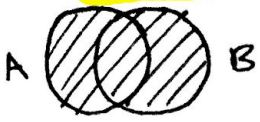
SET THEORY

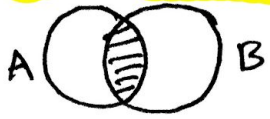
quick rundown:

UNION



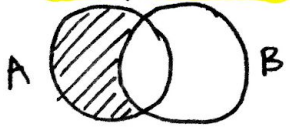
$A \cup B$ = things in A or in B

INTERSECTION



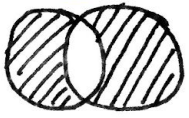
$A \cap B$ = things in both A and B

DIFFERENCE



$A - B$ = things in A but not in B

SYMMETRIC DIFFERENCE



$A \Delta B$ = things in A or B but not both

SUBSET

$A \subseteq B$

every element of A is an element of B

POWER SET

$\mathcal{P}(A)$

set of all subsets of A

PROOFS ABOUT SETS

to show $A \subseteq B$:

- pick arbitrary $x \in A$.
- show $x \in B$.

to show $A = B$:

- prove $A \subseteq B$.
- prove $B \subseteq A$.

Example: set theory proof

Let A and B be arbitrary sets. Prove that $A \in \wp(B)$ if and only if $A \cap B = A$.

What is A?

What is B?

to show $A = B$:

- prove $A \subseteq B$.
- prove $B \subseteq A$.

Example: set theory proof

Let A and B be arbitrary sets. Prove that $A \in \wp(B)$ if and only if $A \cap B = A$.

What is A?
What is B?

to show $A = B$:
- prove $A \subseteq B$.
- prove $B \subseteq A$.

Proof 1: We will prove both directions of implication. First, we'll prove that if $A \in \wp(B)$, then $A \cap B = A$. To do so, we'll prove both $A \cap B \subseteq A$ and $A \subseteq A \cap B$.

Let's begin by showing that $A \cap B \subseteq A$. To do so, pick any $x \in A \cap B$. This means in that $x \in A$, and since our choice of x was arbitrary, we conclude that $A \cap B \subseteq A$, as needed.

Next, we'll show that $A \subseteq A \cap B$. Consider any $x \in A$. We will prove that $x \in A \cap B$. We know $A \in \wp(B)$, which means that $A \subseteq B$. Since $x \in A$ and $A \subseteq B$, we see that $x \in B$. Then, since $x \in A$ and $x \in B$, we see that $x \in A \cap B$, as required.

For the other direction of implication, assume that $A \cap B = A$. We will prove that $A \in \wp(B)$. To do so, we will prove that $A \subseteq B$. So pick any $x \in A$. Then since $x \in A$ and $A = A \cap B$, we see that $x \in A \cap B$. Therefore, we see that $x \in B$. Since our choice of $x \in A$ was arbitrary, we see that $A \subseteq B$, as required. ■

Example: set theory proof

Let A and B be arbitrary sets. Prove that $A \in \wp(B)$ if and only if $A \cap B = A$.

to show $A \subseteq B$:

- pick arbitrary $x \in A$.
- show $x \in B$.

What is A?
What is B?

Proof 1: We will prove both directions of implication. First, we'll prove that if $A \in \wp(B)$, then $A \cap B = A$. To do so, we'll prove both $A \cap B \subseteq A$ and $A \subseteq A \cap B$.

Let's begin by showing that $A \cap B \subseteq A$. To do so, pick any $x \in A \cap B$. This means in that $x \in A$, and since our choice of x was arbitrary, we conclude that $A \cap B \subseteq A$, as needed.

Next, we'll show that $A \subseteq A \cap B$.

Example: set theory proof

Let A and B be arbitrary sets. Prove that $A \in \wp(B)$ if and only if $A \cap B = A$.

to show $A \subseteq B$:
- pick arbitrary $x \in A$.
- show $x \in B$.

What is A?
What is B?

Proof 1: We will prove both directions of implication. First, we'll prove that if $A \in \wp(B)$, then $A \cap B = A$. To do so, we'll prove both $A \cap B \subseteq A$ and $A \subseteq A \cap B$.

Let's begin by showing that $A \cap B \subseteq A$. To do so, pick any $x \in A \cap B$. This means in that $x \in A$, and since our choice of x was arbitrary, we conclude that $A \cap B \subseteq A$, as needed.

Next, we'll show that $A \subseteq A \cap B$. Consider any $x \in A$. We will prove that $x \in A \cap B$. We know $A \in \wp(B)$, which means that $A \subseteq B$. Since $x \in A$ and $A \subseteq B$, we see that $x \in B$. Then, since $x \in A$ and $x \in B$, we see that $x \in A \cap B$, as required.

For the other direction of implication, assume that $A \cap B = A$. We will prove that $A \in \wp(B)$. To do so, we will prove that $A \subseteq B$. So pick any $x \in A$. Then since $x \in A$ and $A = A \cap B$, we see that $x \in A \cap B$. Therefore, we see that $x \in B$. Since our choice of $x \in A$ was arbitrary, we see that $A \subseteq B$, as required. ■

TYPES OF PROOFS!

UNIVERSAL STATEMENT – "for all x..."
- proof: pick arbitrary x, show statement is true
- disproof: find counterexample

EXISTENTIAL STATEMENT – "there is an x such that..."
- proof: find example
- disproof: pick arbitrary x, show statement is false

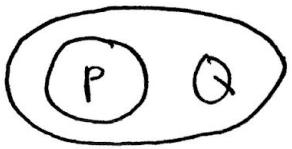
To prove an implication $P \to Q$:

DIRECTLY $(P \to Q)$
- assume P, prove Q

BY CONTRAPOSITIVE $(!Q \to !P)$
- assume !Q, prove !P

BY CONTRADICTION
- assume !P, arrive at a contradiction

$P \to Q$

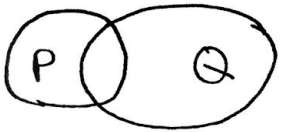


if P is true, then Q is true

$!(P \to Q)$

$!(P \to !Q)$

sometimes P is true and Q is true,



sometimes P is true and Q is false

$P \to !Q$

if P is true, then Q is false

FIRST ORDER LOGIC ← can help us unpack / take the negation of statements we're trying to prove

"All Ps are Qs"	$\forall x.\ P(x) \rightarrow Q(x)$
"no Ps are Qs"	$\forall x.\ P(x) \rightarrow \neg Q(x)$
"some Ps are Qs"	$\exists x.\ P(x) \wedge Q(x)$
"some Ps are not Qs"	$\exists x.\ P(x) \wedge \neg Q(x)$

$\forall$ is usually paired with $\rightarrow$

$\exists$ is usually paired with $\wedge$

$\forall x.\ \exists y.\ P(x,y)$

"for any choice of x, there is some y where $P(x,y)$ is true"

$\exists x.\ \forall y.\ P(x,y)$

"there is some x where for any choice of y, $P(x,y)$ is true"

Existential statements are false unless there's a positive example

Universal statements are true unless there's a counterexample

- Statements with $\exists$ and $\rightarrow$ can be true when talking about something you don't care about
  - ↳ ex. $\exists x.\ (Happy(x) \rightarrow \ldots)$ choose someone unhappy, making antecedent false
- Statements with $\forall$ and $\wedge$ can be false when talking about something you don't care about
  - ↳ ex. $\forall p.\ (person(p) \wedge \ldots)$ choose something that's not a person

BINARY RELATIONS

REFLEXIVE

$\forall a \in A \; aRa$

every element is related to itself

Proof setup:

Pick an $a \in A$. Show aRa.

Visually:

every element has a self-loop

IRREFLEXIVE

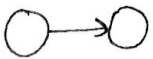
$\forall a \in A. \; \not aRa$

no element is related to itself

Proof setup:

Pick an $a \in A$. Prove $a \not R a$.

Visually:



no element has self-loops

SYMMETRIC

$\forall a \in A. \forall b \in A. (aRb \rightarrow bRa)$

if a is related to b, b is related to a.

Proof setup:

pick $a, b \in A$ such that aRb. prove bRa.

Visually:

for every forwards arrow, you have a backwards arrow

ASYMMETRIC

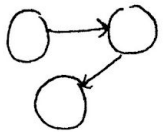
$\forall a \in A. \forall b \in A \; (aRb \rightarrow b \not R a)$

if a is related to b, b is NOT related to a

Proof setup:

pick $a, b \in A$ such that aRb. Prove $b \not R a$.

Visually:



if there is a forwards arrow, there is no backwards arrow

TRANSITIVE

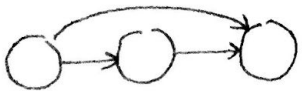
$\forall a \in A. \forall b \in A. \forall c \in A. (aRb \wedge bRc \rightarrow aRc)$

if a is related to b and b is related to c, then a is related to c

proof setup:

pick $a, b, c \in A$ such that aRb and bRc. prove aRc.

Visually:



EQUIVALENCE RELATIONS

- reflexive, symmetric, and transitive

STRICT ORDERS

- irreflexive, asymmetric, and transitive
  or equivalently
- irreflexive + transitive
- asymmetric + transitive

Example: binary relations proof

If R_1 is a binary relation over a set A_1 and R_2 is a binary relation over a set A_2, then an ***embedding of R_1 in R_2*** is a function $f : A_1 \to A_2$ such that

$$\forall a \in A_1.\ \forall b \in A_1.\ (aR_1b \leftrightarrow f(a)\ R_2\ f(b)).$$

If there's an embedding of a relation R_1 in a relation R_2, we say that R_1 ***can be embedded in*** R_2.

Let R_1 be a binary relation over a set A_1 and let R_2 be a *strict order* over some set A_2. Prove that if R_1 can be embedded in R_2, then R_1 is a strict order.

IRREFLEXIVE

$\forall a \in A.\ a \not R a$.

no element is related to itself

Proof setup:

Pick an $a \in A$. Prove $a \not R a$.

What set is the relation defined over?

(Where should we be picking our arbitrary element from?)

Example: binary relations proof

If R_1 is a binary relation over a set A_1 and R_2 is a binary relation over a set A_2, then an ***embedding of R_1 in R_2*** is a function $f : A_1 \to A_2$ such that

$$\forall a \in A_1.\ \forall b \in A_1.\ (aR_1b \leftrightarrow f(a)\ R_2\ f(b)).$$

If there's an embedding of a relation R_1 in a relation R_2, we say that R_1 ***can be embedded in*** R_2.

Let R_1 be a binary relation over a set A_1 and let R_2 be a *strict order* over some set A_2. Prove that if R_1 can be embedded in R_2, then R_1 is a strict order.

IRREFLEXIVE

$\forall a \in A.\ a \not R a$.

no element is related to itself

Proof setup:

Pick an $a \in A$. prove $a \not R a$.

What set is the relation defined over?

(Where should we be picking our arbitrary element from?)

Proof 1: Let $f : A_1 \to A_2$ be an embedding of R_1 in R_2. We will show that R_1 is a strict order by proving that it is irreflexive and transitive.

First, we'll show that R_1 is irreflexive. Consider any $a \in A_1$. Since R_2 is a strict order, we know that R_2 is irreflexive, so $f(a) \not R_2 f(a)$. Then, since f is an embedding of R_1 in R_2, we see that aR_1a, as required.

Example: binary relations proof

If R_1 is a binary relation over a set A_1 and R_2 is a binary relation over a set A_2, then an ***embedding of R_1 in R_2*** is a function $f : A_1 \to A_2$ such that

$$\forall a \in A_1.\ \forall b \in A_1.\ (aR_1b \leftrightarrow f(a)\ R_2\ f(b)).$$

If there's an embedding of a relation R_1 in a relation R_2, we say that R_1 ***can be embedded in*** R_2.

Let R_1 be a binary relation over a set A_1 and let R_2 be a *strict order* over some set A_2. Prove that if R_1 can be embedded in R_2, then R_1 is a strict order.

TRANSITIVE

$\forall a \in A.\ \forall b \in A.\ \forall c \in A.\ (aRb \wedge bRc \to aRc)$

if a is related to b and b is related to c, then a is related to c

proof setup:

pick $a, b, c \in A$ such that aRb and bRc. prove aRc.

What set is the relation defined over?
(Where should we be picking our arbitrary elements from?
What assumptions do we get to make on them?)

Proof 1: Let $f : A_1 \to A_2$ be an embedding of R_1 in R_2. We will show that R_1 is a strict order by proving that it is irreflexive and transitive.

First, we'll show that R_1 is irreflexive. Consider any $a \in A_1$. Since R_2 is a strict order, we know that R_2 is irreflexive, so $f(a) R_2 f(a)$. Then, since f is an embedding of R_1 in R_2, we see that aR_1a, as required.

Example: binary relations proof

If R_1 is a binary relation over a set A_1 and R_2 is a binary relation over a set A_2, then an ***embedding of R_1 in R_2*** is a function $f : A_1 \to A_2$ such that

$$\forall a \in A_1.\ \forall b \in A_1.\ (aR_1b \leftrightarrow f(a)\ R_2\ f(b)).$$

If there's an embedding of a relation R_1 in a relation R_2, we say that R_1 ***can be embedded in*** R_2.

Let R_1 be a binary relation over a set A_1 and let R_2 be a *strict order* over some set A_2. Prove that if R_1 can be embedded in R_2, then R_1 is a strict order.

TRANSITIVE

$\forall a \in A.\ \forall b \in A.\ \forall c \in A.\ (aRb \wedge bRc \to aRc)$

if a is related to b and b is related to c, then a is related to c

proof setup:

pick $a, b, c \in A$ such that aRb and bRc. prove aRc.

What set is the relation defined over?
(Where should we be picking our arbitrary elements from?
What assumptions do we get to make on them?)

Proof 1: Let $f : A_1 \to A_2$ be an embedding of R_1 in R_2. We will show that R_1 is a strict order by proving that it is irreflexive and transitive.

First, we'll show that R_1 is irreflexive. Consider any $a \in A_1$. Since R_2 is a strict order, we know that R_2 is irreflexive, so $f(a) R_2 f(a)$. Then, since f is an embedding of R_1 in R_2, we see that aR_1a, as required.

Next, we'll show that R_1 is transitive. To do so, consider any $a, b, c \in A_1$ where aR_1b and bR_1c. Since f is an embedding of R_1 in R_2, we then see that $f(a)R_2\ f(b)$ and $f(b)R_2\ f(c)$. Then, since R_2 is a strict order, it's transitive, and so $f(a)R_2\ f(c)$. Finally, since f is an embedding of R_1 in R_2, we use the reverse direction of the implication to conclude that aR_1c, as required. ■

FUNCTIONS

All functions: $f: A \to B$

- $\forall a \in A.\ \exists b \in B.\ (f(a) = b)$
  every input maps to some output
- $\forall a_1 \in A.\ \forall a_2 \in A.\ (a_1 = a_2 \to f(a_1) = f(a_2))$
  functions are deterministic - equal inputs produce equal outputs

Injective functions $f: A \to B$

- $\forall a_1 \in A.\ \forall a_2 \in A.\ (a_1 \neq a_2 \to f(a_1) \neq f(a_2))$
  different inputs produce different outputs
  or equivalently
- $\forall a_1 \in A.\ \forall a_2 \in A.\ (f(a_1) = f(a_2) \to a_1 = a_2)$
  some outputs come from same inputs

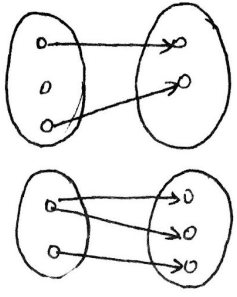
Surjective functions $f: A \to B$

- $\forall b \in B.\ \exists a \in A.\ f(a) = b$
  for every possible output, there's at least one possible input that produces it

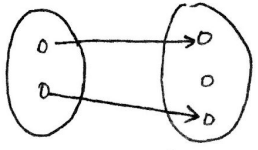
Bijective functions

- functions that are both injective and surjective

NOT functions!

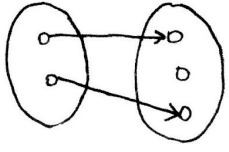


Note:

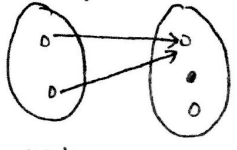


this is okay. not all elements of the codomain must be produced as outputs.

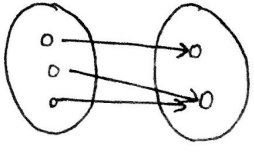
injection



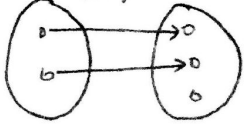
not an injection



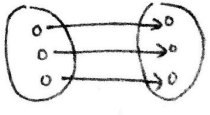
surjection



not a surjection



bijection



Example: function proof

Imagine you have a function $f : A \to B$ from some set A to some set B. We can use f to construct a new function called the ***lift of f***, denoted $\mathbf{lift}_f$, from $\wp(A)$ to $\wp(B)$. Specifically $\text{lift}_f : \wp(A) \to \wp(B)$ is defined as follows:

$$\text{lift}_f(S) = \{\ y \mid \exists x \in S.\ f(x) = y\ \}$$

Let A and B be sets. Prove that if $f : A \to B$ is injective, then lift_f is injective.

Injective functions $f: A \to B$

- $\forall a_1 \in A.\ \forall a_2 \in A.\ (a_1 \neq a_2 \to f(a_1) \neq f(a_2))$

  different inputs produce different outputs

What function are we trying to prove things about?

What is the domain of that function?

Example: function proof

Imagine you have a function $f : A \to B$ from some set A to some set B. We can use f to construct a new function called the ***lift of f***, denoted $\mathbf{lift}_f$, from $\wp(A)$ to $\wp(B)$. Specifically $\text{lift}_f : \wp(A) \to \wp(B)$ is defined as follows:

$$\text{lift}_f(S) = \{\ y \mid \exists x \in S.\ f(x) = y\ \}$$

Let A and B be sets. Prove that if $f : A \to B$ is injective, then lift_f is injective.

Injective functions $f: A \to B$

- $\forall a_1 \in A.\ \forall a_2 \in A.\ (a_1 \neq a_2 \to f(a_1) \neq f(a_2))$

different inputs produce different outputs

What function are we trying to prove things about?

What is the domain of that function?

Proof 1: Let $f : A \to B$ be an injective function. We will prove that lift_f is injective as well. To do so, consider any $S_1, S_2 \in \wp(A)$ where $S_1 \neq S_2$. We will prove that $\text{lift}_f(S_1) \neq \text{lift}_f(S_2)$.

Since $S_1 \neq S_2$, we know that either $S_1 \not\subseteq S_2$ or that $S_2 \not\subseteq S_1$. Without loss of generality, assume $S_1 \not\subseteq S_2$, which means that there is some $a \in S_1$ where $a \notin S_2$.

First, notice that since $a \in S_1$, we see that $f(a) \in \text{lift}_f(S_1)$. We now claim that $f(a) \notin \text{lift}_f(S_2)$. To see this, suppose for the sake of contradiction that $f(a) \in S_2$. This means that there must be some $a' \in S_2$ such that $f(a') = f(a)$. Since f is injective, that tells us that $a' = a$, and since $a' \in S_2$, we see that $a \in S_2$ as well. But this is impossible, since we know that $a \notin S_2$. We've reached a contradiction, so our assumption was wrong and $f(a) \notin \text{lift}_f(S_2)$.

Since $f(a) \in \text{lift}_f(S_1)$ but $f(a) \notin \text{lift}_f(S_2)$, we see $\text{lift}_f(S_1) \neq \text{lift}_f(S_2)$, which is what we needed to show. ■

CANTOR'S THM PROOF SKETCH

$|A| = |B|$ if there exists a bijection $f: A \to B$
↳ to prove this, just find a bijection that works

$|A| \neq |B|$ if <u>every</u> function $f: A \to B$
↳ to prove this, we have to show that <u>no possible function</u> will work

claim: if S is a set, then $|S| \neq \mathcal{P}(S)$

pf. sketch:
- pick an arbitrary set S
- pick an <u>arbitrary function</u> $f: S \to \mathcal{P}(S)$
- show f is not <u>surjective</u> using a diagonal argument

	x_0	x_1	x_2	$x_3 \ldots$
$x_0 \to$	$\{$	x_1,		$x_3 \ldots \}$
$x_1 \to$	$\{x_0$,		x_2,	$x_3 \ldots \}$
$x_2 \to$	$\{x_0$,		x_2,	$\ldots \}$
$\ldots$				

f maps these elements of S ↑ to these elements of $\mathcal{P}(S)$

	x_0	x_1	x_2	$x_3 \ldots$
$x_0 \to$	N	Y	N	Y ...
$x_1 \to$	Y	N	Y	Y ...
$x_2 \to$	Y	N	Y	N ...
$x_3 \to$	N	N	N	Y ...

no matter what f we choose, we can always take the diagonal:

N N Y Y

and flip it to get a subset of S that no element maps to

Y Y N N

we've found an element of the codomain $\mathcal{P}(S)$ which no element of the domain S maps to.

this shows that f is not surjective!

our choice of f was arbitrary so there are no bijections from S to $\mathcal{P}(S)$, thus $|S| \neq |\mathcal{P}(S)|$

Example: pigeonhole proof

PIGEONHOLE PRINCIPLE!! :)

General idea: if you have more items than bins, some bin has to have more than one item

Formally: if m objects are distributed into n bins, and $m > n$, then at least 1 bin will have at least 2 objects

Generalized: if you distribute m objects into n bins, then
- some bin has at least $\lceil \frac{m}{n} \rceil$ objects
- some bin has at most $\lfloor \frac{m}{n} \rfloor$ objects

↳ intuitively: you can't have everyone be too far from the average

USEFUL FOR: when we're putting objects into categories and we care about the counts of those categories

Let's begin with some new definitions. First, we'll say that a ***matching*** in a graph $G = (V, E)$ is a set $M \subseteq E$ of edges in G such that no two edges in M share an endpoint. The ***size*** of a matching is the number of edges it contains. The ***matching number*** of a graph G, denoted $\upsilon(G)$, is the size of the largest matching in G.

Now, let's introduce a variation on a definition we've seen before. A ***k-edge coloring*** of a graph $G = (V, E)$ is a way of coloring each of the *edges* in G one of k different colors so that no two edges that share an endpoint are assigned the same color. The ***chromatic index*** of a graph G, denoted $\chi_1(G)$, is the minimum number of colors needed in any edge coloring of G.

Let G be an undirected graph with exactly n^2+1 *edges* for some natural number $n \geq 1$. Prove that either $\chi_1(G) \geq n+1$ or $\upsilon(G) \geq n+1$ (or both).

How do you prove a statement of the form P or Q?

Example: pigeonhole proof

PIGEONHOLE PRINCIPLE!! ☺

General idea: if you have more items than bins, some bin has to have more than one item

Formally: if m objects are distributed into n bins, and $m > n$, then at least 1 bin will have at least 2 objects

Generalized: if you distribute m objects into n bins, then
- some bin has at least $\lceil \frac{m}{n} \rceil$ objects
- some bin has at most $\lfloor \frac{m}{n} \rfloor$ objects

↳ intuitively: you can't have everyone be too far from the average

USEFUL FOR: when we're putting objects into categories and we care about the counts of those categories

Let's begin with some new definitions. First, we'll say that a ***matching*** in a graph $G = (V, E)$ is a set $M \subseteq E$ of edges in G such that no two edges in M share an endpoint. The ***size*** of a matching is the number of edges it contains. The ***matching number*** of a graph G, denoted $\upsilon(G)$, is the size of the largest matching in G.

Now, let's introduce a variation on a definition we've seen before. A ***k-edge coloring*** of a graph $G = (V, E)$ is a way of coloring each of the *edges* in G one of k different colors so that no two edges that share an endpoint are assigned the same color. The ***chromatic index*** of a graph G, denoted $\chi_1(G)$, is the minimum number of colors needed in any edge coloring of G.

Let G be an undirected graph with exactly n^2+1 *edges* for some natural number $n \geq 1$. Prove that either $\chi_1(G) \geq n+1$ or $\upsilon(G) \geq n+1$ (or both).

How do you prove a statement of the form P or Q?

Proof: Let G be an arbitrary undirected graph with n^2+1 edges for some positive natural number n. We will prove that if $\chi_1(G) \leq n$, then $\upsilon(G) \geq n+1$.

Example: pigeonhole proof

PIGEONHOLE PRINCIPLE!! :)

General idea: if you have more items than bins, some bin has to have more than one item

Formally: if m objects are distributed into n bins, and $m > n$, then at least 1 bin will have at least 2 objects

Generalized: if you distribute m objects into n bins, then
- some bin has at least $\lceil \frac{m}{n} \rceil$ objects
- some bin has at most $\lfloor \frac{m}{n} \rfloor$ objects

↳ intuitively: you can't have everyone be too far from the average

USEFUL FOR: when we're putting objects into categories and we care about the counts of those categories

Let's begin with some new definitions. First, we'll say that a ***matching*** in a graph $G = (V, E)$ is a set $M \subseteq E$ of edges in G such that no two edges in M share an endpoint. The ***size*** of a matching is the number of edges it contains. The ***matching number*** of a graph G, denoted $\upsilon(G)$, is the size of the largest matching in G.

Now, let's introduce a variation on a definition we've seen before. A ***k-edge coloring*** of a graph $G = (V, E)$ is a way of coloring each of the *edges* in G one of k different colors so that no two edges that share an endpoint are assigned the same color. The ***chromatic index*** of a graph G, denoted $\chi_1(G)$, is the minimum number of colors needed in any edge coloring of G.

Let G be an undirected graph with exactly n^2+1 *edges* for some natural number $n \geq 1$. Prove that either $\chi_1(G) \geq n+1$ or $\upsilon(G) \geq n+1$ (or both).

What are the pigeons?
What are the holes?

How do you prove a statement of the form P or Q?

Proof: Let G be an arbitrary undirected graph with n^2+1 edges for some positive natural number n. We will prove that if $\chi_1(G) \leq n$, then $\upsilon(G) \geq n+1$.

Example: pigeonhole proof

PIGEONHOLE PRINCIPLE!! ☺

General idea: if you have more items than bins, some bin has to have more than one item

Formally: if m objects are distributed into n bins, and $m > n$, then at least 1 bin will have at least 2 objects

Generalized: if you distribute m objects into n bins, then
- some bin has at least $\lceil \frac{m}{n} \rceil$ objects
- some bin has at most $\lfloor \frac{m}{n} \rfloor$ objects

↳ intuitively: you can't have everyone be too far from the average

USEFUL FOR: when we're putting objects into categories and we care about the counts of those categories

Let's begin with some new definitions. First, we'll say that a ***matching*** in a graph $G = (V, E)$ is a set $M \subseteq E$ of edges in G such that no two edges in M share an endpoint. The ***size*** of a matching is the number of edges it contains. The ***matching number*** of a graph G, denoted $\upsilon(G)$, is the size of the largest matching in G.

Now, let's introduce a variation on a definition we've seen before. A ***k-edge coloring*** of a graph $G = (V, E)$ is a way of coloring each of the *edges* in G one of k different colors so that no two edges that share an endpoint are assigned the same color. The ***chromatic index*** of a graph G, denoted $\chi_1(G)$, is the minimum number of colors needed in any edge coloring of G.

Let G be an undirected graph with exactly n^2+1 *edges* for some natural number $n \geq 1$. Prove that either $\chi_1(G) \geq n+1$ or $\upsilon(G) \geq n+1$ (or both).

What are the pigeons?
What are the holes?

How do you prove a statement of the form P or Q?

Proof: Let G be an arbitrary undirected graph with n^2+1 edges for some positive natural number n. We will prove that if $\chi_1(G) \leq n$, then $\upsilon(G) \geq n+1$.

Suppose that $\chi_1(G) \leq n$. This means that there is an n-edge coloring of the graph G. Since there are n^2+1 edges and n colors, by the generalized pigeonhole principle we know that there must be at least $\lceil (n^2+1) / n \rceil = \lceil n + 1/n \rceil = n+1$ edges that are all the same color in the n-edge coloring. Since all those edges are assigned the same color, we know that no two of them can share an endpoint. Therefore, this set of $n+1$ edges forms a matching, so $\upsilon(G) \geq n+1$, as required. ■

Example: induction proof

Let's begin with a refresher of the key terms and definitions involved. As a reminder, if L_1 and L_2 are languages over an alphabet Σ, then the ***concatenation of L_1 and L_2***, denoted L_1L_2, is the language

$$L_1L_2 = \{ wx \mid w \in L_1 \text{ and } x \in L_2 \}.$$

From concatenation, we can define ***language exponentiation*** of a language L inductively as follows:

$$L^0 = \{\varepsilon\} \qquad L^{n+1} = LL^n$$

You may find these formal terms helpful in the course of solving this problem.

Let A and B be arbitrary languages over some alphabet Σ. Prove, by induction, that if $X = AX \cup B$, then $A^nB \subseteq X$ for every $n \in \mathbb{N}$. Please use the formal definitions of concatenation, language exponentiation, union, and subset in the course of writing up your answer.

What is P(n)?

What is the base case?

Build up or build down?

Do we need complete induction?

INDUCTION

if it starts true: $P(0)$ is true

and it stays true: $\forall k \in \mathbb{N}. (P(k) \rightarrow P(k+1))$

then its always true: $\forall n \in \mathbb{N}. P(n)$

proof setup:

- define a $P(n)$
- prove the base case ← choose simplest base case possible!
- prove the inductive step
  - build up if $P(n)$ is existentially quantified, build down if $P(n)$ is universally quantified
  - complete induction is useful if you're shrinking the problem but you don't know how much (eg. Nim)
    ↳ make sure you actually need all those assumptions!

Example: induction proof

INDUCTION

if it starts true: $P(0)$ is true

and it stays true: $\forall k \in \mathbb{N}. (P(k) \rightarrow P(k+1))$

then it always true: $\forall n \in \mathbb{N}. P(n)$

proof setup:

- define a $P(n)$
- prove the base case ← choose simplest base case possible!
- prove the inductive step
  - build up if $P(n)$ is existentially quantified, build down if $P(n)$ is universally quantified
  - complete induction is useful if you're shrinking the problem but you don't know how much (eg. Nim)
    ↳ make sure you actually need all those assumptions!

Let's begin with a refresher of the key terms and definitions involved. As a reminder, if L_1 and L_2 are languages over an alphabet Σ, then the ***concatenation of L_1 and L_2***, denoted L_1L_2, is the language

$$L_1L_2 = \{ wx \mid w \in L_1 \text{ and } x \in L_2 \}.$$

From concatenation, we can define ***language exponentiation*** of a language L inductively as follows:

$$L^0 = \{\varepsilon\} \qquad L^{n+1} = LL^n$$

You may find these formal terms helpful in the course of solving this problem.

Let A and B be arbitrary languages over some alphabet Σ. Prove, by induction, that if $X = AX \cup B$, then $A^nB \subseteq X$ for every $n \in \mathbb{N}$. Please use the formal definitions of concatenation, language exponentiation, union, and subset in the course of writing up your answer.

What is $P(n)$? What is the base case?

Build up or build down? Do we need complete induction?

Proof: Let A and B be arbitrary languages over some alphabet Σ where $X = AX \cup B$. Let $P(n)$ be the statement "$A^nB \subseteq X$." We will prove by induction that $P(n)$ is true for all $n \in \mathbb{N}$, from which the theorem follows.

As our base case, we prove $P(0)$, that $A^0B \subseteq X$. Consider any $w \in A^0B$. This string must be of the form xy where $x \in A^0$ and $y \in B$. Since the only string in A^0 is ε, this means that $w = \varepsilon y = y$, so $w \in B$. Then, since $w \in B$, we know that $w \in AX \cup B$, and therefore that $w \in X$. Since our choice of w was arbitrary, this shows that every element of A^0B is an element of X, so $A^0B \subseteq X$, as required.

For our inductive step, assume for some arbitrary $k \in \mathbb{N}$ that $P(k)$ holds and that $A^kB \subseteq X$. We will prove that $A^{k+1}B \subseteq X$. To do so, consider any arbitrary $w \in A^{k+1}B$. We will prove that $w \in X$.

Since $A^{k+1}B = AA^kB = A(A^kB)$, we know see that $w \in A(A^kB)$. Consequently, there exist some $x \in A$ and $y \in A^kB$ such that $w = xy$. Since $y \in A^kB$, by our inductive hypothesis we see that $y \in X$. Overall, this shows that $w = xy$ where $x \in A$ and $y \in X$, so we see that $w \in AX$. Since $w \in AX$, we see that $w \in AX \cup B$, or equivalently that $w \in X$, as required. Thus $P(k+1)$ is true, completing the induction. ■

General strategies

- Don't panic! You have a ton of mathematical tools you've been practicing with all quarter. No matter how daunting the problem may initially seem, I promise if you break it down step by step you'll find it's not as bad as it seems.
- Write out everything you know and what you're trying to prove.
    - What is the quantifier on the statement you're trying to prove? What does that tell you about how the proof should be set up?
    - What kind of structure are you trying to reason about? (binary relations, sets, functions, etc.) You know how to write proofs about all of these! Use proof templates and formal definitions to guide you.
    - What proof strategy are you using? What do you get to assume? If you're doing an indirect proof, would it be helpful to write out the statement in FOL and negate it?
- Make sure you're using all parts of what's given to you! Usually there's a good reason why you need each assumption/condition to get the proof to work.
- Draw pictures! Work backwards! Try a different proof strategy! It's okay if the first thing you try doesn't work, just try something else!

REGULAR LANGUAGES

problems you can solve with finite memory

to show a language is regular, you can:
- build a DFA for it
- build an NFA for it
- write a regex for it

DFA Design Tips

- states = pieces of information
  ↳ what do I have to keep track of in the course of figuring out whether something is in this language?
- transitions = updating state
  ↳ from the state I'm currently in, what do I know about my string? How would reading this character change what I know?

NFA Design Tips

- everything above, plus:
  embrace nondeterminism!
  ↳ assume that you'll magically know when to take the appropriate transitions

Regex Design Tips

- Write out some sample strings in the language and look for patterns:
  - can I separate out the strings into two (or more) categories?
    ↳ UNION - find the pattern for each category, then union together
  - can I break this problem down into solving some smaller subproblems?
    ↳ CONCATENATION - find the pattern for each piece/subproblem, then concatenate together
  - is there some sort of repeating structure?
    ↳ KLEENE STAR - find smallest repeating unit, then star that pattern

Example: DFA construction

Let $\Sigma = \{a, b\}$ and let $L_1 = \{ w \in \Sigma^* \mid w \text{ does not contain bbb as a substring} \}$.

Design a DFA for L_1.

DFA Design Tips

- states = pieces of information
  ↳ what do I have to keep track of in the course of figuring out whether something is in this language?
- transitions = updating state
  ↳ from the state I'm currently in, what do I know about my string? How would reading this character change what I know?

What are the states? How should my transitions link together the states?

Example: DFA construction

Let $\Sigma = \{a, b\}$ and let $L_1 = \{ w \in \Sigma^* \mid w \text{ does not contain bbb as a substring} \}$.

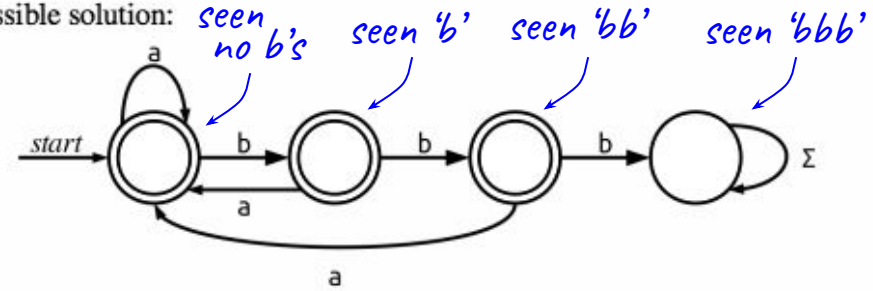
Design a DFA for L_1.

DFA Design Tips

- states = pieces of information
  - ↳ what do I have to keep track of in the course of figuring out whether something is in this language?
- transitions = updating state
  - ↳ from the state I'm currently in, what do I know about my string? How would reading this character change what I know?

states = "how much of the string 'bbb' have I seen?"

Here is one possible solution:



This automaton works by advancing forward every time it sees a b and resetting whenever it sees an a. If it finds three consecutive b's, it enters a dead state.

Example: regex construction

Regex Design Tips

- Write out some sample strings in the language and look for patterns:
  - can I separate out the strings into two (or more) categories?
    - ↳ UNION - find the pattern for each category, then union together
  - can I break this problem down into solving some smaller subproblems?
    - ↳ CONCATENATION - find the pattern for each piece/subproblem, then concatenate together
  - is there some sort of repeating structure?
    - ↳ KLEENE STAR - find smallest repeating unit, then star that pattern

Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Write a regular expression for L.

a

b

ab

ba

aba

bab

abab

baba

ababa

babab

...

Example: regex construction

Regex Design Tips

- Write out some sample strings in the language and look for patterns:
  - can I separate out the strings into two (or more) categories?
    - ↳ UNION - find the pattern for each category, then union together
  - can I break this problem down into solving some smaller subproblems?
    - ↳ CONCATENATION - find the pattern for each piece/subproblem, then concatenate together
  - is there some sort of repeating structure?
    - ↳ KLEENE STAR - find smallest repeating unit, then star that pattern

Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Write a regular expression for L.

Starts with a:	Starts with b:
a	b
ab	ba
aba	bab
abab	baba
ababa	babab
...	...

Example: regex construction

Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Write a regular expression for L.

Regex Design Tips

- Write out some sample strings in the language and look for patterns:
  - can I separate out the strings into two (or more) categories?
    - ↳ UNION - find the pattern for each category, then union together
  - can I break this problem down into solving some smaller subproblems?
    - ↳ CONCATENATION - find the pattern for each piece/subproblem, then concatenate together
  - is there some sort of repeating structure?
    - ↳ KLEENE STAR - find smallest repeating unit, then star that pattern

Starts with a:	Starts with b:
a	b
ab	ba
aba	bab
abab	baba
ababa	babab
...	...

a(sequence of ba's)(possibly another b)

Example: regex construction

Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Write a regular expression for L.

Regex Design Tips

- Write out some sample strings in the language and look for patterns:
  - can I separate out the strings into two (or more) categories?
    - ↳ UNION - find the pattern for each category, then union together
  - can I break this problem down into solving some smaller subproblems?
    - ↳ CONCATENATION - find the pattern for each piece/subproblem, then concatenate together
  - is there some sort of repeating structure?
    - ↳ KLEENE STAR - find smallest repeating unit, then star that pattern

Starts with a:	Starts with b:
a	b
ab	ba
aba	bab
abab	baba
ababa	babab
...	...

a(ba)*b?

Example: regex construction

Let $\Sigma = \{a, b\}$ and let $L = \{ w \in \Sigma^* \mid w$ is a nonempty string whose characters alternate between a's and b's $\}$. Write a regular expression for L.

Regex Design Tips

- Write out some sample strings in the language and look for patterns:
  - can I separate out the strings into two (or more) categories?
    - ↳ UNION - find the pattern for each category, then union together
  - can I break this problem down into solving some smaller subproblems?
    - ↳ CONCATENATION - find the pattern for each piece/subproblem, then concatenate together
  - is there some sort of repeating structure?
    - ↳ KLEENE STAR - find smallest repeating unit, then star that pattern

Starts with a:	Starts with b:
a	b
ab	ba
aba	bab
abab	baba
ababa	babab
...	...

$a(ba)^*b? \cup b(ab)^*a?$

MYHILL NERODE – showing languages are not regular!

General idea: non-regular languages = problems that can't be solved w/ finite memory

⛤ what do you need to remember to determine if a string is in the language?

↳ if you have to remember infinitely many things of one or infinitely many possibilities, the language is probably not regular!

Proof structure:

- find an infinite set S such that every pair of strings is distinguishable relative to L the language
  - use the answer to ⛤ to help find this set! have each string represent one of the infinitely many possibilities
- prove that these strings are distinguishable relative to the language
  - pick two arbitrary strings from S and show they're distinguishable call back to the definition of L as necessary - sometimes showing a string is in/not in a language is non-obvious!
- conclude the language is non-regular by Myhill Nerode

Example: Myhill-Nerode proof

Let $\Sigma = \{a, b\}$. Consider the following language L_2 over Σ:

$$L_2 = \{ a^n b^m \mid m, n \in \mathbb{N} \text{ and } m \leq 2n \}$$

For example, aa $\in L_2$ aab $\in L_2$ aabb $\in L_2$, aabbb $\in L_2$, and aabbbb $\in L_2$, but aabbbbb $\notin L_2$.

Prove that L_2 is not a regular language.

Ⓐ what do you need to remember to determine if a string is in the language?

Proof structure:

- find an infinite set S such that every pair of strings is distinguishable relative to L the language
  - use the answer to Ⓐ to help find this set!
    have each string represent one of the infinitely many possibilities
- prove that those strings are distinguishable relative to the language
  - pick two arbitrary strings from S and show they're distinguishable
    call back to the definition of L as necessary - sometimes showing a string is in/not in a language is non-obvious!

What is the answer to this question?
Based on that, what should S be?

Example: Myhill-Nerode proof

Let $\Sigma = \{a, b\}$. Consider the following language L_2 over Σ:

$$L_2 = \{ a^n b^m \mid m, n \in \mathbb{N} \text{ and } m \leq 2n \}$$

For example, aa $\in L_2$ aab $\in L_2$ aabb $\in L_2$, aabbb $\in L_2$, and aabbbb $\in L_2$, but aabbbbb $\notin L_2$.

Prove that L_2 is not a regular language.

Ⓐ what do you need to remember to determine if a string is in the language?

Proof structure:

- find an infinite set S such that every pair of strings is distinguishable relative to L the language
  - use the answer to Ⓐ to help find this set!
    have each string represent one of the infinitely many possibilities
- prove that those strings are distinguishable relative to the language
  - pick two arbitrary strings from S and show they're distinguishable
    call back to the definition of L as necessary - sometimes showing a string is in/not in a language is non-obvious!

We need to remember how many a's are in the string (to ensure that the number of b's is at most twice the number of a's)

Proof: Let $S = \{ a^n \mid n \in \mathbb{N} \}$. The set S is infinite because it contains one string for each natural number. Now, consider any two strings a^n, $a^m \in S$. Without loss of generality, assume that $n < m$. Now, consider the strings $a^n b^{2m}$ and $a^m b^{2m}$. The string $a^n b^{2m}$ is not in L_2 because $2m > 2n$, so there are too many b's in the string for it to be in L_2. On the other hand, the string $a^m b^{2m}$ is in L because the number of b's is precisely twice the number of a's. Therefore, we see that $a^n \not\equiv_{L_2} a^m$. Since our choices of a^n and a^m were arbitrary, we therefore see that any two distinct strings in S are distinguishable relative to L_2. Therefore, since S is infinite, by the Myhill-Nerode theorem we see that L_2 is not regular. ■

CONTEXT FREE GRAMMARS

Design Tips

- write out some example strings in the language
  - think recursively - can you find smaller strings of the language within larger strings?
  - common pattern: "for every x I see here, I need a y somewhere else in the string"
    - if two different parts of the string have to agree/match up with one another, they have to be built at the same time
  - non-terminals should represent different states / types of strings
    - ↳ eg. different phases of the build, different possible structures for the string

Example: CFG construction

Let $\Sigma = \{a, b\}$ and consider the following language L_5 over Σ:

$$L = \{ w \in \Sigma^* \mid |w| \equiv_3 0 \text{ and all the characters in the first third of } w \text{ are the same } \}$$

Here, aababa $\in L_5$, bbbaaaaaa $\in L_5$, aaa $\in L_5$, and $\varepsilon \in L_5$, but abbbbb $\notin L_5$ and aaaaa $\notin L_5$. (For convenience, I've underlined the first third of the characters in each string.)

Write a context-free grammar for L.

ε

abb

bab

aababa

aaaaaa

bbabbb

bbbaaaaaa

aaa

bbbbabbaa

CONTEXT FREE GRAMMARS

Design Tips

- write out some example strings in the language
  - think recursively - can you find smaller strings of the language within larger strings?
  - common pattern: "for every x I see here, I need a y somewhere else in the string"
    - if two different parts of the string have to agree/match up with one another, they have to be built at the same time
  - non-terminals should represent different states / types of strings
    - eg. different phases of the build, different possible structures for the string

Example: CFG construction

Let $\Sigma = \{a, b\}$ and consider the following language L_5 over Σ:

$$L = \{\ w \in \Sigma^* \mid |w| \equiv_3 0 \text{ and all the characters in the first third of } w \text{ are the same}\ \}$$

Here, aababa $\in L_5$, bbbaaaaaa $\in L_5$, aaa $\in L_5$, and $\varepsilon \in L_5$, but abbbbb $\notin L_5$ and aaaaa $\notin L_5$. (For convenience, I've underlined the first third of the characters in each string.)

Write a context-free grammar for L.

First third is a's:	First third is b's:
abb	bab
aababa	bbabbb
aaaaaa	bbbaaaaaa
aaa	bbbbabbaa
...	...

CONTEXT FREE GRAMMARS

Design Tips

- write out some example strings in the language
  - think recursively - can you find smaller strings of the language within larger strings?
  - common pattern: "for every x I see here, I need a y somewhere else in the string"
    - if two different parts of the string have to agree/match up with one another, they have to be built at the same time
  - non-terminals should represent different states / types of strings
    - ↳ eg. different phases of the build, different possible structures for the string

Example: CFG construction

Let $\Sigma = \{a, b\}$ and consider the following language L_5 over Σ:

$$L = \{ w \in \Sigma^* \mid |w| \equiv_3 0 \text{ and all the characters in the first third of } w \text{ are the same } \}$$

Here, aababa $\in L_5$, bbbaaaaaa $\in L_5$, aaa $\in L_5$, and $\varepsilon \in L_5$, but abbbbb $\notin L_5$ and aaaaa $\notin L_5$. (For convenience, I've underlined the first third of the characters in each string.)

Write a context-free grammar for L.

First third is a's:	First third is b's:
abb	bab
aababa	bbabbb
aaaaaa	bbbaaaaaa
aaa	bbbbabbaa
...	...

For every a in the first third, I need two other characters in the last two thirds

CONTEXT FREE GRAMMARS

Design Tips

- write out some example strings in the language
  - think recursively - can you find smaller strings of the language within larger strings?
  - common pattern: "for every x I see here, I need a y somewhere else in the string"
    - if two different parts of the string have to agree/match up with one another, they have to be built at the same time
  - non-terminals should represent different states / types of strings
    - eg. different phases of the build, different possible structures for the string

Example: CFG construction

Let $\Sigma = \{a, b\}$ and consider the following language L_5 over Σ:

$$L = \{ w \in \Sigma^* \mid |w| \equiv_3 0 \text{ and all the characters in the first third of } w \text{ are the same } \}$$

Here, aababa $\in L_5$, bbbaaaaaa $\in L_5$, aaa $\in L_5$, and $\varepsilon \in L_5$, but abbbbb $\notin L_5$ and aaaaa $\notin L_5$. (For convenience, I've underlined the first third of the characters in each string.)

Write a context-free grammar for L.

First third is a's:	First third is b's:
abb	bab
aababa	bbabbb
aaaaaa	bbbaaaaaa
aaa	bbbbabbaa
...	...

A -> aAXX | ε where X is "any character"

CONTEXT FREE GRAMMARS

Design Tips

- write out some example strings in the language
  - think recursively - can you find smaller strings of the language within larger strings?
  - common pattern: "for every x I see here, I need a y somewhere else in the string"
    - if two different parts of the string have to agree/match up with one another, they have to be built at the same time
  - non-terminals should represent different states / types of strings
    - ↳ eg. different phases of the build, different possible structures for the string

Example: CFG construction

Let $\Sigma = \{a, b\}$ and consider the following language L_5 over Σ:

$$L = \{ w \in \Sigma^* \mid |w| \equiv_3 0 \text{ and all the characters in the first third of } w \text{ are the same } \}$$

Here, aababa $\in L_5$, bbbaaaaaa $\in L_5$, aaa $\in L_5$, and $\varepsilon \in L_5$, but abbbbb $\notin L_5$ and aaaaa $\notin L_5$. (For convenience, I've underlined the first third of the characters in each string.)

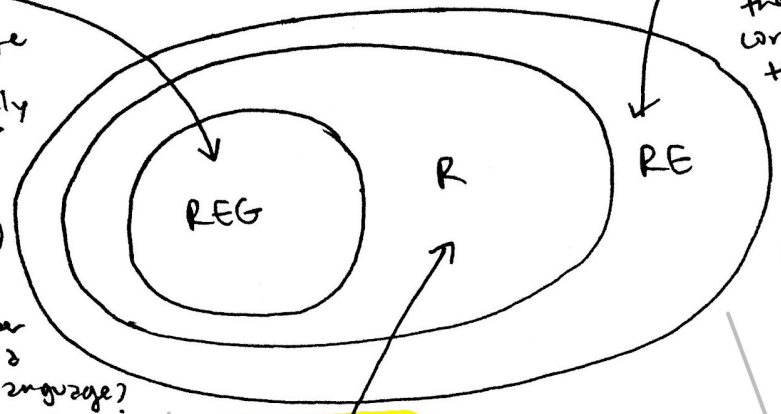
Write a context-free grammar for L.

One possibility is

$$S \to A \mid B$$
$$A \to aAXX \mid \varepsilon$$
$$B \to bBXX \mid \varepsilon$$
$$X \to a \mid b$$

The nonterminal A generates strings of the form $a^n\Sigma^{2n}$ and the nonterminal B generates strings of the form $b^n\Sigma^{2n}$, so overall the grammar generates all and only the strings in L.

LANDSCAPE OF COMPUTABILITY



REGULAR

- is the language finite?
  - YES → definitely regular
  - NO → could still be regular (eg. Σ^*)
- what do you have to remember to determine if a string is in the language?
  - if you only have to remember/check finitely many things, then the language is regular

assume that you can tell whether or not a program has property X. Then you can write a program:

if P has property X:
↳ P doesn't have property X

if P doesn't have property X:
↳ P has property X

DECIDABLE

- in addition to being recognizable, if you had something *not* in the language, could you convince someone of that fact?

GOOD INTUITION there is no single TM that can predict the behavior of all other TMs. If I claim I can predict what any TM will do, I can always create a self-referential program that goes against the prediction.

RECOGNIZABLE

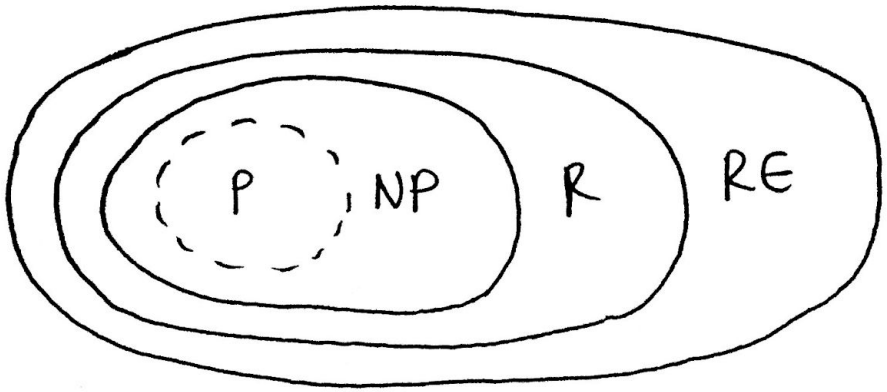
- if you had something in the language, could you convince someone else that that thing is in the language?
  - ↳ has to work for *all* things in the language, not just special cases!
  - ↳ good evidence convinces you something is in the language, bad evidence doesn't tell you anything

GOOD INTUITION

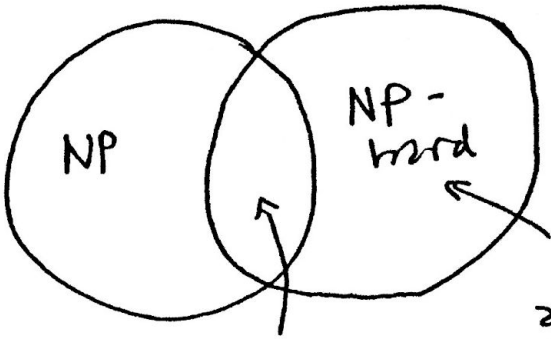
if you're trying to convince someone that something *won't* happen, the fact that it hasn't happened *yet* is not sufficient evidence of that fact.

→ keep this in mind when thinking of languages dealing with the *behavior* of TMs/programs

LANDSCAPE OF COMPLEXITY



computability	R = decidable can we solve it?	RE = verifiable can we check answers?
complexity	P = decidable in polynomial time can we solve it efficiently?	NP = verifiable in polynomial time can we check answers efficiently?



NP-complete
- "hardest of the problems in NP"
- if any one NP-complete problem is in P, then P = NP
- if any one NP-complete problem is not in P, then P ≠ NP