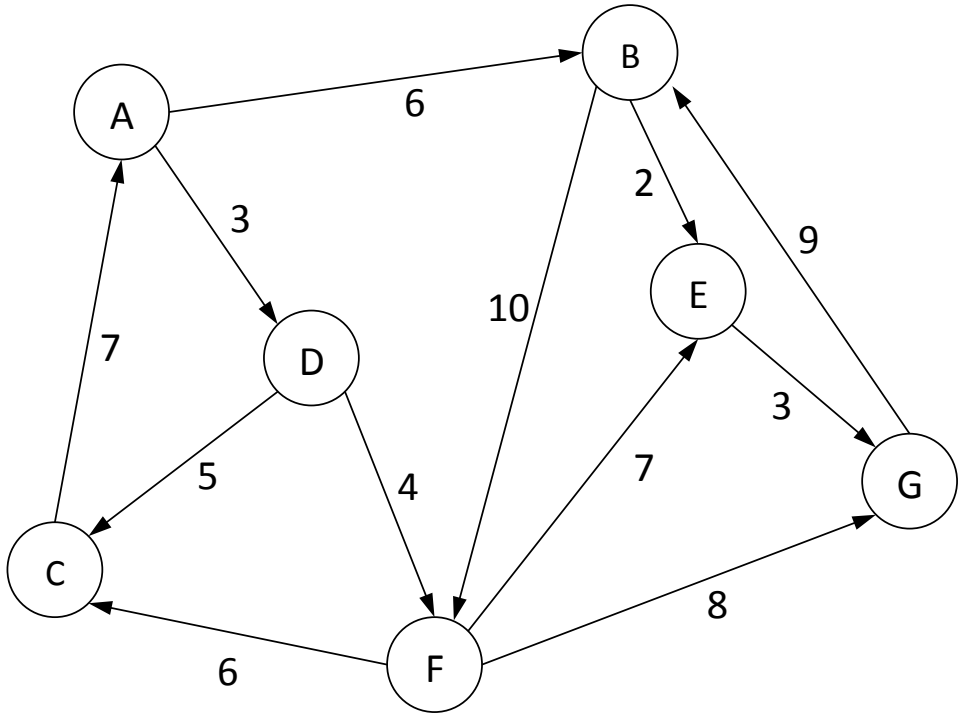
	Ari	Bev	Cal	Don	Eva	Fay	Gia
Ari	1	1	0	1	0	0	0
Bev	1	1	0	0	0	1	0
Cal	0	0	1	1	0	0	0
Don	1	0	1	1	0	1	0
Eva	0	0	0	0	1	0	1
Fay	0	1	0	1	0	1	0
Gia	0	0	0	0	1	0	1

We'd like to find all the people you're connected with (friends, friends of friends, etc.)

Solution: Transitive Closure

- Finding all paths in a graph is called *transitive closure*.
- Use matrix multiplication algorithm, but replace:
  - "Plus" operator $\oplus$ with Boolean OR ($\vee$)
  - "Times" operator $\otimes$ with Boolean AND ($\wedge$)
- This is a *Boolean* $\{\vee, \wedge\}$-*semiring*
- "Multiplying" connectivity matrix with itself gives us friends of your friends
- Repeating $n-1$ times gives everyone in your network
- i.e. Raise matrix to $n-1$ power
  USING OUR POWER ALGORITHM!

Shortest Path in a Directed Graph



What's the shortest path between any pair of nodes in the graph?

Matrix for Shortest Path Problem

- Matrix a_{ij} shows distance from node i to node j
- If no edge between nodes, distance is infinite

	A	B	C	D	E	F	G
A	0	6	∞	3	∞	∞	∞
B	∞	0	∞	∞	2	10	∞
C	7	∞	0	∞	∞	∞	∞
D	∞	∞	5	0	∞	4	∞
E	∞	∞	∞	∞	0	∞	3
F	∞	∞	6	∞	7	0	8
G	∞	9	∞	∞	∞	∞	0

Solution: *Tropical* or {min, +} Semiring

- Use matrix multiplication algorithm, but replace:
  - "Plus" operator $\oplus$ with min
  - "Times" operator $\otimes$ with +
- i.e. the resulting matrix is computed by the formula:

$$b_{ij} = \min_{k=1}^{n}(a_{ik} + a_{kj})$$

- Raise matrix to n–1 power to get shortest path of any length up to n–1 steps
  USING OUR POWER ALGORITHM AGAIN

Noether Answers the Question

- What are the most general mathematical entities that Euclid's GCD algorithm works on (the *domain* or *setting* for the algorithm)?
- Euclidean Domain (aka Euclidean Ring)

Euclidean Domain

E is a *Euclidean domain* if:

- E is an integral domain
- E has operations quotient and remainder such that

$$b \neq 0 \implies a = \text{quotient}(a, b) \cdot b + \text{remainder}(a, b)$$

- E has non-negative norm $\|x\| : E \to \mathbb{N}$ satisfying

$$\|a\| = 0 \iff a = 0$$

$$b \neq 0 \implies \|ab\| \geq \|a\|$$

$$\|\text{remainder}(a, b)\| < \|b\|$$

Norm in Euclidean Domain

- "Norm" in the definition is a measure of magnitude, but it's not the Euclidean norm (length of a vector) from linear algebra.
- Examples:
  - Integers: Norm is the absolute value
  - Polynomials: Norm is the degree of the polynomial
  - Gaussian Integers: The complex norm
- When you compute remainder, norm decreases and eventually goes to zero.

Most Generic GCD

```
template <EuclideanDomain E>
E gcd(E a, E b) {
    while (b != E(0)) {
        a = remainder(a, b);
        std::swap(a, b);
    }
    return a;
}
```

To make something generic, you don't add extra mechanisms. Rather, you remove constraints and strip down the algorithm to its essentials.

Fields

An integral domain where every nonzero element is invertible is called a *field*.

Rational numbers are the canonical example of fields. Other examples are:

- Real numbers
- Prime remainder fields
- Complex numbers

More About Fields

- A prime field is a field that does not have a proper subfield.
- Every field has one of two kinds of prime subfields, rational numbers Q or prime remainder fields Z_p.
- The characteristic of a field is p if its prime subfield is Z_p and 0 if its prime subfield is Q.

Extending Fields

- Algebraically: Add an extra element that is the root of a polynomial.
  - e.g. Extend rational numbers with $\sqrt{2}$ (the root of $x^2 - 2$)
- Topologically: "Fill in the holes."
  - e.g. Rational numbers leave gaps in the number line, but real numbers have no gaps, so the field of real numbers is a topological extension of the field of rational numbers.

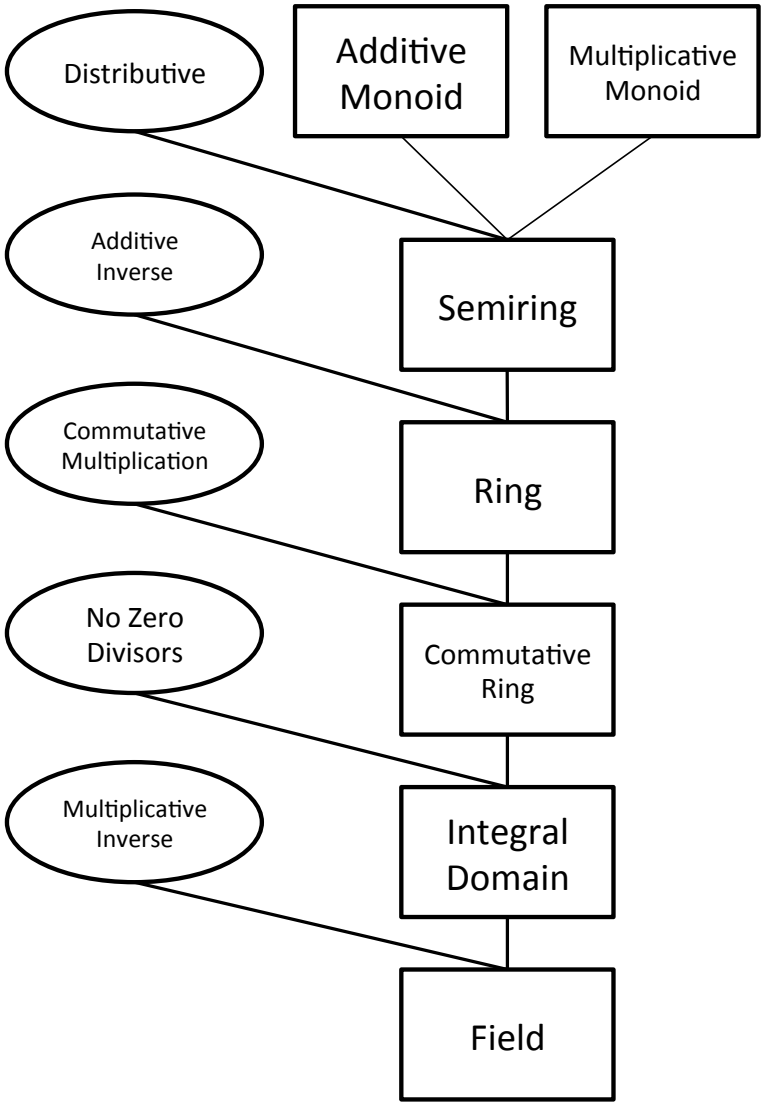
Modules and Vector Spaces

- Structures defined in terms of more than one set.
- *Module*:
  - Primary set: additive group G
  - Secondary set: ring of coefficients R
  - Additional operation: $R \times G \longrightarrow G$ that obeys axioms:

$$a, b \in R \ \wedge \ x, y \in G :$$
$$(a + b)x = ax + bx$$
$$a(x + y) = ax + ay$$

- *Vector Space*: A module whose ring is also a field.

Second Recap of Algebraic Structures



Things to Consider

- An ancient algorithm for multiplying positive integers provided the tool we need to solve modern practical problems (shortest path, social networks) efficiently.
- We've gone through many levels of generalization to get there:
  - We replaced addition with multiplication to get power
  - We replaced the hardcoded operation with an argument
  - We went from integers to matrices
  - We generalized matrix multiplication to new operations

Lecture 15

Building Blocks: Proofs, Theorems, and Axioms

(Covers material from Sec. 9.1-9.4)

Building Blocks of Mathematical Knowledge

- Proofs
- Theorems
- Axioms

What exactly are they?

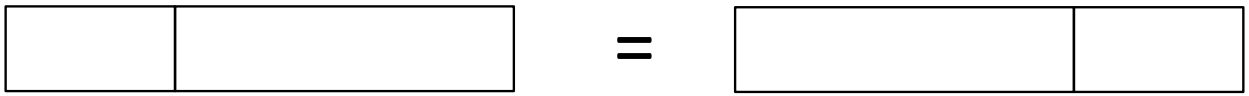
How do we use them?

Early Proofs

- Mathematical proofs date back at least to ancient Greece
- The first proofs were visual
  - Diagram as proof
  - Look at it to understand it
- Relied on our innate spatial reasoning

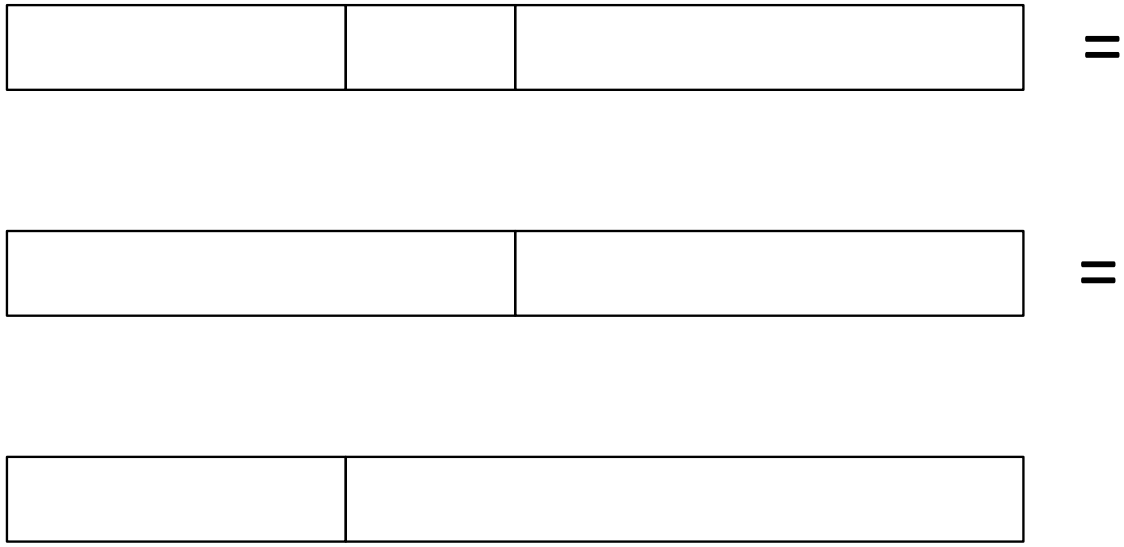
Commutativity of Addition

$$a + b = b + a$$



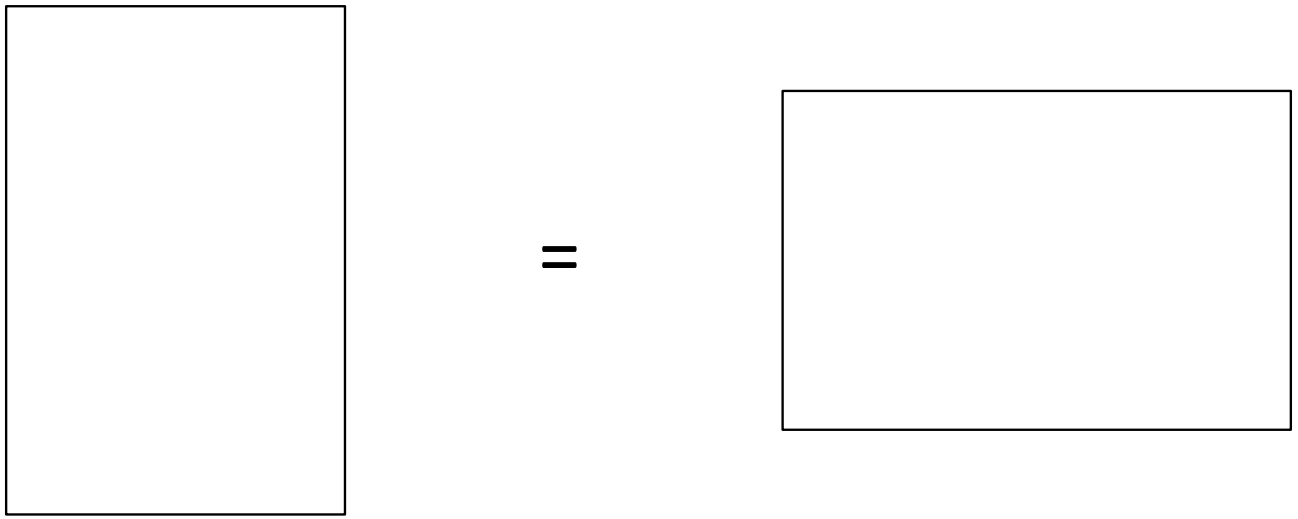
Associativity of Addition

$$(a + b) + c = a + (b + c)$$



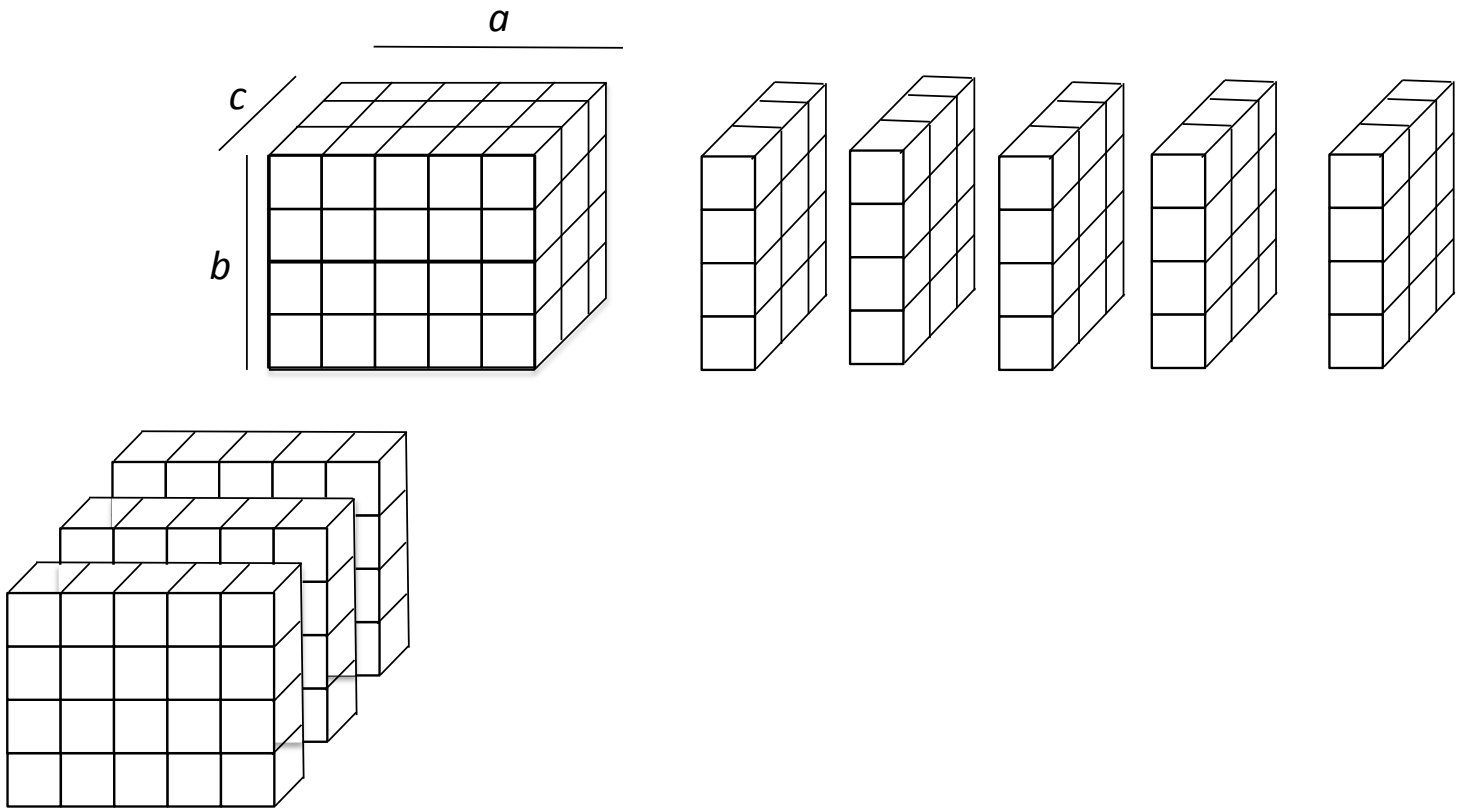
Commutativity of Multiplication

$$ab = ba$$



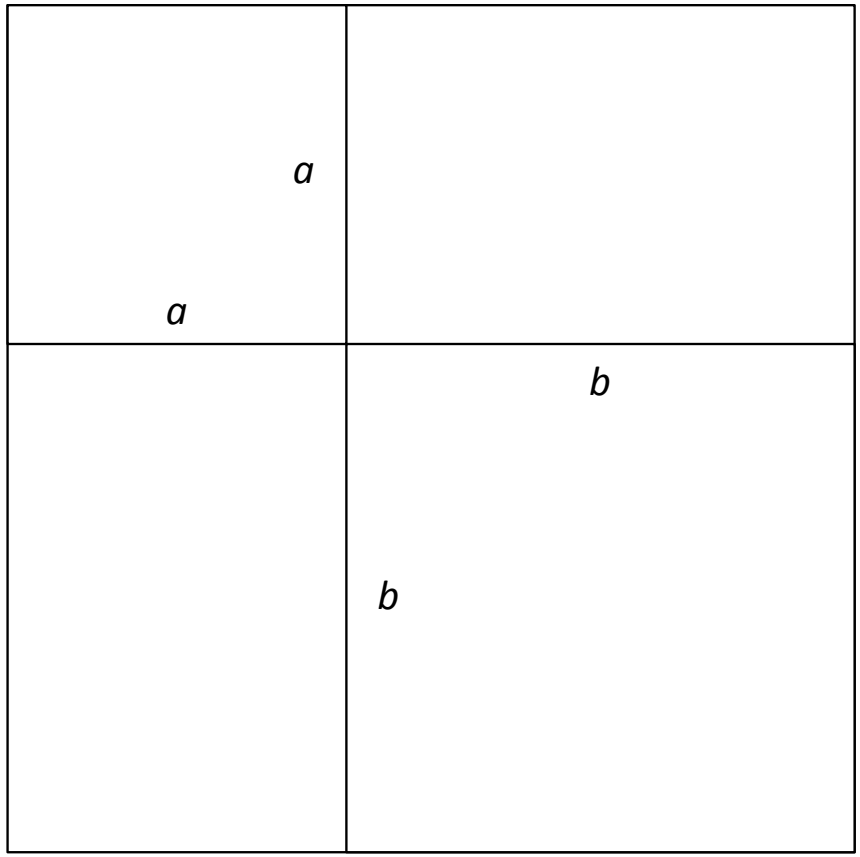
Associativity of Multiplication

$$(ab)c = a(bc)$$



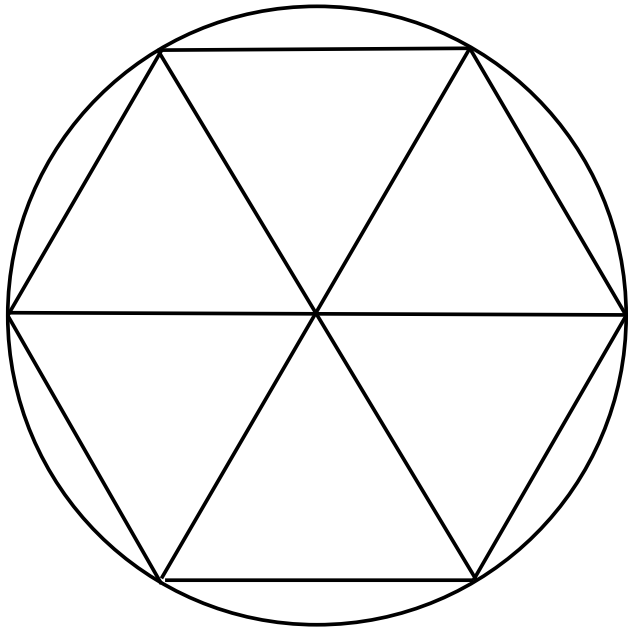
Squaring a Sum

$$(a + b)^2 = a^2 + 2ab + b^2$$



Bounding π

$\pi > 3$



- Unit hexagon (all sides length 1) inscribed in circle
- Triangles are equilateral, so all sides also length 1
- So diameter of circle is 2
- Perimeter of hexagon is 6
- Perimeter of circle obviously greater than of hexagon
- So ratio of perimeter of circle to diameter (2) greater than ratio of perimeter of hexagon (6) to diameter (2).

Limitations of Visual Proofs

- Not sufficient to prove every type of proposition
- No longer considered rigorous enough

Many other proof techniques are available.

What Is a Proof?

A *proof* of a proposition is:

1. An argument
2. Accepted by the mathematical community
3. To establish the proposition as valid.

Observation: #2 above is often overlooked.

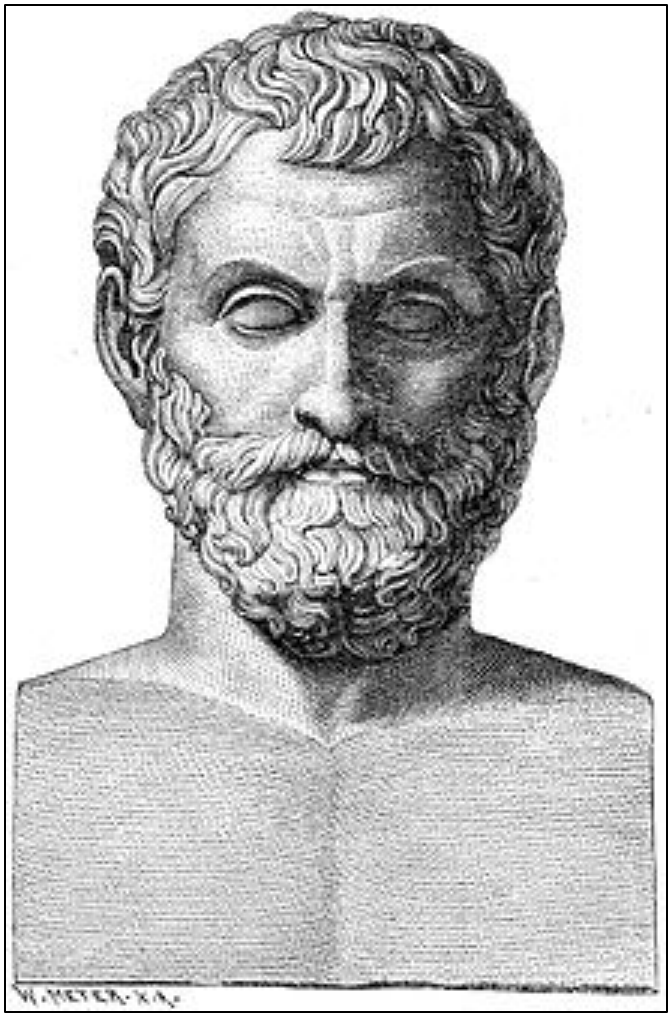
- What's considered a valid proof changes over time
- Proof is fundamentally a social process

Theorems

A *theorem* is a proposition derivable from other propositions.

Idea of a theorem invented by the founder of Western philosophy, Thales of Miletus.

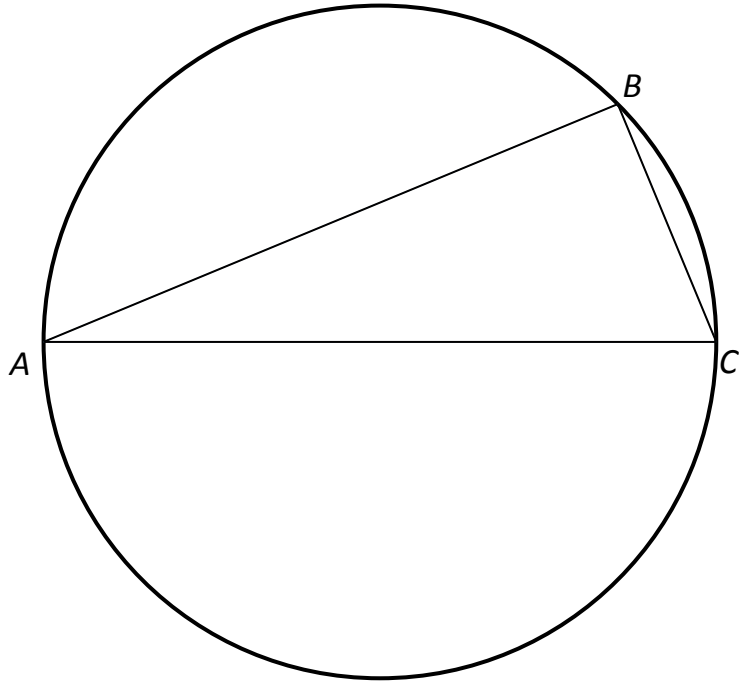
Thales of Miletus (flourished early 6th century BC)



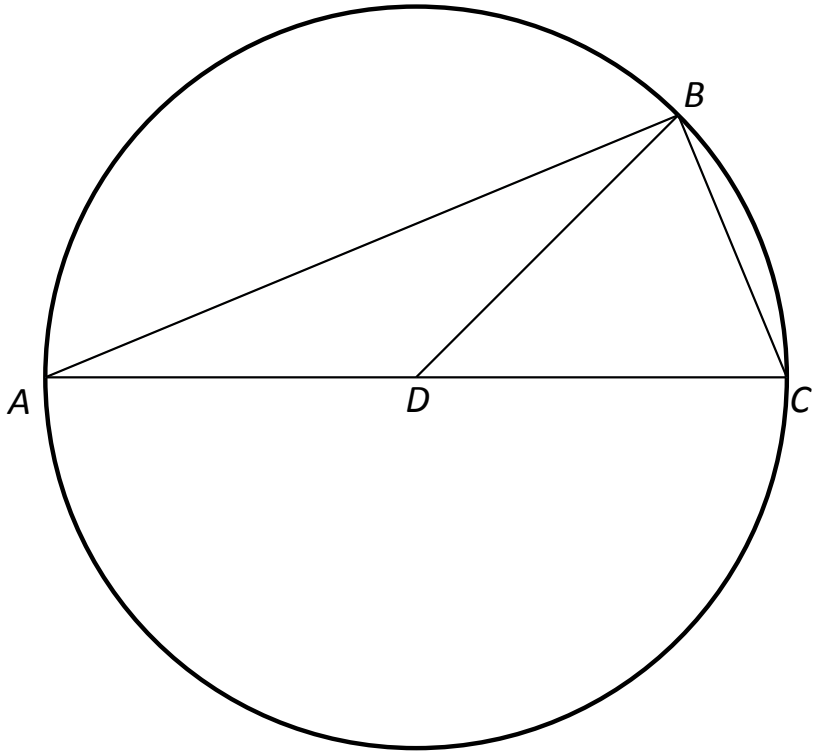
- Founder of what ancient Greeks called philosophy, but we call science
- Looked for natural explanations of reality
- Collected mathematical knowledge
- Predicted solar eclipse, and weather patterns (making money in process)

Thales' Theorem

For any triangle ABC formed by connecting the two ends of a circle's diameter (AC) with any other point B on the circle, $\angle ABC = 90^\circ$



Proof of Thales' Theorem (1)



- Since *DA* and *DB* are both radii, they are equal and triangle *ADB* is isoceles
- The same is true for *DB*, *DC*, and triangle *BDC*. Therefore

$$\angle DAB = \angle DBA$$

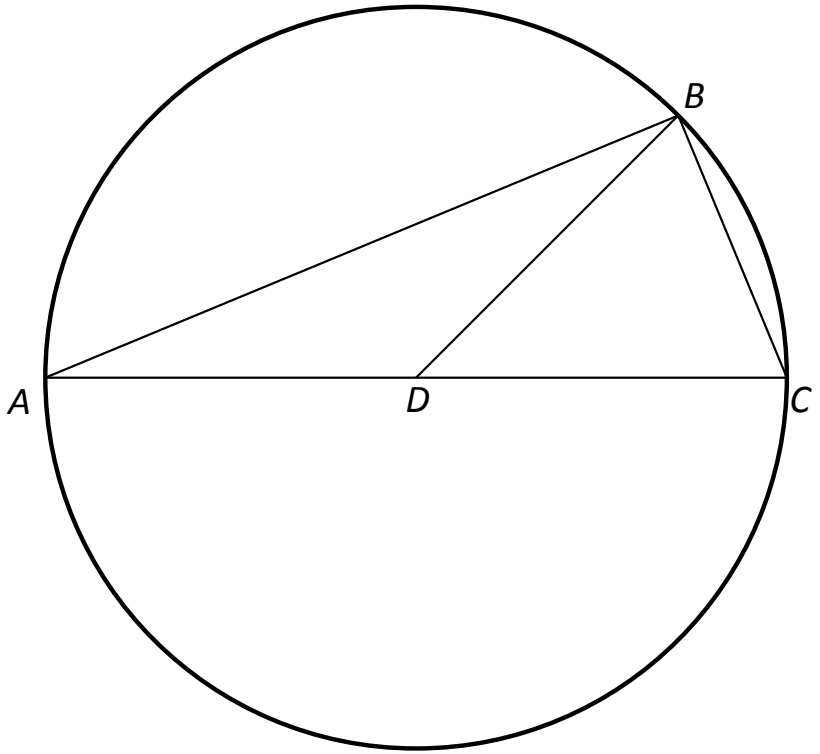
$$\angle DCB = \angle DBC$$

$$\angle DAB + \angle DCB = \angle DBA + \angle DBC$$

Proof of Thales' Theorem (2)

- Angles of triangle sum to 180° and $\angle CBA = \angle DBA + \angle DBC$, so

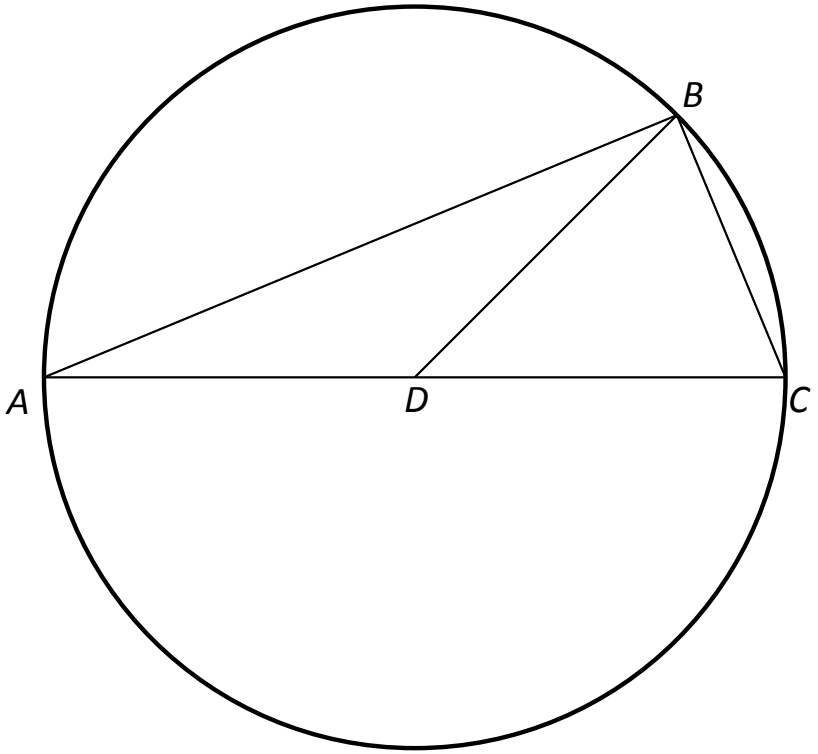
$$\angle DAB + \angle DCB + \angle DBA + \angle DBC = 180^\circ$$



Proof of Thales' Theorem (3)

- Substitute using our previous equality:

$$\angle DAB + \angle DCB + \angle DBA + \angle DBC = 180^\circ$$
$$(\angle DBA + \angle DBC) + (\angle DBA + \angle DBC) = 180^\circ$$
$$2 \cdot (\angle DBA + \angle DBC) = 180^\circ$$
$$\angle DBA + \angle DBC = 90^\circ$$
$$\angle CBA = 90^\circ$$



Importance of Thales' Discovery

- He saw that if you have one piece of knowledge, you can use it to find another.
- Theorems are essential to abstraction, since the value of a theorem is that it applies to *all* entities that have certain properties.

Axioms

- Proofs and theorems are essential tools for building up mathematical knowledge
- But we need a set of starting assumptions as a foundation
- *Axioms* are unprovable assumptions on which the rest of the system is built.

Axiomatic Method

- The *axiomatic method* is the process of constructing an entire mathematical system starting from a few formal principles.
- Euclid's *Elements* was the first example of the axiomatic method. His called his basic principles:
  - Definitions
  - Postulates
  - Common Notions

Euclid's 23 Definitions

1. A point is that which has no parts.
2. A line is a breadthless length.

. . .

23. Parallel straight lines are straight lines which, being in the same plane and being produced indefinitely in both directions, do not meet one another in either directions.

Euclid's Common Notions

1. Things which are equal to the same thing are also equal to one another.
2. If equals be added to equals, the whole are equal.
3. If equals be subtracted from equals, the remainders are equal.
4. Things which coincide with one another are equal to one another.
5. The whole is greater than the part.

Modern Restatement of Common Notions

1. $a = c \ \wedge \ b = c \implies a = b$
2. $a = b \ \wedge \ c = d \implies a + c = b + d$
3. $a = b \ \wedge \ c = d \implies a - c = b - d$
4. $a \cong b \implies a = b$
5. $a < a + b$

Observations:

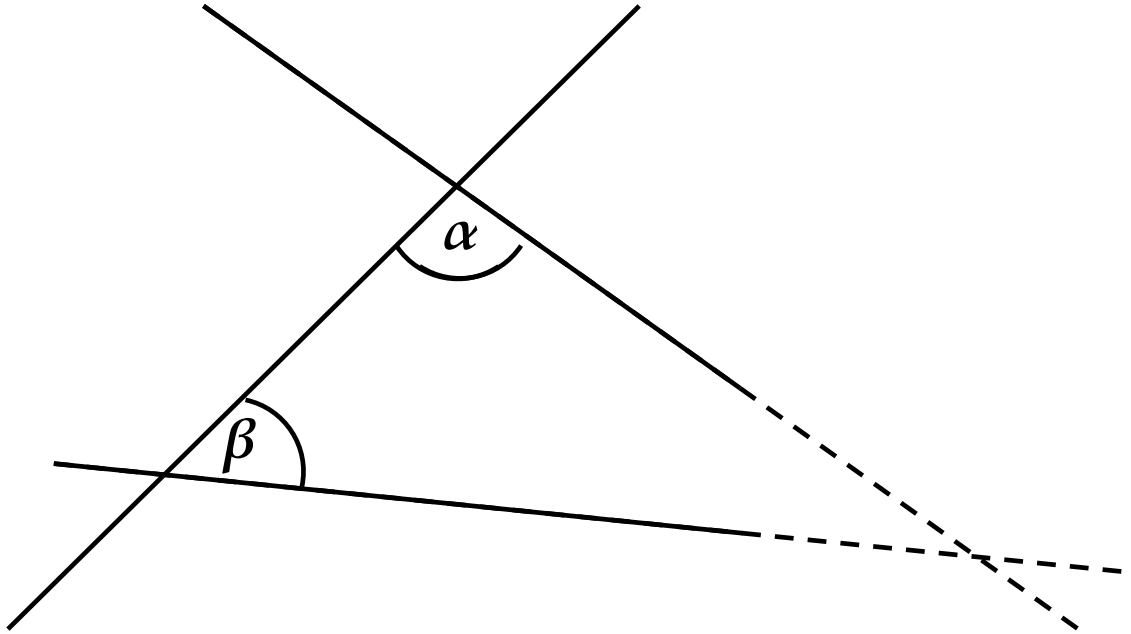
- Not limited to geometry
- Some are essential to programming

Euclid's Postulates

1. To draw a straight line from any point to any point.
2. To produce a finite straight line continuously in a straight line.
3. To describe a circle with any center and distance.
4. That all right angles are equal to one another.
5. That, if a straight line falling on two straight lines make the interior angles on the same side less than two right angles, the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the two right angles.

Illustration of Euclid's Fifth Postulate (aka Parallel Postulate)

if a straight line falling on two straight lines make the interior angles on the same side less than two right angles,
the two straight lines, if produced indefinitely, meet on that side on which are the angles less than the two right angles.



$$\alpha + \beta < 180°$$

Euclid's Fifth Postulate: Mathematically Equivalent Variations

- Playfair's Axiom (1795):
  Given a line and a point not on it, at most one parallel to the given line can be drawn through the point.
- There exists a triangle whose angles add up to 180°.
- There exist two similar triangles that are not congruent.

Euclid's Fifth Postulate
Can it be proved from the others?

Some notable attempts:

- Ptolemy (90-168)
- Omar Khayyam (1050-1153)
- Giovanni Girolamo Saccheri (1667-1733)
  - In *Euclidus Vindicatus* ("Euclid Vindicated"), constructed geometrical system based on assumption that postulate is false, then said consequences were so bizarre that postulate must be true.

Non-Euclidean Geometry

In 1824, Nikolai Lobachevsky realized that the parallel postulate was just one possible assumption.

- Instead of saying "there is at most one line through a point parallel to a given line," he explored the assumption that "there are many lines…"
- Unlike Saccheri, Lobachevsky realized that resulting system was as consistent as Euclid's
- Invented *hyperbolic* geometry

Behavior of Hyperbolic Geometry

- There are no similar triangles except congruent ones
- Space is curved in a “concave” way
  - Sum of angles of triangles don’t sum to 180°
  - The bigger the triangles, the smaller the angles

Reaction to Lobachevsky

- Results initially dismissed by Russian mathematical community
- Eventually recognized by Gauss as valid
- After many years, an accepted part of mathematics
- Today, considered a monumental discovery

Nikolai Ivanovich Lobachevsky (1792-1856)

- From minor province, went to minor university
- Revolutionized geometry
- Elected to Göttingen Academy of Sciences
- Died blind and impoverished
- First great Russian mathematician

Parallel Discovery

- Hungarian mathematician János Bolyai discovered non-Euclidean geometry at almost the same time.
- Bolyai's father sent his son's work to Gauss, whom he knew. Gauss dismissed it as nothing he hadn't already thought of.
- Crushed by the response, Bolyai became mentally unstable and believed Gauss stole his ideas and published them as Lobachevsky.

Which Geometry Describes Our Reality?

- Gauss's proposed experiment:
  1. Find three mountains close together.
  2. Use surveying equipment on each peak to measure the angles.
  3. If they sum to 180°, Euclid is right.
     If less, Lobachevsky is.
- No longer matters
  - Mathematicians have shown independence of 5th postulate
  - Mathematics became purely formal exercise

Things to Consider

- Euclid's *Elements* is still the canonical example of an axiomatic system over 2000 years later.
- Yet even this system that has stood that amazing test of time can still be expanded to reveal new insights – just ask Lobachevsky.
- Therefore, never assume that there is nothing beyond the textbook solution.

Lecture 16

Arithmetic from the Ground Up

(Covers material from Sec. 9.5-9.8)

Rethinking Geometry

- The great mathematician David Hilbert spent 10 years coming up with a new, more rigorous axiomatic system for geometry.
- Hilbert's system has:
  - 7 axioms of connection
  - 4 axioms of order
  - 1 axiom of parallels
  - 6 axioms of congruence
  - 1 Archimedes' axiom
  - 1 completeness axiom

Formalist Approach

One must be able to say 'tables, chairs, beer-mugs' each time in place of 'points, lines, planes.'

-- David Hilbert

- Hilbert advocated a formalist approach to mathematics, where results could be proved without relying on intuitions about the real world.

David Hilbert (1862-1943)

- Professor at Göttingen
- Many contributions to mathematics and physics
- Proposed 23 problems that drove much of 20th century math research
- Work on mechanizing mathematics led to modern theory of computation

Rethinking Arithmetic

- Italian mathematician Giuseppe Peano had a similar idea about building a new, more rigorous axiomatic system for arithmetic
- Published *Formulario Mathematico* ("Mathematical Formulas") in 1899 – a book containing all essential theorems in mathematics
- Invented much of the symbolic notation used today (quantifiers, set operations, etc.)

Peano's Axioms

There is a set $\mathbb{N}$ called the *natural numbers*:

1. $\exists 0 \in \mathbb{N}$
2. $\forall n \in \mathbb{N} \ : \ \exists n' \in \mathbb{N}$ -- called its *successor*
3. $\forall \mathbb{S} \subset \mathbb{N} \ : \ (0 \in \mathbb{S} \ \wedge \ \forall n \ : \ n \in \mathbb{S} \implies n' \in \mathbb{S}) \implies \mathbb{S} = \mathbb{N}$
4. $\forall n, m \in \mathbb{N} \ : \ n' = m' \implies n = m$
5. $\forall n \in \mathbb{N} \ : \ n' \neq 0$

Peano's Axioms in English

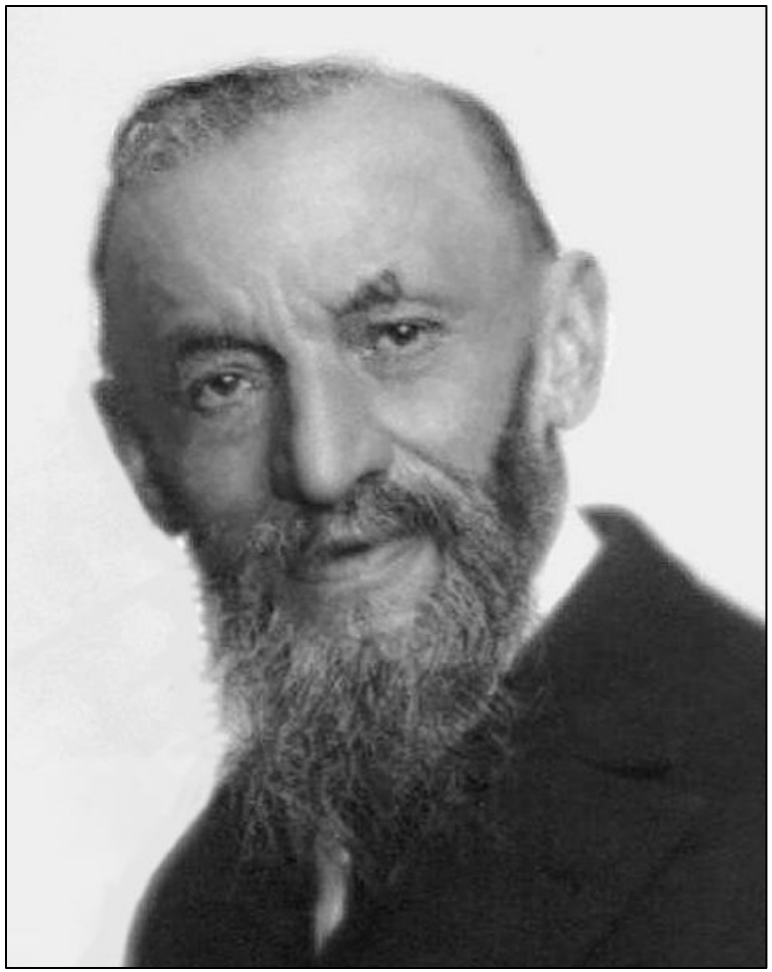
1. Zero is a natural number.
2. Every natural number has a successor.
3. If a subset of natural numbers contains zero, and every element in the subset has a successor in the subset, then the subset contains all natural numbers.
4. If two natural numbers have the same successor, then they are equal.
5. Zero is not the successor of any natural number.

Peano's Third Axiom

"If a subset of natural numbers contains zero, and every element in the subset has a successor in the subset, then the subset contains all natural numbers."

- Known as *axiom of induction*
- Implies that there are no "unreachable" natural numbers
  - If you start with zero and keep taking successor, you'll eventually get to every natural number

Giuseppe Peano (1858-1932)



- Discovered space-filling curve
- Invented a language for writing science and math unambiguously
- Published much of his work in this language
- Most has never been translated

Independence of Peano Axioms

- How do we know all of Peano's axioms are needed?
- We can try removing each axiom and show that the resulting system does not capture what we mean by natural numbers.

Removing Existence of 0 Axiom

$$\exists 0 \in \mathbb{N}$$

- If we remove this axiom, we are forced to drop *all* axioms that refer to zero.
- With no elements to start with, the other axioms never apply and can be satisfied with the empty set.
- The empty set is not a model of natural numbers.

Removing Totality of Successor Axiom

$$\forall n \in \mathbb{N} \ : \ \exists n' \in \mathbb{N} \text{ -- called its } \textit{successor}$$

- If we remove the requirement that every value have a successor, we end up allowing finite sets like {0} or {0, 1, 2}.
- No finite set is a model of natural numbers.

Note: On computers, we give up this axiom, because our data types are finite.

Removing Induction Axiom

$$\forall \mathbb{S} \subset \mathbb{N} \ : \ (0 \in \mathbb{S} \ \wedge \ \forall n \ : \ n \in \mathbb{S} \implies n' \in \mathbb{S}) \implies \mathbb{S} = \mathbb{N}$$

- If we remove this axiom, then we end up with more integer-like things than there are integers.
- These "unreachable" numbers are called transfinite ordinals, designated by ω.
- {0, 1, 2, 3, …, ω_1, ω_1+1, ω_1+2, …, ω_2, ω_2+1, ω_2+2…}
- Clearly not a model of natural numbers.

Removing Invertibility of Successor Axiom

$$\forall n, m \in \mathbb{N} \ : \ n' = m' \implies n = m$$

- If we remove the requirement that equal successors have equal predecessors, we're allowing ρ-shaped structures
- E.g. {0, 1, 1, 1, …} or {0, 1, 2, 3, 2, 3…}
- An item could have multiple predecessors, some earlier in the sequence and some later
- All these structures are finite, so do not model natural numbers.

Remove “Nothing has 0 as successor” Axiom

$$\forall n \in \mathbb{N} \ : \ n' \neq 0$$

- If we remove this axiom, we’d allow structures that loop back to zero
- E.g. {0, 0, 0…} or {0, 1, 0, 1, 0, 1…}
- Again, these are finite, so do not model natural numbers.

Building Arithmetic

- Peano's axioms provide a foundation for arithmetic on natural numbers
- We can create definitions of addition and multiplication starting with these axioms

Definition of Addition

$$a + 0 = a$$

$$a + b' = (a + b)'$$

0 is the left additive identity

$$0 + a = a$$

basis:

$$0 + 0 = 0$$

inductive step:

$$0 + a = a \implies 0 + a' = (0 + a)' = a'$$

Definition of Multiplication

$$a \cdot 0 = 0$$

$$a \cdot b' = (a \cdot b) + a$$

Multiplication by zero on the left

$$0 \cdot a = 0$$

basis:

$$0 \cdot 0 = 0$$

inductive step:

$$0 \cdot a = 0 \quad \Longrightarrow \quad 0 \cdot a' = 0 \cdot a + 0 = 0$$

Definition of 1

$$1 = 0'$$

Adding 1:

$$a + 1 = a + 0' = (a + 0)' = a'$$

Multiplying by 1:

$$a \cdot 1 = a \cdot 0' = a \cdot 0 + a = 0 + a = a$$

Associativity of Addition

$$(a + b) + c = a + (b + c)$$

basis:

$$(a + b) + 0 = a + b = a + (b + 0)$$

inductive step:

$$(a + b) + c = a + (b + c) \Longrightarrow$$

$$\begin{aligned}(a + b) + c' &= ((a + b) + c)' \\ &= (a + (b + c))' \\ &= a + (b + c)' \\ &= a + (b + c')\end{aligned}$$

Commutativity of Addition for 1

$$a + 1 = 1 + a$$

basis:

$$0 + 1 = 1 = 1 + 0$$

inductive step:

$$a + 1 = 1 + a \implies$$

$$\begin{aligned} a' + 1 &= a' + 0' \\ &= (a' + 0)' \\ &= ((a + 1) + 0)' \\ &= (a + 1)' \\ &= (1 + a)' \\ &= 1 + a' \end{aligned}$$

Commutativity of Addition

$$a + b = b + a$$

basis:

$$a + 0 = a = 0 + a$$

inductive step:

$$a + b = b + a \implies$$

$$\begin{aligned} a + b' &= a + (b + 1) \\ &= (a + b) + 1 \\ &= (b + a) + 1 \\ &= b + (a + 1) \\ &= b + (1 + a) \\ &= (b + 1) + a \\ &= b' + a \end{aligned}$$

Limitations of Peano Axioms

- Do Peano Axioms define natural numbers?

  "…the answer is that number (positive integer) cannot be defined (seeing that the ideas of order, succession, aggregate, etc., are as complex as that of number)."

  Giuseppe Peano

- If you don't already known what they are, Peano's axioms won't tell you.
- Instead, the axioms *formalize* our existing ideas.

Things to Consider

- Peano's axioms get us to think about which properties of numbers are essential and which are not.
- This is a good attitude to take when studying the documentation of a programming interface.
  - Why is that requirement imposed?
  - What would be the consequences if it were not there?