Computational Theory

Introduction and Mathematical Preliminaries

Dr Russ Ross

Dixie State University—Computer and Information Technologies

Fall 2017

Adapted from notes by Harry Lewis

Introduction to Formal Systems and Computation

Computer Science 3530

Objectives:

- What are the **fundamental** capabilities and limitations of computers?
- Make a **theory** out of the idea of **computation**

What is “computation”?

“Processing of information based on a finite set of operations or rules.”

- Paper + Pencil Arithmetic

$$\begin{array}{rr} & 121 \\ + & 99 \\ \hline & 220 \end{array}$$

- Abacus
- Slide rule
- Calculator w/moving parts (Babbage wheels, Mark I)
- Ruler & compass geometry constructions
- Digital computers

Further computing devices

- Programs in Python, C, Java
- The Internet and other distributed systems
- Cells/DNA?
- Human brain?
- Quantum computers?

For us computation will be

Processing information by unlimited application of a finite set of operations or rules

What we would like to get past

- “This must be hard because I can’t figure out how to do it”
- “This must be hard because I can’t figure out how to do it and neither can anybody else, including a lot of really smart people”
- “This method seems to get the right answer on every case I’ve tried”
- “It never crashed while I was testing it”

What do we want in a "theory"?

- **Precision**
  - Mathematical, formal
  - Can **prove** theorems about computation, both positive (what can be computed) and negative (what cannot be computed).
- **Generality**
  - Technology-independent, applies to the future as well as the present
  - Abstraction: ignores inessential details

Representing "Information"

- Alphabet

  Ex: $a, b, c, \ldots, z$

- Strings: finite concatenation of alphabet symbols, order matters

  Ex: $qaz, abbab$

  ε = empty string (length 0; sometimes e)

- Inputs (& outputs) of computations are strings

  $\Rightarrow$ we focus on **discrete** computations

Computational Problems (i.e., Tasks)

A single question that has infinitely many different instances

- PARITY: given a string x, does it have an even number of a's?
- MAJORITY: given a string x, does it have more a's than b's?

Problems are defined extentionally: a problem is

- the set of all instances of the question to which the answer is positive
- the set of all $\langle$question, answer$\rangle$ pairs

Computational Problems

Examples of computational problems on numbers

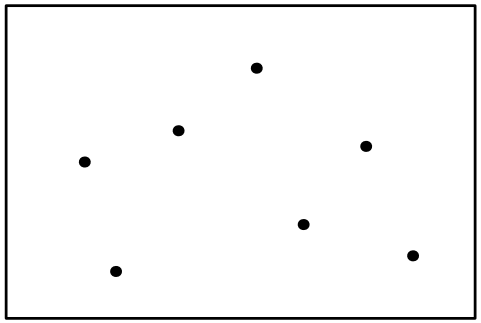
- ADDITION: given two numbers x, y, compute $x + y$
- PRIMALITY: given a number x, is x prime?

Examples of computational problems about computer programs

- C SYNTAX: given a string of ASCII symbols, does it follow the syntax rules for the C programming language?
- HALTING PROBLEM: given a computer program (say in C), can it ever get stuck in an infinite loop?

Computational problems from pure and applied mathematics

- DIOPHANTINE EQUATIONS: Given a polynomial equation (e.g. $x^2 + 3xyz - 44z^3 = 0$), does it have an integer solution?
- TRAVELLING SALESMAN PROBLEM: Given a set of 'cities' in the plane, what is the fastest way to visit them all?



- GRAPH 2-COLORING (3-COLORING): Given a set of people, can they be partitioned into 2 groups so that every pair of people in each group gets along? (3 groups?)

More examples of computational problems

- Register Allocation
- Multiprocessor Scheduling
- Protein Folding
- Decoding Error-Correcting Codes
- Neuron Training
- Auction Winner
- Min-Energy Configuration Of A Gas
- ...

The (Mathematical) Idea of a Language

Underlying Principle: Whatever can be computed can be written down

- Language: any set of strings
- "Solving a yes/no computational problem"
  $\Leftrightarrow$ "Determining if a string is in a given language"

Examples of Languages

- All words in the *American Heritage Dictionary*:

  $\{a, aah, aardvark, \ldots, zyzzva\}$

  Mathematically simple, because it's **finite**!

- All strings with an even number of a's:

  $\{\varepsilon, b, bb, aa, aab, aba, baa, \ldots\}$

  Note: "ε" denotes the string of length 0—the empty string

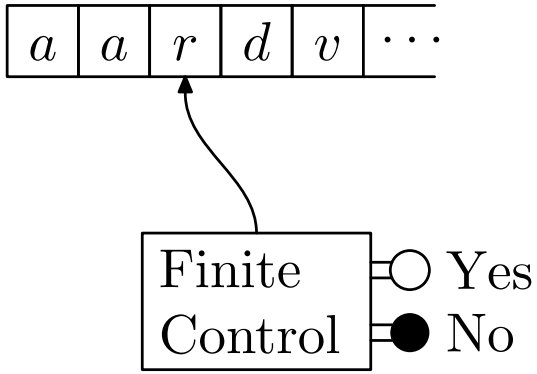
  Infinite—but simple membership **rule**

- All syntactically correct C programs

  (counting space and newline as characters)

Computational Models

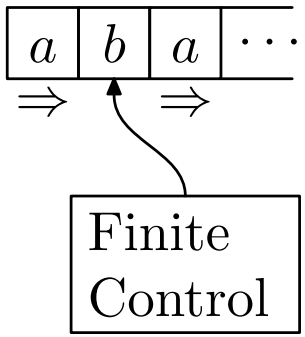
What is a computer? First try: a mathematical automaton.



We don't care how the control is implemented—only that it have a **finite** number of states and change states based on **fixed rules**.

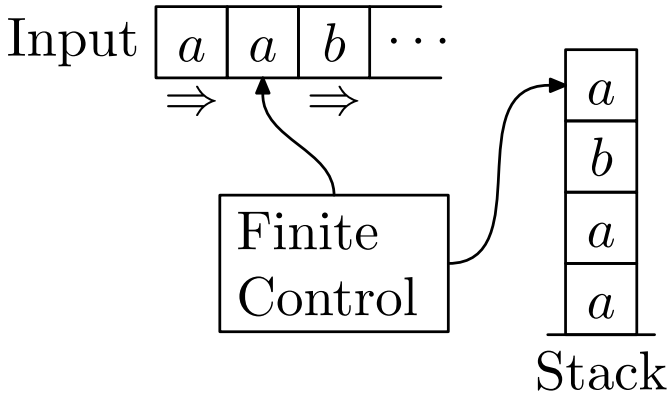
Kinds of Automata

Finite Automata



- Head scans left to right
- Check simple patterns
- Finite table lookup
- Can't count without limit

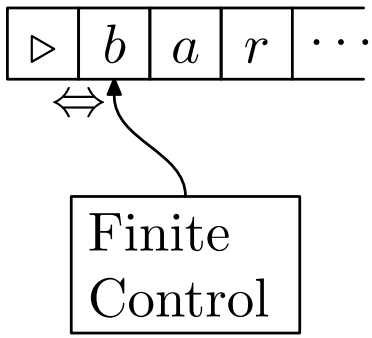
Pushdown Automata



- Use stack to count, balance parentheses
- Check many syntax rules

A model for general-purpose computers

Turing Machines



- **Control** is still finite
- Head moves left and right, reads, and **writes**

Q1: What computational problems can be solved by using these automata?

- Finite Automata: the **regular languages**.

  A FA can determine whether or not strings are "generated" by any fixed **regular expression**, e.g.

a^*	generates	$\{\varepsilon, a, aa, aaa, \ldots\}$
$(ab)^*$	generates	$\{\varepsilon, ab, abab, ababab, \ldots\}$
$(a^*ab)^*a^*$	generates	$\{???\}$
$(a \cup ab)^*$	generates	$\{???\}$

  * = "any number of"

Models

- Pushdown Automata: the **context-free languages**.

  A PDA can determine whether or not strings are generated by any fixed **context-free grammar**, e.g.

  $$\left\{ \begin{array}{l} S \to aSb \\ S \to \varepsilon \end{array} \right\} \text{ generates } \{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

  $$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

  Note: this is **not** the same as $a^* b^*$!

CFGs as models for natural languages

$$\left\{\begin{array}{rcl} \langle\text{sentence}\rangle & \rightarrow & \langle\text{noun-phrase}\rangle\_\langle\text{verb}\rangle \\ \langle\text{noun-phrase}\rangle & \rightarrow & \langle\text{noun}\rangle \mid \langle\text{adjective}\rangle\_\langle\text{noun-phrase}\rangle \\ \langle\text{noun}\rangle & \rightarrow & \text{cat} \mid \text{dog} \mid \text{mouse} \\ \langle\text{adjective}\rangle & \rightarrow & \text{black} \mid \text{hungry} \\ \langle\text{verb}\rangle & \rightarrow & \text{jumps} \mid \text{barks} \end{array}\right\}$$

generates $\{\text{black\_dog\_jumps}, \text{hungry\_black\_cat\_barks}, \ldots\}$

More powerful models

- Turing machines: the **computable languages**
  - Captures our intuitive notion of "computable" (**Church-Turing Thesis**)
  - TMs equivalent in expressiveness to C programs, LISP programs, Pentium CPU, (hypothetical) quantum computers, ...
  - Concept of computability is independent of technology!

Church's Thesis (Church-Turing Thesis)

Intuitive notion of "computable"

$$\equiv$$

Formal notion of "computable by a Turing Machine"

Are there non-computable languages?

Yes—in fact "almost all" languages are not computable

Whare are some examples?

[Problems to avoid!]

Q2: Are there computational problems that **cannot** be solved by these automata?

- Yes—in fact “almost all” problems are not computable
- But what are some examples? [Problems to avoid!]
- A non-regular problem?
- A non-context-free problem?
- Non-computable problems?

Classifying languages

	FA/ regular?	PDA/ context free?	TM/ computable?
PARITY			
MAJORITY			
PRIMALITY			
C SYNTAX			
HALTING			
TSP			
2-COLORING			
DIOPHANTINE EQ.			

Q3: Are there computable problems that cannot be solved **efficiently**?

- A problem need not be **uncomputable** to be **practically unsolvable** (It may just take too long!)
- Theory of relative difficulty of problems

  $\rightarrow$ Based on resources required:

  - Time
  - Memory
  - ...

The NP-Complete Problems

TRAVELING SALESMAN PROBLEM, GRAPH 3-COLORING, MULTIPROCESSOR SCHEDULING, PROTEIN FOLDING, ...

Do they have efficient algorithms? Either all do or none do!

The is the famous (and still open) **P vs. NP Question**.

NP-Complete Problems

- **Integer Linear Program**

  Is there a solution over the positive integers to a system like this?

$$\begin{array}{rcrcrcl} x_1 & - & 4x_2 & + & x_3 & = & 0 \\ x_1 & + & x_2 & + & x_3 & \leq & 0 \\ x_1 & & & + & 7x_3 & \geq & 0 \end{array}$$

- **Boolean Satisfiability**

  Are there true/false values for the variables to make this formula true?

$$(x \vee y \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg z \vee y)$$

[$\vee$ = "or" $\wedge$ = "and" $\neg$ = "not"]

For Computer Scientists

- Technology-independent foundations of CS
- How to reason precisely about computation
- Topics applicable to other parts of CS

Circuit Design	Finite Automata
Parsing + Compiling	Context-free Languages
	Pushdown Automata
Programming	Regular Expressions
Languages	Formalization in General
Program Analysis	Uncomputability
+ Synthesis	
Artificial Intelligence	Formal Systems, Logic
Algorithm Design	Complexity Theory
Databases	Formal Representation + Reasoning
Cryptography	Complexity Theory

For mathematicians

A "computational perspective" on mathematics.

Ex: which is a "better" formula for the nth Fibonacci number (1,1,2,3,5,8,13,21,...)?

1. $F_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n$
2. $F_n =$ the number of strings over alphabet $\{a, b\}$ of length $n-2$ with no two consecutive b's

Connection between computation and mathematical proofs

- Uncomputability $\leftrightarrow$ Gödel's Incompleteness Theorem
- P vs. NP $\leftrightarrow$ "are mathematical proofs as easy to find as they are to verify?"
- Can mathematics be automatized?

Important and famous problem for Mathematics

Rich interplay between the Theory of Computation and various areas of mathematics (logic, combinatorics, algebra, number theory, probability, functional analysis, algebraic geometry, topology, ...). Many research opportunities.

For others

- How to recognize and intepret computational intractability in case it appears in your domain, e.g., PROTEIN FOLDING, NEURON TRAINING, AUCTION WINNER-DETERMINATION, MIN-ENERGY CONFIGURATION OF A GAS
- How to model computation, e.g., as it may occur in Cells/DNA, the brain, economic systems, physical systems, social networks, ...

Philosophically interesting questions

- Are there well-defined problems that cannot be solved automatically?
- Can we always **always** search for a solution to a puzzle more quickly than trying all possibilities?
- Can we formalize the idea that one problem is “harder” than another?

Mathematical Preliminaries

Reading: Sipser, §0.1 and §0.2.

Sets

- **Sets** are defined by their members

  $A = B$ means that for every x, $x \in A$ iff $x \in B$

  Example $\mathcal{N} = \{0, 1, 2, \ldots\}$

- **Cardinality**

  Sets can be **finite** (e.g. $\{1, 3, 5\}$) or **infinite** (e.g. $\mathcal{N}$).

  Q: Is $\{\mathcal{N}\}$ finite?

  If A is finite ($A = \{a_1, \ldots, a_n\}$ for some $n \in \mathcal{N}$), then its **cardinality** (or **size**) $|A|$ is the number of elements in A.

  The **empty set** $\emptyset$ has cardinality 0.

  Cardinality of infinite sets to be discussed later!

Set Operations

$\cup$	union	$\{a, b\} \cup \{b, c\} = \{a, b, c\}$
$\cap$	intersection	$\{a, b\} \cap \{b, c\} = \{b\}$
$-$	difference	$\{a, b\} - \{b, c\} = \{a\}$

- A and B are **disjoint** iff $A \cap B = \emptyset$

Set Operations

$\cup$	union	$\{a, b\} \cup \{b, c\} = \{a, b, c\}$
$\cap$	intersection	$\{a, b\} \cap \{b, c\} = \{b\}$
$-$	difference	$\{a, b\} - \{b, c\} = \{a\}$

- A and B are **disjoint** iff $A \cap B = \emptyset$
- The **power set** of $S = \mathcal{P}(S) = \{X : X \subseteq S\}$

  e.g. $\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

  Q: What is $|\mathcal{P}(S)|$?

Set Operations

$\cup$	union	$\{a, b\} \cup \{b, c\} = \{a, b, c\}$
$\cap$	intersection	$\{a, b\} \cap \{b, c\} = \{b\}$
$-$	difference	$\{a, b\} - \{b, c\} = \{a\}$

- A and B are **disjoint** iff $A \cap B = \emptyset$
- The **power set** of $S = \mathcal{P}(S) = \{X : X \subseteq S\}$

  e.g. $\mathcal{P}(\{a, b\}) = \{\emptyset, \{a\}, \{b\}, \{a, b\}\}$

  Q: What is $|\mathcal{P}(S)|$?
- The **Cartesian product** of sets A, B

  $A \times B = \{(a, b) : a \in A, b \in B\}$

  triples,...

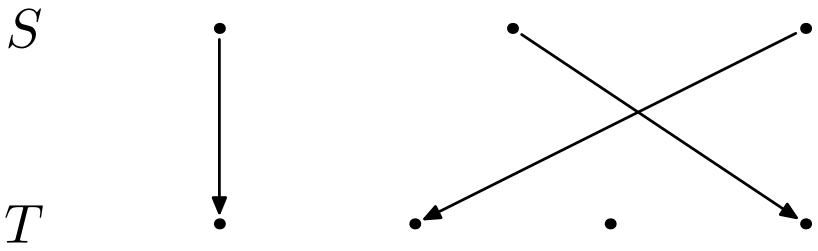
Functions

A **function** $f : S \to T$ maps each element $s \in S$ to (exactly one) element of T, denoted $f(s)$.

For example, $f(n) = n^2$ is a function from $\mathcal{Z} \to \mathcal{N}$

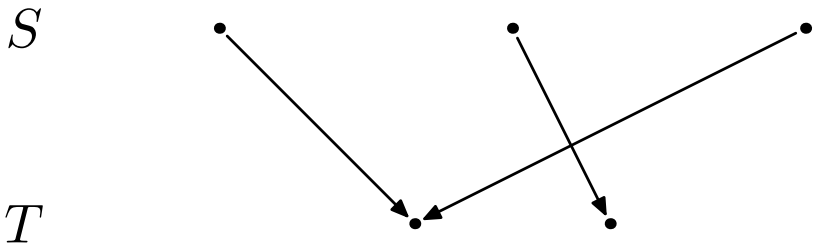
($\mathcal{Z} =$ all integers)

Special varieties of functions



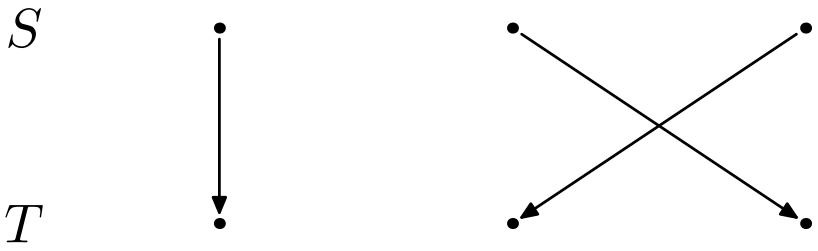
1-1:

$s_1 \neq s_2 \Rightarrow$
$f(s_1) \neq f(s_2)$



Onto:

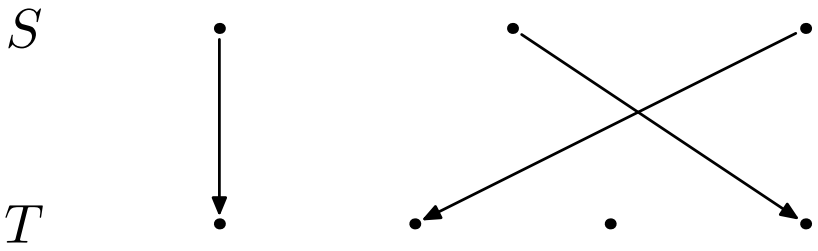
For every $t \in T$
there is an $s \in S$
such that $f(s) = t$



Bijection:

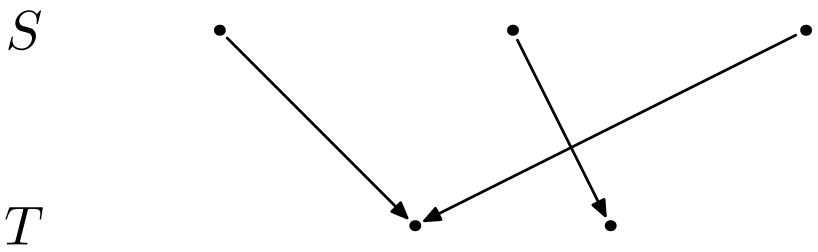
1-1 and onto
"1-1 Correspondence"

Special varieties of functions



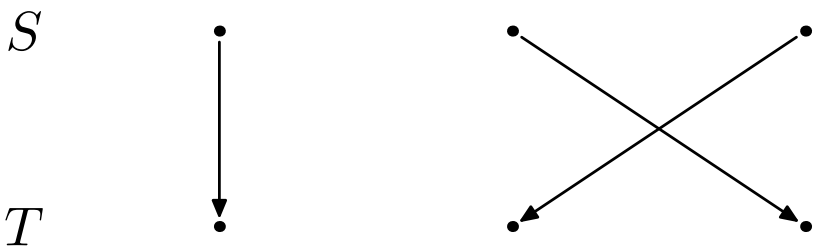
1-1:

$s_1 \neq s_2 \Rightarrow$
$f(s_1) \neq f(s_2)$



Onto:

For every $t \in T$
there is an $s \in S$
such that $f(s) = t$



Bijection:

1-1 and onto
"1-1 Correspondence"

Formal definition of **cardinality**: S has (finite) cardinality $n \in \mathcal{N}$ iff there is a bijection $f : \{1, \ldots, n\} \rightarrow S$.

Relations

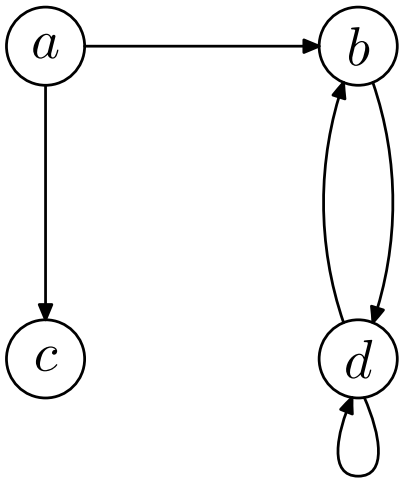
- A k-ary **relation** on $S_1, \ldots, S_k$ is a subset of $S_1 \times \ldots \times S_k$

  [A function $f : S \to T$ corresponds to the relation $\{(s, f(s)) : s \in S\} \subseteq S \times T$.]

- A **binary relation** on S is a subset of $S \times S$
- For example, $(\text{GWB}, \text{GHWB}) \in \text{Son}$, where $\text{Son} = \{(x, y) : x \text{ is a son of } y\}$

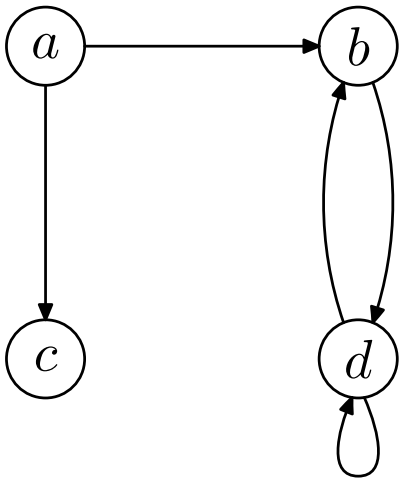
What is a (directed) graph?

- For finite S, a binary relation can be pictured as a "directed graph":



What is a (directed) graph?

- For finite S, a binary relation can be pictured as a "directed graph":

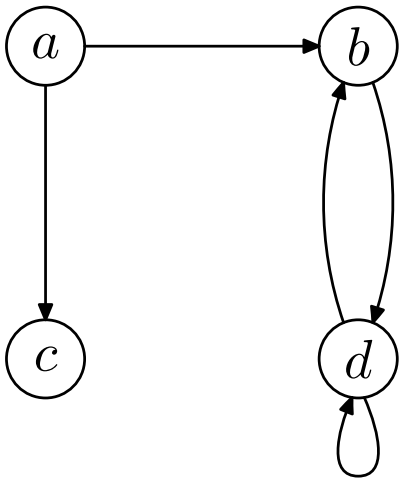


$$= \quad \{(a,b),(a,c),(b,d),(d,b),(d,d)\}$$

- Formally, a **directed graph** G consists of a finite set V of **vertices** (or **nodes**), and a set of edges $E \subseteq V \times V$.

What is a (directed) graph?

- For finite S, a binary relation can be pictured as a "directed graph":



$$= \quad \{(a,b),(a,c),(b,d),(d,b),(d,d)\}$$

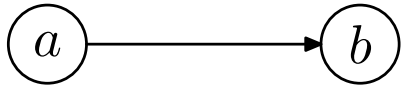
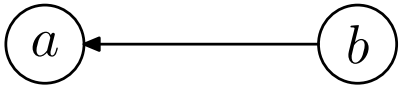
- Formally, a **directed graph** G consists of a finite set V of **vertices** (or **nodes**), and a set of edges $E \subseteq V \times V$.
- **NB**: Because a relation (or edge set) is a **set** (of ordered pairs), there can be only one arrow from one node to another.

Properties of Binary Relations

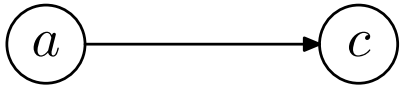
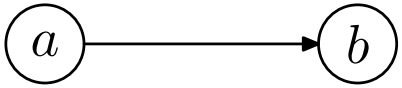
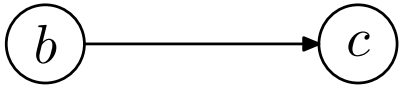
A relation $R \subseteq S \times S$ is:

- **reflexive** if  for each $a \in S$

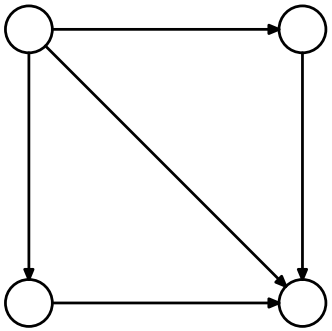
  i.e., $(a, a) \in R$ for each $a \in S$

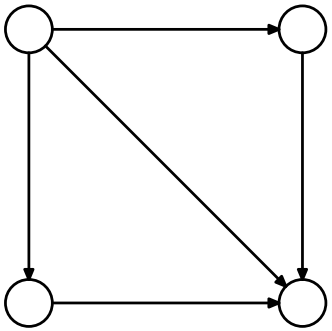
- **symmetric** if  whenever 

  i.e., for any $a, b \in S$, if $(a, b) \in R$ then $(b, a) \in R$

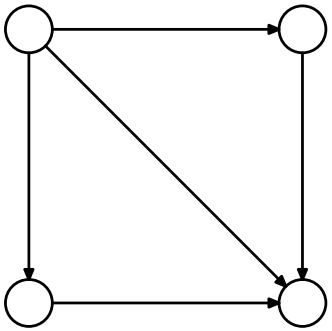
- **transitive** if  whenever  and 

  i.e., if $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$

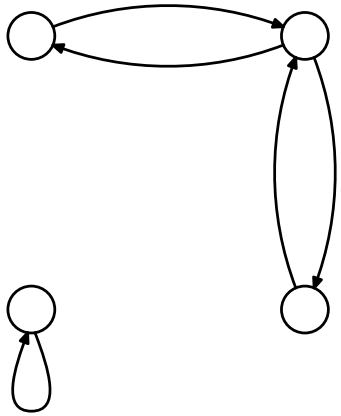


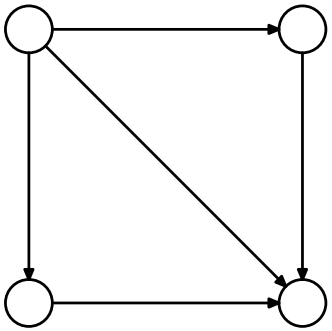


- Transitive
- Not symmetric
- Not reflexive

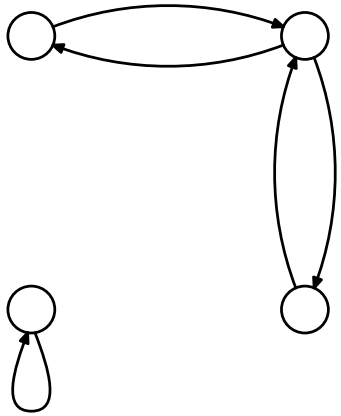


- Transitive
- Not symmetric
- Not reflexive

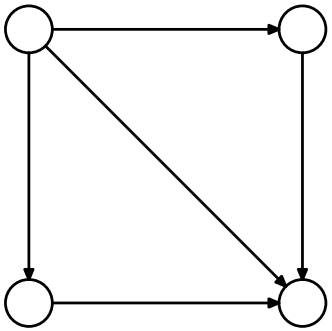




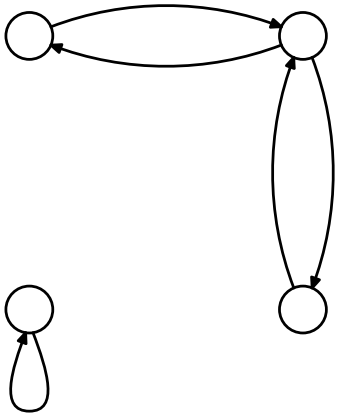
- Transitive
- Not symmetric
- Not reflexive



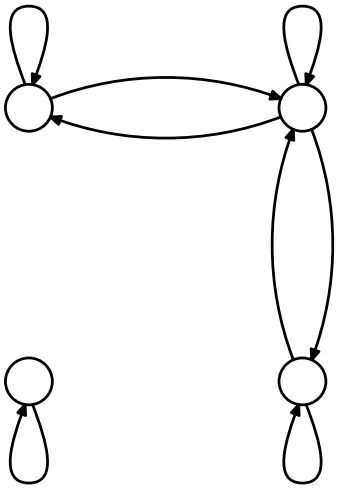
- Symmetric
- Not reflexive
- Not transitive

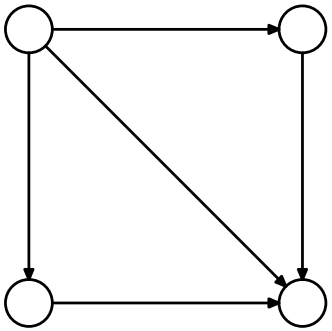


- Transitive
- Not symmetric
- Not reflexive

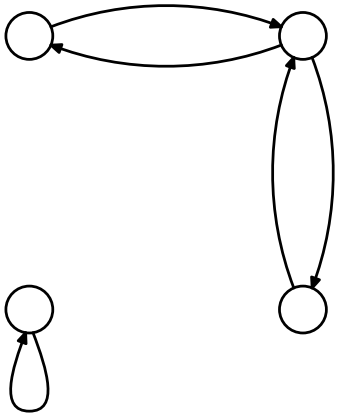


- Symmetric
- Not reflexive
- Not transitive

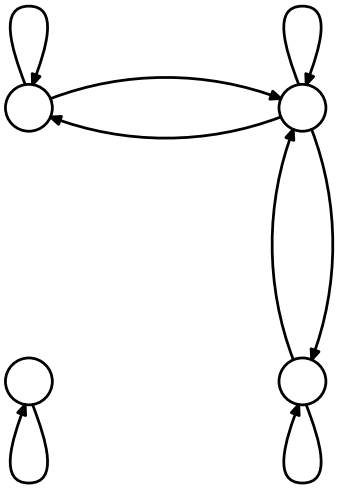




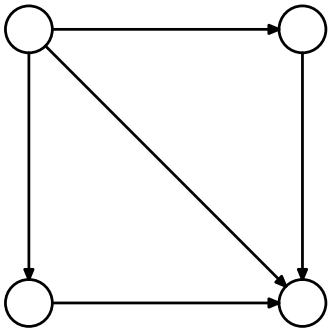
- Transitive
- Not symmetric
- Not reflexive



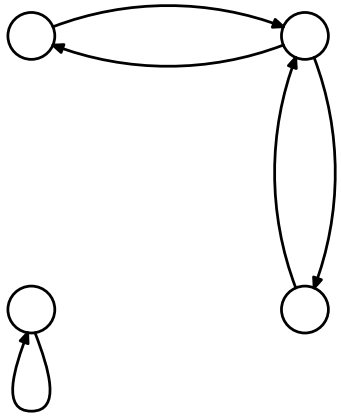
- Symmetric
- Not reflexive
- Not transitive



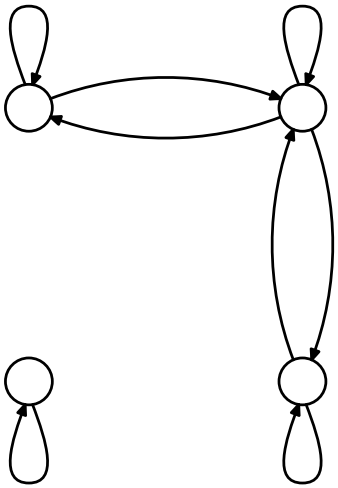
- Reflexive
- Symmetric
- Not transitive



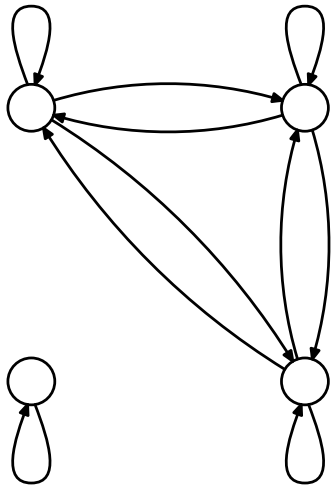
- Transitive
- Not symmetric
- Not reflexive

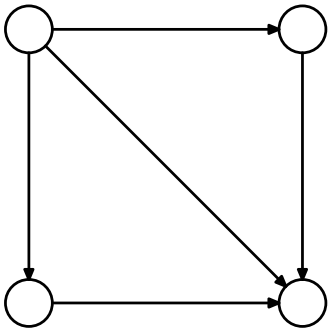


- Symmetric
- Not reflexive
- Not transitive

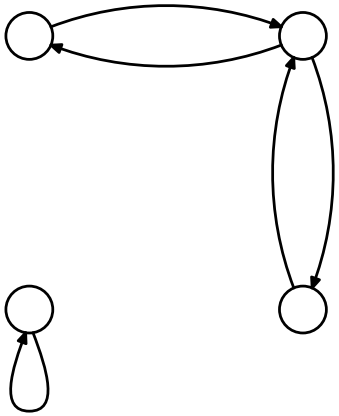


- Reflexive
- Symmetric
- Not transitive

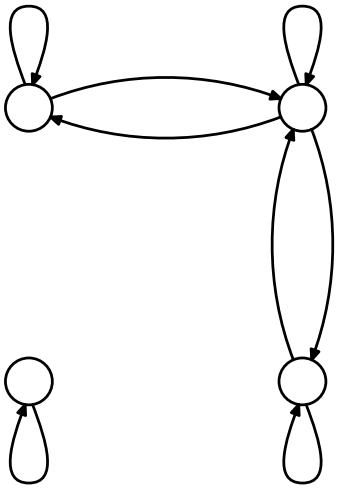




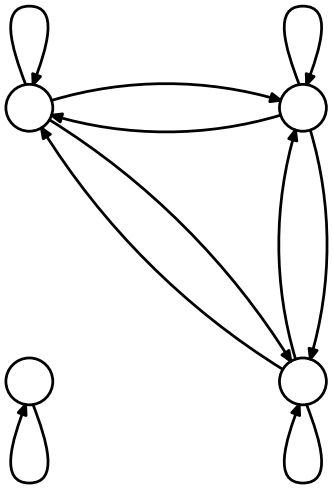
- Transitive
- Not symmetric
- Not reflexive



- Symmetric
- Not reflexive
- Not transitive



- Reflexive
- Symmetric
- Not transitive



- Reflexive
- Transitive
- Symmetric

Equivalence Relations

A relation that satisfies all three properties is called an **equivalence relation**.

An equivalence relation decomposes S into **equivalence classes**—any two members of the same equivalence class bear the relation to each other.

Which Properties Do These Relations Have?

Domain	**Relation**
Cities	Reachable-By-One-Flight-From
Cities	Reachable-From
People	Lives-In-The-Same-City-As
People	Is-An-Ancestor-Of
People	Is-A-Brother-Of

Strings and Languages

- **Symbol** $a, b, \ldots$
- **Alphabet** A finite, nonempty set of symbols
  usually denoted by Σ
- **String** (informal) Finite number of symbols "put together"

  e.g. $abba, b, bb$

  Empty string denoted by ε
- $\Sigma^* =$ set of all strings over alphabet Σ

  e.g. $\{a, b\}^* = \{\varepsilon, a, b, aa, ab, \ldots\}$

More on Strings

- **Length** of a string x is written $|x|$

  $|abba| = 4$

  $|a| = 1$

  $|\varepsilon| = 0$

  The set of strings of length n is denoted Σ^n

Concatenation

- **Concatenation** of strings

  Written as $x \cdot y$, or just xy

  Just follow the symbols of x by the symbols of y

  $x = abba, y = b \Rightarrow xy = abbab$

  $x\varepsilon = \varepsilon x = x$ for any x

- The **reversal** $x^{\mathcal{R}}$ of a string x is x written backwards.

  If $x = x_1 x_2 \cdots x_n$, then $x^{\mathcal{R}} = x_n x_{n-1} \cdots x_1$.

Formal Inductive Definitions

- Like recursive data structures and recursive procedures when programming.
- **Strings** and their **length**:

  ε is a string of length 0.

  If x is a string of length n and $\sigma \in \Sigma$, then $x\sigma$ is a string of length $n + 1$.

  (i.e. start with ε and add one symbol at a time, like $\varepsilon aaba$, but we don't write the initial ε unless the string is empty)

Inductive definitions of string operations

- The **concatenation** of x and y, defined by induction on $|y|$.

$$
\begin{array}{ll}
[|y| = 0] & x \cdot \varepsilon = x \\
[|y| = n + 1] & \text{write } y = z\sigma \text{ for some } |z| = n, \sigma \in \Sigma \\
 & \text{define } x \cdot (z\sigma) = (x \cdot z)\sigma
\end{array}
$$

Inductive definitions of string operations

- The **concatenation** of x and y, defined by induction on $|y|$.

$$\begin{array}{ll} [|y| = 0] & x \cdot \varepsilon = x \\ [|y| = n+1] & \text{write } y = z\sigma \text{ for some } |z| = n, \sigma \in \Sigma \\ & \text{define } x \cdot (z\sigma) = (x \cdot z)\sigma \end{array}$$

- The **reversal** of x, defined by induction on $|x|$:

$$\begin{array}{ll} [|x| = 0] & \varepsilon^{\mathcal{R}} = \varepsilon \\ [|x| = n+1] & (y\sigma)^{\mathcal{R}} = \sigma \cdot y^{\mathcal{R}}, \\ & \text{for any } |y| = n, \sigma \in \Sigma \end{array}$$

A proof by Induction

Proposition: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all $x, y, z \in \Sigma^*$.

(So it doesn't matter what order we concatenate, so we can just write xyz in the future)

- Proof by induction on $|z|$ ...

Proofs by Induction

To prove $P(n)$ for all $n \in \mathcal{N}$:

- "Base Case": Prove $P(0)$.
- "Induction Hypothesis": Assume that $P(k)$ holds for all $k \leq n$ (where n is fixed but arbitrary).
- "Induction Step": Given induction hypothesis, prove that $P(n+1)$ holds.

If we prove the Base Case and the Induction Step, then we have proved that $P(n)$ holds for $n = 0, 1, 2, \ldots$ (i.e., for all $n \in \mathcal{N}$)

Languages

- A **language** L over alphabet Σ is a set of strings over Σ (i.e., $L \subseteq \Sigma^*$)

  Computational problem: given $x \in \Sigma^*$, is $x \in L$?

  Any YES/NO problems can be cast as a language.

- Examples of simple languages
  - All words in the *American Heritage Dictionary* $\{a, aah, aardvark, \ldots, zyzzva\}$.
  - $\emptyset$
  - Σ^*
  - Σ
  - $\{x \in \Sigma^* : |x| = 3\} = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

More complicated languages

- The set of strings $x \in \{a, b\}^*$ such that x has more a's than b's.
- The set of strings $x \in \{0, 1\}^*$ such that x is the binary representation of a prime number.
- All 'C' programs that do not go into an infinite loop.
- $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$ if L_1 and L_2 are languages
- ...

The highly abstract and metaphorical term “language”

- A language can be either finite or infinite
- A language need not have any “internal structure”

Be careful to distinguish

ε	The empty string (a string)
$\emptyset$	The empty set (a set, possibly a language)
$\{\varepsilon\}$	The set containing one element, which is the empty string (a language)
$\{\emptyset\}$	The set containing one element, which is the empty set (a set of sets, maybe a set of languages)

Operations on Languages

- Set operations $\cup$ $\cap$ $-$
- Concatenation of Languages

$$L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$$

e.g. $\{a, b\}\{a, bb\} = \{aa, ba, abb, bbb\}$

e.g. $\{\varepsilon\}L = L$

e.g. $\emptyset L =$?

Kleene star

- Kleene Star

$$L^* = \{w_1 \cdots w_n : n \geq 0, w_1, \ldots, w_n \in L\}$$

e.g. $\{aa\}^* = \{\varepsilon, aa, aaaa, \ldots\}$

e.g. $\{ab, ba, aa, bb\}^*$ = all even length strings

e.g. Σ^* = Kleene Star of Σ

e.g. $\emptyset^* = ?$

Doing Proofs

Reading: Sipser, §0.3 and §0.4.

What is a proof?

A proof is a formal argument of the truth of some mathematical statement.

- "Formal" means that the successive statements are unambiguous, and the steps interlock in a logical vise-grip.
- "Formal" also means that the argument could, in principle, be put in syntax that a machine could check.
- But proofs are meant to be read by human beings, and ordinary conversations and courtesies of human communication should be observed!

Why do we do proofs?

- To be absolutely positive the statement is true
  - For example, it is commonly believed that there are no fast, completely correct algorithms for the Traveling Salesman Problem. But until someone proves “TSP is hard” we won’t know to stop looking
- To understand why it is true, so we can tell whether it can be extended or restricted and still remain true
- (Sometimes) so we can solve a problem associated with the proposition
  - if TSP is not hard, we’d like to find the algorithm

An example (Sipser, Theorem 0.20)

Prove that $\overline{A \cup B} = \overline{A} \cap \overline{B}$

An example (Sipser, Theorem 0.20)

Prove that $\overline{A \cup B} = \overline{A} \cap \overline{B}$

1. Do we know what the statement to be proved means?
   - What kinds of things does it talk about? (What are A and B?)
   - What does the notation mean? (What are $\cup$ and the $\overline{\text{overline}}$?)

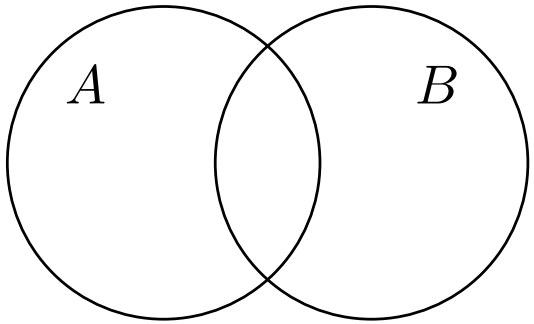
An example (Sipser, Theorem 0.20)

Prove that $\overline{A \cup B} = \overline{A} \cap \overline{B}$

1. Do we know what the statement to be proved means?
   - What kinds of things does it talk about? (What are A and B?)
   - What does the notation mean? (What are $\cup$ and the $\overline{\text{overline}}$?)
2. Try a simple example first
   - Say, $A = \{1, 2\}$ and $B = \{2, 3\}$.
   - Back up to Step 1 if necessary! Ask for help if necessary, but most of the time the information is in the notes and problem sets.

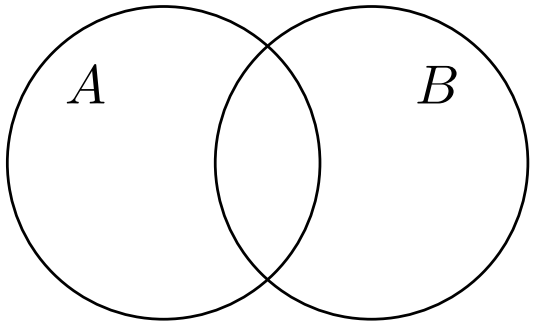
Proof example, continued

3. Try drawing a picture and play with it.



Proof example, continued

3. Try drawing a picture and play with it.



4. Decide on a proof strategy
   - If we are trying to prove two sets equal, a good strategy is *mututal inclusion*: Prove everything in the first is in the second, and then that everything in the second is in the first.
   - This illustrates a more general strategy: Try breaking the problem into simpler chunks and solving them separately.

Now really do the proof

1. Prove that for any sets A and B, $\overline{A \cup B} \subseteq \overline{A} \cap \overline{B}$.
2. Prove that for any sets A and B, $\overline{A} \cap \overline{B} \subseteq \overline{A \cup B}$.

(Done on whiteboard).

Hints for writing up good proofs (thanks largely to Tom Leighton)

1. State the game plan, including the general proof technique you are using.

Hints for writing up good proofs (thanks largely to Tom Leighton)

1. State the game plan, including the general proof technique you are using.
2. Keep the flow linear and use English to explain when you are moving from step to step.

Hints for writing up good proofs (thanks largely to Tom Leighton)

1. State the game plan, including the general proof technique you are using.
2. Keep the flow linear and use English to explain when you are moving from step to step.
3. Proofs are read by human beings, not machines.

Hints for writing up good proofs (thanks largely to Tom Leighton)

1. State the game plan, including the general proof technique you are using.
2. Keep the flow linear and use English to explain when you are moving from step to step.
3. Proofs are read by human beings, not machines.
4. Use as little new symbolism as possible, and use old symbolism correctly.

Hints for writing up good proofs (thanks largely to Tom Leighton)

1. State the game plan, including the general proof technique you are using.
2. Keep the flow linear and use English to explain when you are moving from step to step.
3. Proofs are read by human beings, not machines.
4. Use as little new symbolism as possible, and use old symbolism correctly.
5. Avoid “clearly,” which bullies the reader and often hides errors.

Hints for writing up good proofs (thanks largely to Tom Leighton)

1. State the game plan, including the general proof technique you are using.
2. Keep the flow linear and use English to explain when you are moving from step to step.
3. Proofs are read by human beings, not machines.
4. Use as little new symbolism as possible, and use old symbolism correctly.
5. Avoid “clearly,” which bullies the reader and often hides errors.
6. When you are done, explain why you are done.

Finding a pattern

- One way to come up with an idea for a proof is to find a pattern in small examples
- How many strings of length n over $\{a, b\}$ have no consecutive occurrences of a?

Finding a pattern

- One way to come up with an idea for a proof is to find a pattern in small examples
- How many strings of length n over $\{a, b\}$ have no consecutive occurrences of a?
  1. a, b: 2
  2. ab, ba, bb: 3
  3. aba, abb, bab, bba, bbb: 5
  4. $abab, abba, abbb, baba, babb, bbab, bbba, bbbb$: 8
- Are these the Fibonacci numbers?

The Fibonacci Numbers

$$
\begin{aligned}
F_0 &= 0, \\
F_1 &= 1, \\
F_n &= F_{n-1} + F_{n-2} \text{ for all } n > 1
\end{aligned}
$$

n	0	1	2	3	4	5	6	7	...
F_n	0	1	1	2	3	5	8	13	...

- Maybe the number of strings of length n with no two consecutive a is F_{n+2}?

The Fibonacci Numbers

$$
\begin{aligned}
F_0 &= 0, \\
F_1 &= 1, \\
F_n &= F_{n-1} + F_{n-2} \text{ for all } n > 1
\end{aligned}
$$

n	0	1	2	3	4	5	6	7	...
F_n	0	1	1	2	3	5	8	13	...

- Maybe the number of strings of length n with no two consecutive a is F_{n+2}?
- Try breaking strings of length n with no consecutive a into two subsets:
  1. Those beginning with a
  2. Those beginning with b
- How many of each?

Proof by induction

- Think of an infinite ladder. We need to show something about every rung. Here is the strategy:
  1. Prove it for the bottom rung. This is called the **Base Case**.
  2. Prove that no matter where you are on the ladder, if it is true for the rung you are on and all the lower rungs, it must be true for the next rung. This is called the **Induction Step**.
- If you can prove the base case and the induction step, you have in one fell swoop proved it for every rung of the ladder.
- The **Induction Hypothesis** is the assumption, for some fixed but arbitrary value of n, that the statement is true for the nth rung of the ladder and all the lower rungs.
- So the Induction Step is to prove that the Induction Hypothesis proves that the statement is true at the next rung of the ladder.

Proof

- Let $P[n]$ be the statement that the number of strings of length n without consecutive a is F_{n+2}.
- Base case: $n = 1$ and $n = 2$, i.e. check $P[1]$ and $P[2]$.
- Induction hypothesis is that $P[r]$ holds for every $r \leq n$. Here n is fixed but arbitrary.
- Induction step. Consider strings of length $n + 1$ with no consecutive a.
  1. How many begin with a? F_{n+1} by the induction hypothesis (the next symbol must be b).
  2. How many begin with b? F_{n+2} by the induction hypothesis (no restriction on remaining symbols except no consecutive a).
- Total $= F_{n+1} + F_{n+2} = F_{n+3}$ proving $P[n + 1]$

Discovering, backing up, and fixing proofs

- Mathematicians always hide the process by which they discover things
- In this example we would not have realized that we needed two base cases until we were well into the proof
- And the crucial insight, to break the strings of length n into subsets by their first letter, is an unexplained "rabbit out of the hat"

A bogus inductive "proof"

Let $P(n)$ be the statement that for any x, $x^n = 1$.
This is plainly false but we will "prove" it.

1. Base case: $n = 0$. Then $P(0)$ is the statement that for any x, $x^0 = 1$, which is plainly true.
2. Induction hypothesis is that $P(k)$ holds for all $k \leq n$.
3. Induction step. To prove $P(n+1)$ on the basis of the base case and the induction hypothesis, note that

$$x^{n+1} = \frac{x^n \cdot x^n}{x^{n-1}}$$

But by the induction hypothesis, $x^n = 1$ and $x^{n-1} = 1$ (that is, $P(n)$ and $P(n-1)$ are both true). So

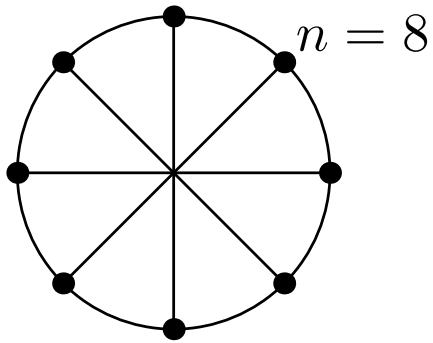
$$x^{n+1} = \frac{1 \cdot 1}{1} = 1$$

What went wrong???

Other proof techniques

- Proof by contradiction
  1. Assume the exact opposite of what you are trying to prove
  2. Deduce a logical contradiction

  (Sipser Thm 0.24: $\sqrt{2}$ is irrational)
- Proof by construction
  - Proves the existence of something and also delivers it into your hands
  - For every even number n, there is an undirected graph in which every node is the endpoint of exactly 3 edges (i.e., is of **degree** 3)
  - Proof idea:



Nonconstructive proofs

- Prove the existence of something without indicating how to put your hands on it
- Two people in this class have a birthday in the same month
- Proof by counting argument

The Pigeonhole Principle

- If $n+1$ pigeons are in n pigeonholes then some pigeonhole has more than one pigeon in it
  - The "pigeons" are the class members
  - Pigeonhole(p) = the month of the year on which p was born
  - If there are more than 12 people in the class, this mapping cannot be 1-1
  - So there are p_1, p_2 such that Pigeonhole(p_1) = Pigeonhole(p_2)
- This **existence** proof gives no help in identifying people with the same birth month

Nonconstructive Proof Example #2: Numbers with a certain property

- Proof that there exist irrational a, b such that a^b is rational
  - Is $\sqrt{2}^{\sqrt{2}}$ rational? Don't know. But consider both possibilities:
    1. $\sqrt{2}^{\sqrt{2}}$ is rational. In that case we are done ($a = b = \sqrt{2}$)
    2. $\sqrt{2}^{\sqrt{2}}$ is irrational. But then

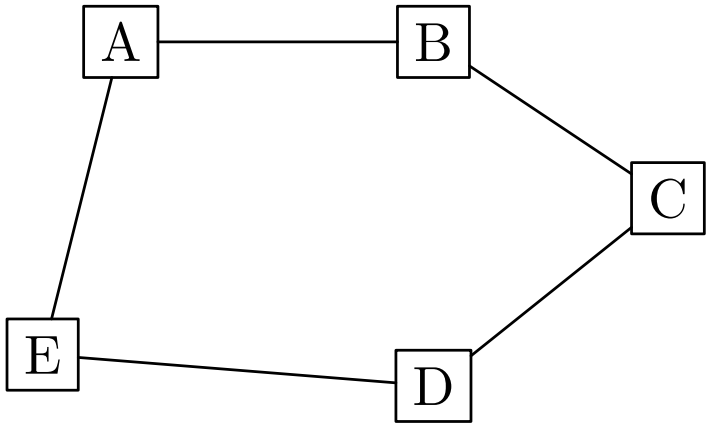
$$\left(\sqrt{2}^{\sqrt{2}}\right)^{\sqrt{2}} = \left(\sqrt{2}\right)^{\sqrt{2}\cdot\sqrt{2}} = \left(\sqrt{2}\right)^2 = 2$$

       which is rational and we are done ($a = \sqrt{2}^{\sqrt{2}}, b = \sqrt{2}$)
  - Proof does not tell us whether case 1 or case 2 holds, but one of the other must be true
  - Law of the Excluded Middle or *tertium non datur*

A combined constructive and nonconstructive proof

(A) In any group of six people there are either three that know each other or three that don't, but (B) in some groups of five people there are no three who mutually know each other and also no three who mutually don't know each other.

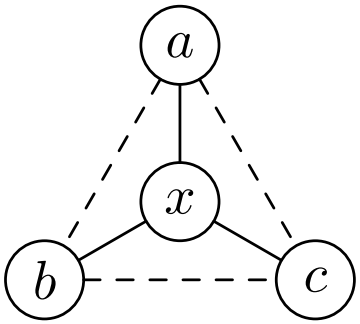
- Constructive proof of (B):



Proof, continued

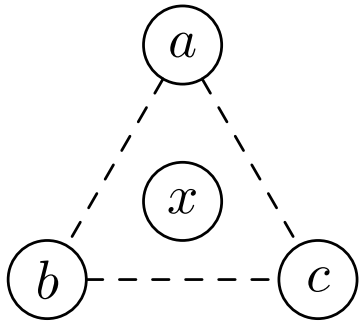
- Nonconstructive proof of (A) by contradiction
  1. Suppose not. Then in some particular group of 6 people, there are no 3 who mutually know each other and no 3 who mutually don't.
  2. Pick some individual X. Either X knows 3 of the other 5, or there are some 3 of the other 5 whom X does not know. (Pigeonhole)
     - 2.1 If X knows 3, say A, B, C, then no two of them can know each other. For example, if A knew B, then X, A, B would all know each other. But then no two of A, B, C know each other, contradiction.
     - 2.2 If there are 3 whom X does not know, say A, B, C, then each two of those must know each other. For example, if A and B did not know each other, then no two of X, A, B would know each other. But then A, B, C all know each other, contradiction.
- NB: The opposite of a "for all" statement is a "for some . . . not" or "there exists . . . not" statement.
- In theory at least, (A) could also be solved by **exhaustive search**.
- It would be enough to say that case (2.2) is "symmetrical"

Visualizing the proof of (A)



A solid line means the two know each other. If any of the dashed lines is a "knows" relationship, then there would be a group of 3 that know each other. If none of those lines is a "knows" relationship, then A, B, and C are a group of 3 that do not know each other.

If any of these dashed edges is not a "knows" relationship, then the two nodes together with X form a group of 3 that do not know each other. If all of the dashed edges are "knows" relationships, A, B, and C are a group of 3 that all know each other.



A nonconstructive proof that tells us almost nothing

- There exists an unbiased 7-sided die
- ???