Advanced Computation 1 / CS 101

Alp Ozen

Fall 2019

Contents

! Notes starting in gray and ending with grey are from Prof.Lenstra's own lecture notes.

1 Propositional logic and notions of sets(Week 1 - Week 4)

1.1 Propositions

A proposition is a declarative sentence that is either true or false.

How much does it cost? **is not a proposition**
I like red **is a proposition**

To make life easier, we represent propositional statements through letters such as p.

The conditional statement $p \implies q$ appears very often. Thus, we have the *converse,contrapositive and inverse* which are:
converse: $q \implies p$
contrapositive: $\neg q \implies \neg p$
inverse: $\neg p \implies \neg q$

We note that a conditional is logically equivalent to its contrapositive.

p	q	$p \implies$ q	$\neg q$	$\neg p$	$\neg q \implies \neg p$
t	t	t	f	f	t
t	f	f	t	f	f
f	t	t	f	t	t
f	f	t	t	t	t

1.2 Precedence of logical operators

TABLE 8
Precedence of Logical Operators.

Operator	*Precedence*
$\neg$	1
$\wedge$	2
$\vee$	3
$\rightarrow$	4
$\leftrightarrow$	5

1.3 Fuzzy logic

In fuzzy logic, truth values are between 0 and 1. So if the statement "I like riding a bike" has a value of 0.8, it's negation has 1 minus this value, in this case -0.2.

1.4 Applications of logic

1.4.1 Logic gates

Here are the basic logic circuits from which more complex circuits are made:

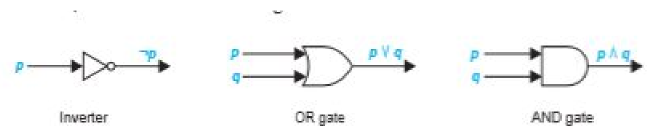


Figure 1: Logic gates

Note that the OR and AND gates accept only and only 2 inputs and output one. This input may be as compounded as possible but not exceed 'two chunks'. Thus, when given a complex logical output and reverse engineering, we identify the outer most outer operation, branch it into two or one(if it is simply a negation) and so on.

1.4.2 More on propositions

- **Tautology** is a compound proposition that is always true regardless of the truth value of its variables
- **Contradiction** is a compound proposition that is always false regardless of the truth value of its variables
- **Contingency** compound statement that is neither tautology nor contradiction

Example 1. $p \land \neg p$ is a contradiction
$p \lor \neg p$ is a tautology

Here are some examples of logical calculus:
Show that $\neg(p \lor (\neg p \land q))\neg p \land \neg q$

$$\begin{aligned} \neg(p \lor \neg p \land p \lor q) \\ \neg(T \land p \lor q) \\ \neg(p \lor q) \\ \neg p \land \neg q \end{aligned}$$

1.4.3 Satisfiability

A compound proposition is **satisfiable** if a truth assignment can be made to its variables that make it true making it either a tautology or a contingency. It is **unsatisfiable** if the negation of the compound statement is a contradiction.

1.5 Logical calculus and useful equivalences

Definition 1. If $A \iff B$ is a tautology, then A is logically equivalent to B.

Here are some useful logical equivalences(omitting most obvious ones):

$$p \implies q \equiv \neg p \vee q \equiv \neg(\neg q \vee p) \equiv \neg(q \implies p) \equiv \neg q \implies \neg p$$
$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$\wedge \text{ distributes over } \vee \text{ and vice versa}$$
$$p \vee \neg p \equiv T$$
$$p \wedge \neg p \equiv F$$
$$P \wedge T \equiv p$$
$$p \vee F \equiv p \text{ both } \wedge \text{ and } \vee \text{ are associative}$$

For more see this figure:

TABLE 7 Logical Equivalences Involving Conditional Statements.
$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv p \wedge \neg q$
$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to r) \wedge (q \to r) \equiv (p \vee q) \to r$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \vee (q \to r) \equiv (p \wedge q) \to r$

TABLE 8 Logical Equivalences Involving Biconditional Statements.
$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$
$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$
$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
$\neg(p \leftrightarrow q) \equiv p \leftrightarrow \neg q$

Figure 2: Logic, yey!

Definition 2. A **rule of inference** is based on the tautology $p \wedge (p \implies q) \implies q$. That is, whenever we are given that both p and $p \implies q$ is true, we infer that q must be true. That is :

$$\frac{p \quad (p \implies q)}{q}$$

Another important fact of logic is that we may boil down all of $\vee, \oplus, \implies, \iff$ to simply propositions involving$\neg, \wedge$:

$$p \vee q \equiv \neg\neg(p \vee q) \equiv \neg(\neg p \wedge \neg q)$$
$$p \oplus q \equiv \neg(p \wedge q) \wedge (p \vee q)$$
$$p \implies q \equiv \neg p \vee q \equiv \neg(p \wedge \neg q)$$
$$p \iff q \equiv \neg(\neg(p \wedge q) \wedge (\neg(\neg p \wedge \neg q))$$

Question *?* 1.5.1. Given propositional variables and truth values of the single variables for which the compound proposition takes a value, is there a way of deducing a compound proposition?

1.6 Lec.03 notes

The **contrapositive** is the following statement:

$$\begin{aligned} p \implies q &\equiv \neg p \vee q \\ &\equiv q \vee \neg p \\ &\equiv \neg q \implies \neg p \\ \therefore p \implies q &\equiv \neg q \implies \neg p \end{aligned}$$

Some useful logical equivalences involving implication:

$$\begin{aligned} (p \implies q) \wedge (p \implies r) &\equiv (\neg p \vee q) \wedge (\neg p \vee r) \\ &\equiv \neg p \vee (q \wedge r) \equiv p \implies (q \wedge r) \end{aligned}$$

And here's a more trivial one:

$$\begin{aligned} (p \implies q) \vee (p \implies r) &\equiv (\neg p \vee q) \vee (\neg p \vee r) \\ &\equiv \neg p \vee \neg p \vee q \vee r \equiv \neg p \vee (q \vee r) \\ &\equiv p \implies (q \vee r) \end{aligned}$$

And slightly more complicated involving De Morgan:

$$\begin{aligned} (p \implies r) \wedge (q \implies r) &\equiv (\neg p \vee r) \wedge (\neg q \vee r) \\ &\equiv \neg r \vee (\neg p \wedge \neg q) \equiv \neg r \vee (\neg (p \vee q) \\ &\equiv (p \vee q) \implies r \end{aligned}$$

We must also add some comments on base b systems of numbers and a general algorithm for conversion. Let's take an example in base 5. Suppose we want to convert 60_{10} to its base 5 representation. Well the largest power of 5 less than or equal to 60 is 25 and thus we know that 60 can be **uniquely** represented as a linear combination of the powers of 5 less than or equal to it. In fact, using powers of 5 less than or equal to 60, we may represent all numbers up to $5^k - 1$ as $4 \cdot 5^{k-1} + \ldots + 4 \cdot 5^0$. Thus, to represent some l in base b the algorithm is to find the largest power of $b^k < l$, perform $\lfloor \frac{l}{b^k} \rfloor$ then repeat step 1 and proceed as $l - b^k \cdot \lfloor \frac{l}{b^k} \rfloor$ and repeat until $l - b^i \cdot \lfloor \frac{l}{b^i} \rfloor = 0$

1.7 Lec.04 notes

1.7.1 More on CMOS

This was a lecture with a steep curve and here is a summary. First of all, we first revisit some of the concepts of the pmos and nmos resistors. A very important convention is that pmos must never be used for pull-down(connected to GND) and nmos never used for pullup(connected to VDD).

Some principles of cmos gates:

- Pmos goes to top, nmos to bottom.
- Never connect high voltage to low voltage to prevent a short circuit.
- Any circuit may be realized as a combination of the NAND and NOR ciruit.
- Each pmos must connect to an nmos.

The most challenging part of CMOS circuits is the circuit analysis itself. Consider this example to see a method of circuit analysis:

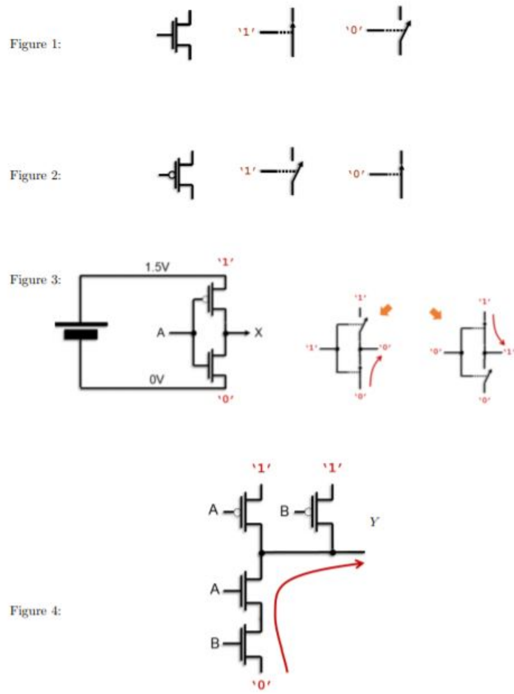
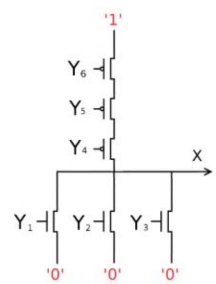


Figure 3: Caption



(*français*) La sortie X du circuit CMOS donné ci-dessus est égale à $\neg(A \vee B \vee C)$ si
(où $\overline{D}$ indique $\neg D$ pour un signal D)

(*English*) The output X of the CMOS circuit given above equals $\neg(A \vee B \vee C)$ if
(where $\overline{D}$ denotes $\neg D$ for a signal D)

Figure 4: exam question

Now, first of all, realize that X is connected to VDD iff all of $Y_6 \wedge Y_5 \wedge Y_4$ are grounded. That is $Y_6 \wedge Y_5 \wedge Y_4 = 0$. For symbolic purposes, supposing that $Y_i = 0 \equiv \neg Y_i$ we get that $X = 1 \iff \neg(Y_4 \vee Y_5 \vee Y_6)$ Now given that this is a CMOS circuit, we know that the bottom part does the exact opposite of the upper part. Thus, we have that $X = 0 \iff \neg(\neg(Y_4 \vee Y_5 \vee Y_6)) = Y_4 \vee Y_5 \vee Y_6 \equiv Y_1 \vee Y_2 \vee Y_3$. As a final step, for $\neg(A \vee B \vee C)$ to be true, we need that the output equals $\neg(A \vee B \vee C)$ and since $X = 1 \iff \neg(Y_4 \vee Y_5 \vee Y_6)$ we get that $Y_1 = Y_2 \ldots$

1.7.2 Binary addition circuit

Given that we can construct any compound logical gate using CMOS, suppose we want to implement a binary addition calculator. Now here are all the possible cases for doing binary addition:

a	b	c_{in}	c_{out}	s	
0	0	0	0	0	(in binary $0+0+0$ equals the 2-bit string 00)
0	0	1	0	1	(in binary $0+0+1$ equals the 2-bit string 01)
0	1	0	0	1	(in binary $0+1+0$ equals the 2-bit string 01)
0	1	1	1	0	(in binary $0+1+1$ equals the 2-bit string 10)
1	0	0	0	1	(in binary $1+0+0$ equals the 2-bit string 01)
1	0	1	1	0	(in binary $1+0+1$ equals the 2-bit string 10)
1	1	0	1	0	(in binary $1+1+0$ equals the 2-bit string 10)
1	1	1	1	1	(in binary $1+1+1$ equals the 2-bit string 11)

Figure 5: Addition possibilities

Now suppose we want a function $f(a, b, c_{in})$ to evaluate s. Well notice that s is only true when the parity of a, b, c_{in} is odd. That is, we may describe this outcome with the function $a \oplus b \oplus c_{in}$. Similarly, devising $g(a, b, c_{in})$ to compute c_{out} we notice that c_{out} evaluates to 1 iff at least two variables are true. This is equivalent to $(a \wedge b) \vee (a \wedge c_{in}) \vee (b \wedge c_{in})$

1.7.3 Fast Multiplication aka. Karatsuba

We now ponder whether there is a quick way of multiplying some v and w. Now notice that for v and w in base 10, $v = aX + b$ and $w = cX + d$. Now notice that $v \cdot w = (aX + b)(cX + d)$ which in turn is:

$$v \cdot w = acX^2 + (ad + bc)X + bd$$

And further notice that:

$$ad + bc = (ac + bd) - (a - b)(c - d)$$

to get:

$$v \cdot w = acX^2 + ((ac + bd) - (a - b)(c - d))X + bd$$

Now if for instance v and w were 2 digit numbers, we would normally perform 4 digit by digit multiplications but with this new method, we end up performing only 3 and a trivial subs traction.
As a general result, for multiplication of two k by k digit numbers, we end up performing $3^{\log_2 k}$ multiplications and $log_2 k$ many additions. Finally, notice that Karatsuba is a recursive algorithm.

1.7.4 Two's compliment

Consider how a computer is to represent negative integers. A very smart way of doing so is **two's compliment**. That is given a binary representation, we invert all 1's with 0's and all 0's with 1's and then add 1. Note that now the 0's take the role of 1's and vice versa. The reason for adding 1 is that the most significant digit is reserved for the sign. A 1 is a negative, a 0 a positive.

Remark 1.7.1. Note that when multiplying two numbers with non-matching number of digits, we simply pad both numbers with 0's until both have number of digits that are a power of 2.

1.8 Lec 05. notes

1.8.1 Main points

In this lecture quantifiers and their properties were discussed along with common pitfalls.
Consider defining a proposition on some sub-domain. That is take $D = \{0, 1, 2\}, S = \{2\}$ Now we want to express the proposition $\forall x \in S, P(x)$ where $P(x)$ is to mean that x is even in the form $\forall x Q(x)$. A major mistake made is to try and express this as $x \in S \wedge P(x) \equiv Q(x)$ Now clearly $\forall x Q(x) \not\equiv \forall x \in S, P(x)$ as taking $x = 1$ leads to the LHS being false. So here is the correct way to do this: Let $Q(x) \equiv x \in S \to P(x)$. It is now the case that $\forall x Q(x) \equiv \forall x \in S, P(x)$. Therefore we have that:

$$\forall x \in S, P(x) \equiv \forall x(x \in S \to P(x)) \tag{1.8.1}$$
$$\exists x \in T P(x) \equiv \exists x(x \in T \wedge P(x)) \tag{1.8.2}$$

Remark 1.8.1. Note that both $\exists$ and $\forall$ have precedence over any other logical operator.

We would also like to point out that whenever we have an empty domain, then both $\exists x P(x)$ and $\exists x \neg P(x)$ are both false. Similarly, $\forall x P(x)$ and $\forall x \neg P(x)$ are both true.
We now ask, how do we negate the quantifiers?

$$\neg(\exists x P(X)) \equiv \forall x \neg P(x) \tag{1.8.3}$$
$$\neg(\forall x P(X)) \equiv \exists x \neg P(x) \tag{1.8.4}$$

And we add the note that the same negation rules also hold for quantifiers over a sub-domain. Here is an example proof:

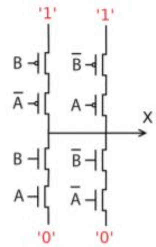
$$\begin{aligned} \equiv \neg\forall x \in S\ P(x) &\equiv \neg(\forall x \in S \to P(x)) \\ &\equiv \exists x \neg(x \in S \to P(x)) \\ &\equiv \exists x \neg(x \notin S \vee P(x)) \\ &\equiv \exists x(x \in S \wedge \neg P(x)) \\ &\equiv x \in S,\ \neg P(x) \end{aligned}$$

1.9 Lec 06. notes

First of all, we note that when solving CMOS problems, always check if the circuit is complementary. That is whenever the upper part is 1, lower part must be disconnected. Hence if the upper part evaluates $A \wedge B$, lower part must evaluate $\neg(A \wedge B)$

Here is a nice CMOS circuit problem from week 3:

Exercise 4. Consider the following circuit (where $\overline{Y}$ for a signal Y denotes the complementary signal $\neg Y$; thus, $Y = 0$ if and only if $\overline{Y} = 1$):



As a function of the inputs A and B, the output X satisfies:

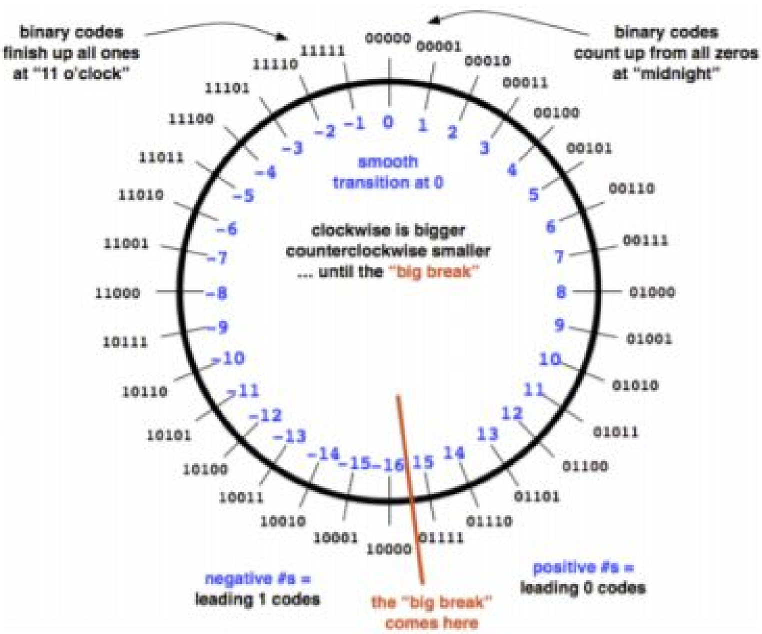
- ○ $X = \neg(A \leftrightarrow B)$
- ○ $X = A \leftrightarrow B$
- ○ $X = A \vee B$
- ○ the circuit may be shorted (connecting '0' to '1')

Figure 6: PSET3 cmos problem

Here's how I solved it:
Now notice that $X = 1$ iff. $(B = 0 \wedge \neg A = 0) \vee (\neg B = 0 \wedge A = 0)$ Thus we obtain:

$$\begin{aligned}
&(\neg B \wedge A) \vee (B \wedge \neg A) \\
&\equiv \neg(B \vee \neg A) \vee \neg(\neg B \vee A) \\
&\equiv \neg(A \to B) \vee \neg(B \to A) \\
&\equiv \neg((A \to B) \wedge (B \to A)) \\
&\equiv \neg(A \iff B)
\end{aligned}$$

And here is some more details on two's compliment:



For the exercises below a bit more on what was mentioned in the September 20 lecture notes.

For a k-bit string n the *ones' complement* is defined as $C_1(n) = 2^k - 1 - n$, and the two's complement of n is defined as $C_2(n) = C_1(n) + 1 = 2^k - n$. When working with k-bit strings on a k-bit computer architecture, all bits beyond the k-th bit are simply chopped off: this results in what is referred to as *arithmetic modulo* 2^k. It follows that for integers a with $0 \le a < 2^{k-1}$ (represented on a k-bit architecture as a k-bit string with a leading zero followed by $k-1$ bits) it is convenient to represent $-a$ as the k-bit string $C_2(a)$: this is easily computed given a by complementing all its k bits (i.e., compute $C_1(n)$) and adding 1. The value $a = 0$ is represented as a string of k zeros: verify that $-a = 0$ is again represented by k zeros (because anything beyond the k-th bit is chopped off). Thus, zero has a unique representation. The 2^{k-1} integers a with $0 \le a < 2^{k-1}$ are the k-bit strings with a leading zero. The other 2^{k-1} k-bit strings (because in total there are $2^k = 2^{k-1} + 2^{k-1}$ k-bit strings) all have a leading one, and are used to represent all negative numbers in the range from -1 down to and including -2^{k-1}: verify that $-a = C_2(a)$ for $-2^{k-1} < a \le 1$ (these are $2^{k-1} - 1$ distinct numbers) and that the value -2^{k-1} satisfies $-2^{k-1} = C_2(-2^{k-1})$ (implying that -2^{k-1} represents its own negative which is correct given that 2^k is regarded as 0). For $k = 5$ the situation is depicted in the figure above.

It follows that for all k-bit strings a it is the case that $C_2(C_2(a)) = a$: the negative of the negative of a is a itself again, as one would expect. Using $C_2(a)$ as the negative of a $k-1$-bit number on k-bit architectures is referred to as *two's complement arithmetic*. It is convenient because, for $k-1$-bit binary integers a and b the value $a - b$ is computed as $a + C_2(b)$. This holds for $-2^{k-1} \le a, b < 2^{k-1}$(both positives and negatives): note that -2^{k-1} is included in this range as well, but 2^{k-1} is not. This replaces the need for a subtraction circuit by first computing the two's complement of b followed by an application of the addition circuit, and holds because all arithmetic is modulo 2^k . Also note that the range of numbers $[-2^{k-1}, 2^{k-1} - 1]$ that can be represented when using k-bit two's complement arithmetic is asymmetric, unlike the alternative of naively using the leftmost bit as the sign in which case the range of numbers $[-(2^{k-1} - 1), 2^{k-1} - 1]$ that can be represented is symmetric (due to the fact that 0 and -0 have different representations, namely $0 = \underbrace{000\cdots000}_{k}$ and $-0 = 1\underbrace{000\cdots00}_{k-1}$); a much greater disadvantage (than the unnecessary representation of -0) is the fact that the naive approach requires separate subtraction circuitry.

Figure 7: More on two's compliment

And now let's say a little more about two's complement. That is, two's compliment simply represents the additive inverse of a number in $\bmod 2^k$ That is, suppose we wanted some b such that $a + b \equiv 0 \bmod 2^k$, then we'd have that $b = 2^k - a$ Which is essentially the formula for two's complement for a given bit architecture.

1.10 Lec 07. notes

We consider how to negate the $\exists!$ expression. Realize that:

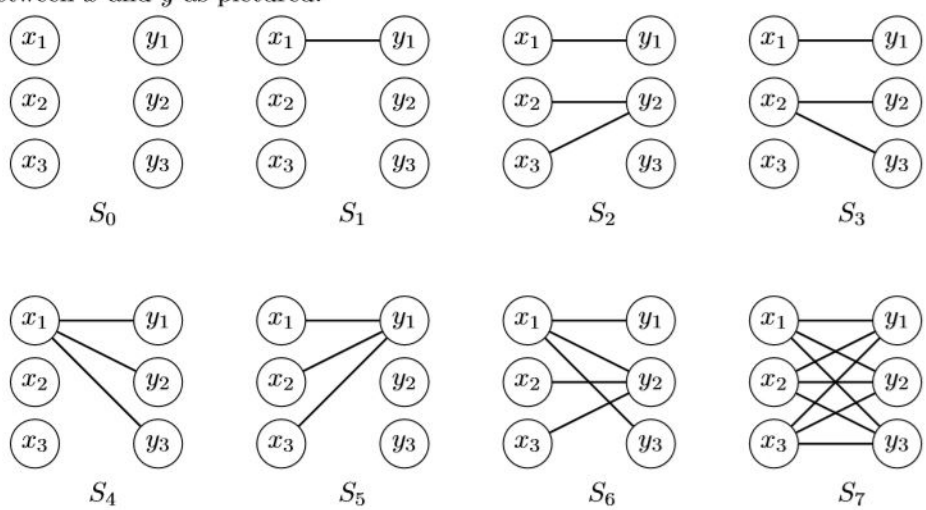
$$\exists! x P(x) \equiv \exists x (P(x) \land P(y) \to (x = y))$$

And hence the negation(using our negation laws gives):

$$\neg \exists! x P(x) \equiv P(x) \to \exists y \neq x P(y))$$

And two demonstrate the importance of nesting order on quantifiers, consider this excerpt:

Nesting order. To show the effect of different nesting orders of different quantifiers, let X be a set $\{x_1, x_2, x_3\}$ of three vertices, let Y be another set $\{y_1, y_2, y_3\}$ of three vertices, and let $S_i(x, y)$ for $0 \le i < 8$ be the eight distinct propositional functions from $X \times Y$ to $\{0, 1\}$ where $S_i(x, y)$ is true if and only if there is an edge between x and y as pictured:



S	$\exists x \exists y\, S(x,y)$	$\exists x \forall y\, S(x,y)$	$\to$	$\forall y \exists x\, S(x,y)$	$\exists y \forall x\, S(x,y)$	$\to$	$\forall x \exists y\, S(x,y)$	$\forall x \forall y\, S(x,y)$
S_0	false	false		false	false		false	false
S_1	true	false		false	false		false	false
S_2	true	false		false	false	$\not\leftarrow$	true	false
S_3	true	false	$\not\leftarrow$	true	false		false	false
S_4	true	true	$\to$	true	false		false	false
S_5	true	false		false	true	$\to$	true	false
S_6	true	true	$\to$	true	true	$\to$	true	false
S_7	true	true	$\to$	true	true	$\to$	true	true

Figure 8: Nesting order demo

And notice here how $\exists x \forall y S(x, y) \to \forall y \exists x S(x, y)$ This makes sense as whenever an x exists for each y, we have to have that for all y, there is an x.

And now we come to rules of inference for quantifiers.

Definition 3. (Inference laws for quantified statements)
Universal instantiation

$$\frac{\forall x P(x)}{P(c)}$$

Universal generalization

$$\frac{P(x)\text{for arbitrary x}}{\forall x P(x)}$$

And we note that the same hold respectively for the existential quantifier.

And finally we present an application of rules of inference. Suppose the following:
$H(x)$: x is here
$U(x)$: x likes C
$L(x)$ x is a fan of D.R.(Denis Ritchie)
Now we assert the following:
(1) There is someone here who likes C
(2) Everyone who likes C is a fan of D.R.
Hence we get:

$$(1) \equiv \exists x(H(x) \wedge U(x))$$
$$(2) \equiv \forall x(U(x) \rightarrow L(x))$$

Now what may we infer from these?
Well we know $\exists x(H(x) \wedge U(x))$ hence we by instantiation we have $H(c) \wedge U(c)$ then $U(c), H(c)$ and since $\forall x(U(x) \rightarrow L(x))$ to give $U(c) \rightarrow L(c), L(c)$ and finally

$$H(c) \wedge L(c)$$

And although it is not too useful to memorize their names, we include here a table of common rules of inference:

TABLE 1 Rules of Inference.

Rule of Inference	*Tautology*	*Name*
p $p \rightarrow q$ $\therefore q$	$(p \wedge (p \rightarrow q)) \rightarrow q$	Modus ponens
$\neg q$ $p \rightarrow q$ $\therefore \neg p$	$(\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$	Modus tollens
$p \rightarrow q$ $q \rightarrow r$ $\therefore p \rightarrow r$	$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$	Hypothetical syllogism
$p \vee q$ $\neg p$ $\therefore q$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctive syllogism
p $\therefore p \vee q$	$p \rightarrow (p \vee q)$	Addition
$p \wedge q$ $\therefore p$	$(p \wedge q) \rightarrow p$	Simplification
p q $\therefore p \wedge q$	$((p) \wedge (q)) \rightarrow (p \wedge q)$	Conjunction
$p \vee q$ $\neg p \vee r$ $\therefore q \vee r$	$((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$	Resolution

Figure 9: Rules of inference

1.11 Lec 08. notes

1.11.1 Proof methods

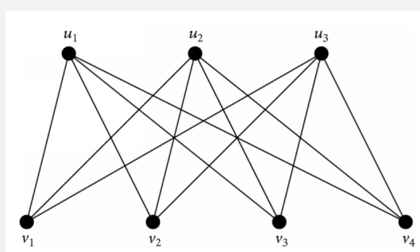
A **proof** is a valid argument establishing the truth of a statement.
We list some common proof methods and elaborate:
Direct proof: When trying to prove a statement like $p \to q$ we consider the only case that $p \to q$ would be false and show that it can not happen. That is we take p true and using our inference laws, reach that q must be true as well
Proof by contraposition: Since $p \to q \equiv \neg q \to \neg p$ we try to show that the contrapositive holds via a direct proof. And here's a mini-example: Consider the statement *if n is an integer then $3n + 2$ is odd.* Now suppose $3n + 2$ is even to give $3n + 2 = 2k,\ k \in \mathbb{N}$ Then we have that $n = \frac{2k-2}{3}$ and taking $k = 2$ suffices to show n is not an integer.
Proof by contradiction: Firstly, proof by contradiction is often confused with proof by contraposition. We use proof by contradiction to show that a statement p is true. To do this, if we can show that $\neg p \to q$ is true where q is always false, we have that $\neg p$ is false hence p true. A common example is showing that $\sqrt{2}$ is irrational. We take the negation of p that $\sqrt{2}$ is rational and derive the contradiction that whenever we try to write $\sqrt{2} = \frac{z}{k}$ with $\gcd zk = 1$ that $\gcd zk = 1 \wedge \gcd zk \neq 1$ Thus the falsity of p implies a contradiction(false value) showing that p itself must be true.
Similarly suppose we have statement $p \equiv a \to b$. We know that $\neg p \to a \wedge \neg b$ and get that $a \wedge \neg b$ is always false showing that p must be true.

1.11.2 Graphs and planarity

A planar graph is one that can be drawn without overlapping edges. Now a planar graph does not imply that for all possible drawing, overlapping edges wont exist. It simply means that a drawing could be made without overlapping edges. Now the two most important facts of graphs are:

($K_{3,3}$ and K_5 are not planar)

Now notice that each connection we make partitions the plane into an inner and outer side following the **Jordan theorem**. We find out that when it comes to drawing the last vertex for $K_{3,3}$, the edge is surrounded by 4 curves making it impossible to not cross a curve.



Now, **Kuratowski's theorem** states that a graph is non-planar iff it contains in one way $K_{3,3}$ or K_5

1.11.3 Different surfaces

We now consider different surfaces, those of the **Torus, Mobius strip, Klein bottle** shown respectively in the figure below.

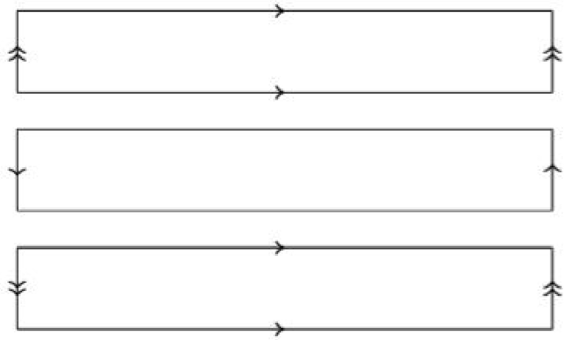
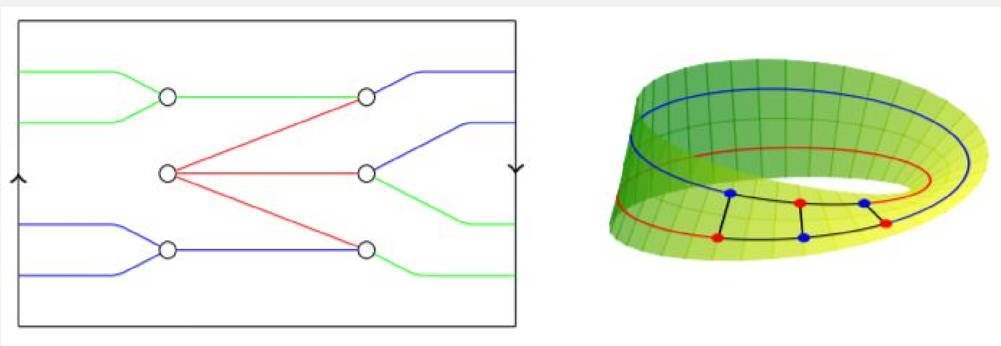


Figure 10: Rectangular abstraction

These surfaces are interesting because the laws of planarity no longer hold. For instance, on a mobius strip, $K_{3,3}$ is indeed planar as below.

($K_{3,3}$ has become planar)



1.12 Lec 09. notes

1.12.1 Sequences

We define a **sequence** as a mapping $f : \mathbb{N} \to \mathbb{R}$. Some most essential sequences are:

$$\begin{aligned} \textbf{constant sequence } &\exists c \forall i \ a_i = c \\ \textbf{arithmetic sequence } &\exists c \forall i \ a_{i+1} - a_i = c \\ \textbf{geometric sequence } &\exists c \forall i \ \frac{a_{i+1}}{a_i} = c \end{aligned}$$

More generally, the constant and arithmetic sequence are a member of the family of consecutive polynomial sequences. That is, since a polynomial is defines as:

$$\sum_{i=0}^{j} a_i X^i$$

we may have sequences of higher degree.

1.12.2 Summations

Consider the summation of the first n terms of a constant sequence $a_i = c$. We would have $\sum_{i=1}^{n} a_i$ which is the same as $\sum_{i=1}^{n} c$ and taking $c = 1$ we get n times 1 added which is $\sum_{i=1}^{n} c = 1$
Let's now consider the summation of an arithmetic sequence by using an nxn square as below:

Figure 11: Summations, yey!

We agree that at each step there are $2i - 1$ colored crosses which add up to n^2 many crosses. So we express it as $\sum_{i=1}^{n}(2i-1) = n^2$ and try to get a useful formula out of it.
Well we have that $\sum_{i=1}^{n}(2i-1) = \sum_{i=1}^{n}(2i) - \sum_{i=1}^{n}(1)$ Which further gives $2\sum_{i=1}^{n}(i) - \sum_{i=1}^{n}(1) = n^2$ hence we obtain

$$\sum_{i=1}^{n}(i) = \frac{n^2+n}{2}$$

We now consider more interesting summations involving **telescoping sequences**. A telescoping sequence is a sequence where during summation of the terms (parts of) consecutive terms cancel each other, so that only few (parts of) terms remain to be added. To illustrate this, consider the sequence $\{b_i\}$ for $i > 0$ with

$$b_i = i - (i-1),$$

and let $\{a_i\}$ be a constant sequence with $a_i = 1$. Obviously, it follows that $b_i = 1$ so that $b_i = a_i$ and thus $\sum_{i=1}^{n} b_i = \sum_{i=1}^{n} a_i$. The latter sum was already calculated, which was a trivial exercise but in principle still required adding together n terms all equal to 1. Using that $\sum_{i=1}^{n} b_i = \sum_{i=1}^{n} a_i$ the same sum $\sum_{i=1}^{n} a_i$ can be computed by computing $\sum_{i=1}^{n} b_i$ instead: due to the telescoping effect, this computation does not require any actual calculations at all, but just involves careful administration:

$$\begin{aligned}
\sum_{i=1}^{n} b_i &= b_n + b_{n-1} + b_{n-2} + \ldots + b_2 + b_1 \\
&= \underbrace{n-(n-1)}_{b_n} + \underbrace{n-1-(n-2)}_{b_{n-1}} + \underbrace{n-2-(n-3)}_{b_{n-2}} + \ldots + \underbrace{2-(2-1)}_{b_2} + \underbrace{1-(1-1)}_{b_1} \\
&= n \underbrace{-(n-1)+n-1}_{=0} \underbrace{-(n-2)+n-2}_{=0} \underbrace{-(n-3)+\ldots}_{=0} \ldots \underbrace{\ldots+2}_{=0} \underbrace{-1+1}_{=0} - 0 \\
&= n.
\end{aligned}$$

Yet a much more formal way of showing this as follows:

$$\begin{aligned}
\sum_{i=1}^{n} b_i &= \sum_{i=1}^{n}(i-(i-1)) \\
&= \Big(\sum_{i=1}^{n} i\Big) - \Big(\sum_{i=1}^{n}(i-1)\Big) \quad \text{(in 2nd sum replace } i-1 \text{ by } j) \\
&= \Big(\sum_{i=1}^{n} i\Big) - \Big(\sum_{j=0}^{n-1} j\Big) \quad \text{(replace } j \text{ by } i) \\
&= \Big(\sum_{i=1}^{n} i\Big) - \Big(\sum_{i=0}^{n-1} i\Big) \quad \text{(identify the part that occurs in both sums)} \\
&= \Big(n + \sum_{i=1}^{n-1} i\Big) - \Big(0 + \sum_{i=1}^{n-1} i\Big) \\
&= n - 0 + \Big(\sum_{i=1}^{n-1} i\Big) - \Big(\sum_{i=1}^{n-1} i\Big) \\
&= n.
\end{aligned}$$

We now consider a different approach to geometric sequences. We are interested in a closed form expression for $\sum_{i=0}^{n} g_i$, where for relevant values of r it may even make sense to talk about the value of $\sum_{i=0}^{\infty} g_i$. Because

$$\sum_{i=0}^{n} g_i = \sum_{i=0}^{n} g_0 r^i = g_0 \sum_{i=0}^{n} r^i$$

the value g_0 is just an uninteresting multiplicative factor, and we omit it (i.e., from now on we assume that $g_0 = 1$; note that the entire sum equals zero if $g_0 = 0$). Obviously, the problem is not interesting if $r = 1$: then we get (with $g_0 = 1$) that

$$\sum_{i=0}^{n} g_i = \sum_{i=0}^{n} r^i = \sum_{i=0}^{n} 1^i = \sum_{i=0}^{n} 1 = n+1.$$

Also $r = -1$ is not inspiring: we get 0 if n is odd and 1 if n is even; but the final arguments below work for $r = -1$ as well, so $r = -1$ is not excluded below.

Assuming $r \neq 1$, we want to find a closed form expression for $\sum_{i=0}^{n} r^i$, i.e., for

$$r^0 + r^1 + \ldots + r^{n-2} + r^{n-1} + r^n.$$

It was noted that we are already familiar with a similar sum, namely

$$r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0$$

which is just the same, but in the reverse order. How come we are familiar with something like $r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0$? Just look at the decimal number $111\ldots11$ that consists of $n+1$ ones: that is precisely the same as our expression $r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0$ if we assume that $r = 10$ (the *basis* or *radix* of our decimal number system) because our decimal number $111\ldots11$ is just a convenient shorthand notation for

$$1 \times 10^n + 1 \times 10^{n-1} + 1 \times 10^{n-2} + \ldots + 1 \times 10^1 + 1 \times 10^0 :$$

that is how the value of a number in decimal notation is defined, where the present case is particularly simple because all the digits are equal to one.

Taking for instance $n = 4$, we find the number 11111, and we know that if we multiply it by 9 (which was carefully chosen as $10 - 1$, i.e., as $r - 1$) we get, without any effort, 99999, so that if we add one we get 100000, for which we have the nice closed form expression 10^5 (i.e., r^{n+1}). We thus found that

$$(11111 \times 9) + 1 = 10^5$$

and more in general that

$$(\underbrace{111\ldots11}_{n+1 \text{ ones}} \times 9) + 1 = 10^{n+1},$$

or even more in general that

$$(\underbrace{111\ldots11}_{n+1 \text{ ones}} \times (r-1)) + 1 = r^{n+1},$$

where the final $111\ldots11$ should be liberally interpreted as a shorthand notation for $r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0$ and where we use that 10 was just a placeholder for r and thus that 9 was a placeholder for $r - 1$. This final expression leads us to believe that

$$\underline{(r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0) \times (r-1)} + 1 = r^{n+1}$$

and thus that

$$r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0 = \frac{r^{n+1} - 1}{r - 1}$$

(where you should remember that indeed $r \neq 1$). But this is precisely the closed formula for $\sum_{i=0}^{n} r^i$ that we were trying to find, so we are done – except we have not proved anything yet other than by intimidation.

To actually prove this result that

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

if $r \neq 1$, have a look at the underlined part

$$(r^n + r^{n-1} + r^{n-2} + \ldots + r^1 + r^0) \times (r-1)$$

of the equation above: apparently we need to prove that this underlined part is equal to $r^{n+1} - 1$. Pulling the factor $r - 1$ inside the parentheses we get

$$(r-1)r^n + (r-1)r^{n-1} + (r-1)r^{n-2} + \ldots + (r-1)r^1 + (r-1)r^0$$

and thus

$$r^{n+1} - r^n + r^n - r^{n-1} + r^{n-1} - r^{n-2} + \ldots + r^2 - r^1 + r^1 - r^0.$$

This is yet another example of our earlier telescoping sum: all the intermediate terms cancel each other.

In any case, the conclusion is that if $r \neq 1$ then

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1} - 1}{r - 1}$$

for any integer $n \geq 0$. It follows that if $|r| < 1$ we have that

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1 - r}$$

because for $n \to \infty$ the term $r^{n+1} \to 0$.

We finally conclude by representing a useful identity as follows:

$$\sum_{i=1}^{\infty} \sum_{j=1}^{i} \cdots = \sum_{j=1}^{\infty} \sum_{i=j}^{\infty} \cdots$$

1.12.3 Sum of i^k

And yet another important point to mention is that all the sums of form i^k may be found using the fact that the telescoping sequence given by

$$f(k+1,i) = i^{k+1} - (i-1)^{k+1}$$

since using the telescoping property we have that

$$\sum_{i=1}^{n} i^{k+1} - (i-1)^{k+1} = n^{k+1}$$

which means that

$$\sum_{i=1}^{n} i^{k+1} = n^{k+1} + \sum_{i=1}^{n} (i-1)^{k+1}$$

where the $\sum_{i=1}^{n}(i-1)^{k+1}$ are recursively found.

1.12.4 Hilbert's Hotel

Some interesting questions to pose on this matter are listed below with answers.

Question ? 1.12.1. A bus carrying countably infinite number of guests arrives, can guests be accommodated?

Answer 1.12.1. Well simply assign each former guest to odd numbered room and each new guest to even numbered room.

Question ? 1.12.2. Now suppose that countably infinite number of buses with countably infinite guests arrive what now?

Answer 1.12.2. Well, we move each guest in room k to room $2k-1$. Now for new guests, kth guest in bus 1 moves to $2^1(2k-1)$ hence in general kth guest in bus j moves to $2^j(2k-1)$ room.

2 Algorithms and more (Week 5 -)

2.1 Lec 10. notes

To demonstrate how summations come in handy consider snipets of following pseudocode

```
B <- n while(B>0){step;}
```

Leads to ∞ steps.

`B < - n while(B>0){for i=1 to B do step}){B =` $\lfloor \frac{B}{2} \rfloor$`}`

Now notice we first perform n, then approximately $\frac{n}{2}$ steps and so on. Thus in total we have $\sum_{i=0}^{\log_2 n}(\frac{n}{2^j})$ many steps where this approximates to $2n$
And now sum must now summation results:

- $\sum_{i=1}^{n} c = cn$: the sum of n constants is linear in n.
- $\sum_{i=1}^{n} ci = \frac{cn(n+1)}{2}$: the sum of n consecutive linearly growing values is quadratic in n;
- $\sum_{i=1}^{n} ci^2 = \frac{cn(n+1)(2n+1)}{6}$: the sum of n consecutive quadratically growing values is cubic in n;
- $\sum_{i=1}^{n} ci^k = \frac{n^{k+1}}{k+1}$ + "lower order terms": the sum of n consecutive values of a polynomial of degree k is of degree $k+1$ in n (note that the $\frac{1}{k+1}$ is just a constant multiplicative factor: the only thing that "really counts" is the n^{k+1}: next time we will see what is meant by this "really counts");
- The latter result should not be confused with $\sum_{i=0}^{k} n^i$ which is "just" a polynomial of degree k evaluated in n and thus (for growing n) bounded by $(k+1)n^k$ (again, the $k+1$ is just a constant multiplicative factor: the only thing that "really counts" is the n^k);
- $\sum_{i=0}^{n} cr^i = \frac{c(r^{n+1}-1)}{r-1}$ where $r \neq 1$. For $n \to \infty$ and $|r| < 1$ we get $\sum_{i=0}^{\infty} cr^i = \frac{c}{1-r}$.

Figure 12: Summation formulae

And now we present **countability**:

Definition 4. (Countabiltiy) A set is countable iff. it is finite or there is a bijection from N to the set.

And some countability facts:

- Union of countable sets is countable
- Z is countable, just map each odd number to the negative integers and each even number to positive integers.
- Union of a finite number of countable sets is countable. That is for the finite A_k k many sets, first map all the $a_0 \in A_0$ to 1, then $a_1 \in A_1$ to 2 and hence $k+1 \to a_1 \in A_0$
- Union of a countable number of countable sets is countable.

$$
\begin{array}{llllllll}
S_0: & s_{0,0}, & s_{0,1}, & s_{0,2}, & s_{0,3}, & \ldots, & s_{0,j}, & \cdots \\
S_1: & s_{1,0}, & s_{1,1}, & s_{1,2}, & s_{1,3}, & \ldots, & s_{1,j}, & \cdots \\
S_2: & s_{2,0}, & s_{2,1}, & s_{2,2}, & s_{2,3}, & \ldots, & s_{2,j}, & \cdots \\
\vdots & \vdots & & & & & \vdots & \\
S_i: & s_{i,0}, & s_{i,1}, & s_{i,2}, & s_{i,3}, & \ldots, & s_{i,j}, & \cdots \\
\vdots & \vdots & & & & & \vdots &
\end{array}
$$

Figure 13: Enumeration

- Cartesian product of two countable sets is countable (for which reason rational numbers are countable)
- Infinite union of bitsrings of finite length are countable. That is suppose we define $\bigcup_{i=0}^{j} S^j$ where $S = \{0,1\}$ Now S^0 has 1 element, S^1 has 1 distinct element, S^2 has 3 distinct elements and so on. Thus, we map $0 \to S^0$, then $1 \to 01$ $2 \to 10$ $3 \to 11$. Since each S^k has 2^k elements our mapping is defined.
- The set $\{0,1\}^*$ consisting of infinite bitstrings is **uncountable** as follows by Cantor diagonilization.

A question from this week's pset asked:

Question ? 2.1.1. Let A be the set of real numbers with a finite number of 1 in the binary form and B the set of all reals with a finite number of 1 in decimal form. Which of these sets is countable?

Answer 2.1.1. As it turns out, when we try enumerating members of A, we notice this is doable because for instance if we were to miss out enumerating one element it would have to be the case that it either contains a 0 which we have already counted or a 1 which again we have already counted. Conversely for B, suppose we have enumerated all such numbers, but then define a new number with a digit differing in at least one decimal place of each element of B.

2.2 Lec 11. notes

We start with the **Halting problem** which states that given a computer program P with input I can we write a program that always determines whether $P(I)$ will terminate or not? As it turns out the answer is no. Now let us define a program $A(P,x)$ that determines whether the program P with input x terminates. Now define $Q(x)$ as a program calling $A(x,x)$. Then $Q(P)$ runs $A(P,P)$ and $Q(P)$ will stop if $A(P,P)$ returns. Now consider $Q(Q)$. We have that $A(Q,Q)$ gets called. Well, if this returns no meaning $Q(Q)$ does not terminate, we get that $Q(Q)$ should terminate and vice versa. Hence, we get that such a program $A(x)$ can not exist.

Definition 5. (Big O) A function $f(x)$ is said to be $O(g(x))$ or informally f is $O(g)$ if

$$\exists C, k\ \forall x > k,\ |f(x)| \leq C|g(x)|$$

Definition 6. (Big Ω) A function $f(x)$ is said to be $\Omega(g(x))$ or informally f is $O(g)$ if

$$\exists C, k\ \forall x > k,\ |f(x)|C|g(x)|$$

Definition 7. (Big Θ) if f is both $O(g)$ and $\Omega(g)$, then f is $\Theta(g)$ meaning that f and g are of the same order

2.3 Lec 12. notes

2.3.1 Basics

Notion of Random access We want our programs to have fast access to data. Thus an array is said to be **random access** if accessing or reading an element is independent of its index. Yet in reality this is far from true as there is only so much the **cache** can store.
When it comes to the cache, cache implements a compromise between spatial and temporal locality. That is, spatial locality refers to the likelihood that referencing a resource is high if something near it was referenced. Temporal locality refers to the notion that once a variable is called it will likely be called again.
An example is that Fortran caches 2-D arrays in column format. Hence to exploit spatial locality, when multiplying AXB we would need to transpose A and define matrix multiplication purely in terms of column to column inner products.

Binary search When performing binary search on an ordered list, we let $m := \lfloor \frac{n}{2} \rfloor$ for a list of n items. Now whenever $l \neq l_m$ and our saught after element $l < l_m$ we reduce array size to range $l_1, ..., l_{m-1}$ and else to $l_{m+1}, ..., l_n$. Repeating the previous step, we obtain the in the worst case, we perform the floor operation until we reach the largest k such that $\frac{n}{2^k} < 1$ to give us that k which is how many steps we take is bounded by $\log_2 n$ as $k < log_2 n$

2.3.2 Sorting algorithms

(Overview of algorithms)

Bucket sort Simply arrange similar elements into buckets and the apply any sorting algorithm.
Bubble sort

for(i = 1 to n-1){ if(l_i > l_{i+1}){.swap(l_i, l_{i+1}}}

The worst case is $\sum_{i=1}^{n}(n-i)$ swaps because for the largest element we have $(n-1)$ comparisons and swaps, for the second largest $(n-2)$ and so on which means that in the worst of cases we perform $\sum_{i=1}^{n}(n-i)$ many operations which is $O(n^2)$. Now in the best case we have simply $n-1$ comparisons which is $\Omega(n)$
Selection sort To order in increasing order do: Locate smallest element in unsorted list and replace with first element in unsorted list.
Insertion sort Inserts l_2 in the right spot in the already sorted list $\{l_1\}$ and so on. Now at the k^{th} step we have $\log_2(k)$ many comparisons using binary search. Hence total operations are bounded by $\sum_{k=1}^{n-1}(\log_2(k)$ comparisons. Insertion sort is $\Theta(n\log(n))$ if generally we use linked lists which prevent shifting problem present with arrays.
Quick sort Take a pivot l_i then create sublists S_1 of all smaller items and S_2 of all larger items. Then sort S_1 and S_2 using same method
Merge sort Treat each single element as a sorted list. Then take sublists of two elements and sort. Then merge each sublist making sure the ordering. Each new sublist of 4 elements is merged again and so on. We have that there are in total $\log_2(n)$ sublists with at most n comparisons hence $\Theta(n\log_2(n))$
Heap sort Create a tree using the initial order elements are given in. Then transform tree into a heap(calling heapify) by replacing each heap-violating node with child-element. Once heap is obtained, swap the largest parent with lowest child. Add largest parent to end of sorted list. Repeat process until tree contains only one element. Heap sort is $\Theta(n\log_2(n))$

2.4 Lec 13. notes

2.4.1 Some problems

We begin with an interesting theorem. Given that g_1 and g_2 for $g_i : \mathbb{N} \to \mathbb{R}_+^*$ and $f : \mathbb{N} \to \mathbb{R}_+^*$ are $\Theta(f)$ we want to show that $g_1 + g_2$ is $\Theta(f)$ Now we know that:

$$\exists k_i, C_{i,n}\ \forall x > k_i\ C_{1,1}|f(x)| \leq |g_1(x)| \leq C_{1,2}|f(x)|$$

and :

$$C_{2,1}|f(x)| \leq |g_2(x)| \leq C_{2,2}|f(x)|$$

Now seeing that $g_1(x) + g_2(x)$ is $O(f)$ is easy since $|g_1(x)| + |g_2(x)| \geq |g_1(x) + g_2(x)|$ which when we add our inequalities we obtain:

$$|g_1(x)| + |g_2(x)| \leq C_{1,2} + C_{2,2}(|f(x)|)$$

implying that $g_1(x) + g_2(x)$ is $O(f)$. For the $g_1(x) + g_2(x)$ is $\Omega(f)$ part, we know that our functions have the natural numbers as a domain which means that $|g_1)(x)| + |g_2(x)| = g_1(x) + g_2(x)$ and since $C_{1,1} + C_{2,1}(|f(x)|) \leq |g_1)(x)| + |g_2(x)|$ we have that $g_1(x) + g_2(x)$ is $\Omega(f)$. Yet this proof also shows that in general $g_1 + g_2$ **is not** $\Theta(f)$. For instance when the $g_i : \mathbb{N} \to \mathbb{R}$ it is not always true that $g_1 + g_2$ is not $\Theta(f)$.

Example 2. Let $g_1(x) = -g_2(x)$. Then we have that $g_1 + g_2 = 0 < c|f(x)|$ which as a consequence shows that $g_1 + g_2$ is not $\Theta(f)$

Example 3. We now consider how to pin down the complexity of a given program. Suppose we are given:

```
for i=1 to n
    for j=1 to ⌊n/2⌋
        x = x + 1;
```

Now looking at the inner loop we see that it yields $\sum_{j=1}^{\frac{n}{2}} 1$ steps at most. And the two loops combined therefore yield

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

which is equal to:

$$\sum_{i=1}^{n} (\frac{i}{2}) = \frac{1}{4}n^2 + n$$

2.4.2 More complexity stuff

An equivalent definition to Big-O is in terms of limsup as:

$$f \text{is } O(g) \iff \limsup_{x \to \infty} \frac{|f(x)|}{|g(x)|} < \infty$$

Which simply means that after some arbitrarily large x, f must increase at a lower rate than $g(x)$. Thus this definition may be used in showing that the witnesses exist.
And yet another complexity definition is:

Definition 8. f is $o(g)$ (read f is little-o) whenever:

$$\forall C > 0,\ \exists k\ \forall x > k,\ |f(x)| < C|g(x)|$$

Or equivalently:

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0$$

We note that f is $o(g)$ implies that f is $O(g)$ as we may simply use universal instantiation to see this.

Fixed powers For all positive constants d, l we have that if $d \geq l$ then n^l is $O(n^d)$ since $\limsup \frac{n^l}{n^d} < 1$.

Polynomials For some polynomial $f(x)$ of degree n such that:

$$f(x) = \sum_{i=0}^{n} a_i x^i$$

we have that $fisO(x^i)$ This may be seen since $f(x) \leq a_{max}x^0 + \ldots + a_{max}x^n \leq a_{max}(d+1)x^n (\forall x > 1)$

Exponentials Now for some exponential $S(n)$ such that:

$$S(n) = \sum_{i=0}^{n} a_i i^d$$

We have that each term is at most as big as $a_{max}n^d$ and with n many such terms we have that $S(n) \leq na_{max}n^d = a_{max}n^{d+1}$ hence S is $O(n^{d+1})$.

Powers of logarithms (Note that this argument works for logarithms of any base). Suppose $f(x) = (\log(x)^t$ where t is a constant. Then f is $O(n^\epsilon)$ where ϵ is any real number larger than 0.
More on logarithms For constant σ it is the case that $\log(n^\sigma)$ is $O(\log(n))$ and this is never the case without logarithms!
We also have that for constant Γ and σ and whenever $a > b > 1$ we have that $\log_b(x^\sigma)$ is $O(\log_a x^\Gamma)$ and not the other way around. We also have that for any $a, \Gamma > 1$ $\log_a(x^\Gamma$ is $O(\log(X)$ which may be seen via change of base.

(Properties and non-properties of big-O)

Properties and non-properties of big-O Let f_i and g_i be functions such that f_i is $O(g_i)$ for $i = 1, 2$. It is often assumed that if some standard operation (such as addition (where $(f_1+f_2)(x) = f_1(x) + f_2(x)$), multiplication (where $(f_1f_2)(x) = f_1(x)f_2(x)$), powering (where $2^{f_i}(x) = 2^{f_i(x)}$), logarithm (where $(\log(f_i))(x) = \log(f_i(x))$) is applied to the f_i-functions, that then the resulting composed function is again the big-O of the same composition of the g_i-functions.

Addition (wrong) In general it is not the case that the function $f_1 + f_2$ is $O(g_1 + g_2)$: take for instance $g_2 = -g_1$. It is not hard to prove, however, that $(f_1+f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|)$ (use the triangle inequality $|x + y| \leq |x| + |y|$ – can you prove the triangle inequality?).

Multiplication (exceptionally correct) It is true (and it is not hard to prove) that the function f_1f_2 is $O(g_1g_2)$.

Powering (wrong) In general it is not the case that 2^{f_1} is $O(2^{g_1})$: take for instance $f_1(x) = 2x$ and $g_1(x) = x$, then f_1 is $O(g_1)$ (witness $C = 2$), but

$$\frac{2^{f_1}(x)}{2^{g_1}(x)} = \frac{2^{f_1(x)}}{2^{g_1(x)}} = \frac{2^{2x}}{2^x} = 2^x$$

and 2^x can obviously not be bounded by any constant (for x going to infinity) as would be required by 2^{f_1} being $O(2^{g_1})$ (remember the equivalence of "f is $O(g)$" and $\limsup_{x\to\infty} \frac{|f(x)|}{|g(x)|} < \infty$: because $\limsup_{x\to\infty} \frac{|2^{f_1}(x)|}{|2^{g_1}(x)|} = \infty$ it is not the case that 2^{f_1} is $O(2^{g_1})$). Note that an incorrect answer is given to exercise 29 of Section 3.2 of the seventh edition of the book. In the eighth edition it is exercise 42 in Section 3.2.

Logarithm (wrong) In general $\log(f_1)$ is not $O(\log(g_1))$. All we know about f_1 and g_1 is that there is a positive constant C (say $C > 2$) such that for all large enough x it is the case that $|f_1(x)| \leq C|g_1(x)|$. It follows that $\log(|f_1(x)|) \leq \log(C)+\log(|g_1(x)|)$, but that does not imply $|\log(f_1(x))| < \tilde{C}|\log(g_1(x))|$ for some constant $\tilde{C}$ and large enough x, as required for $\log(f_1)$ being $O(\log(g_1))$. If f_1 and g_1 are increasing and unbounded it follows that $f_1(x) > 1$ and $g_1(x) > 1$ for all $x > y$ for some y, so that logarithm-accidents involving huge negative values are avoided and we have $\log(f_1(x)) \leq \log(C) + \log(g_1(x)) = \log(C)\frac{\log(g_1(x))}{\log(g_1(x))} + \log(g_1(x))$ and thus, using again that g_1 is increasing, that $\log(f_1(x)) \leq \log(C)\frac{\log(g_1(x))}{\log(g_1(y))} + \log(g_1(x)) \leq \tilde{C}\log(g_1(x))$, where $\tilde{C} = \frac{\log(C)}{\log(g_1(y))} + 1$.

Little-o properties Assume that f is $o(g)$. Not only does this not imply 2^f is $o(2^g)$ (example: take $f(x) = \frac{1}{x^2}$ and $g(x) = \frac{1}{x}$) but neither does it imply that $\log(f)$ is $o(\log(g))$ (example: take $f(x) = x$, $g(x) = x^2$). Addition and multiplication properties are maintained though.

Symmetries and lack thereof It is obvious that f is $O(f)$ (but, as seen above, f is not $o(f)$). Generally speaking "f is not $O(g)$" does not imply "g is $O(f)$": indeed, functions f and g can be constructed such that it is not the case that f is $O(g)$ and it is not the case that g is $O(f)$. A dull example is $f(x) = \sin(x)$ and $g(x) = \cos(x)$; more interesting examples would be where both f and g are strictly increasing functions (cf. this week's exercises).

Note that if f is $O(g)$ it may be the case that g is $O(f)$ as well or it may be the case that g is not $O(f)$. The negation of f is $O(g)$, i.e., "it is not the case that f is $O(g)$" or equivalently "f is not $O(g)$", immediately follows from the above definition of big-O:

$$\text{“}f \text{ is not } O(g)\text{” is equivalent to:} \qquad \forall\, C, k \;\; \exists x > k \;\; |f(x)| > C|g(x)|.$$

Thus, an "occasional x" (i.e., **not necessarily all** x) that breaks the "$\leq$" as required by big-O suffices to negate f is $O(g)$. It follows that to prove that f is not $O(g)$ it suffices to find, for any constants $C > 0$ and k, some $x > k$ (not necessarily all $x > k$) for which $|f(x)/g(x)| > C$.

The big-O hierarchy You should be familiar with the following underlined (and strictly increasing) *hierarchy* of big-Os (and with the proofs: look at the ratios and consider the behavior for n going to infinity, cf. lim sup-equivalence), where $b > 1$, ϵ with $0 < \epsilon < 1$, $k > 0$, and $d > 1$ are fixed constants:

- any constant is $O(1)$
- 1 is $O(\log(\log(n)))$ but $\log(\log(n))$ is not $O(1)$
- $\log(\log(n))$ is $O(\log(n))$ but $\log(n)$ is not $O(\log(\log(n)))$
- $(\log(n))^k$ is $O(n^\epsilon)$ but n^ϵ is not $O((\log(n))^k)$
- n^ϵ is $O(n)$ but n is not $O(n^\epsilon)$
- n is $O(n\log(\log(n)))$ but $n\log(\log(n))$ is not $O(n)$
- $n\log(\log(n))$ is $O(n\log(n))$ but $n\log(n)$ is not $O(n\log(\log(n)))$
- $n(\log(n))^k$ is $O(n^d)$ but n^d is not $O(n(\log(n))^k)$
- n^d is $O(b^n)$ but b^n is not $O(n^d)$
- b^n is $O(n!)$ but $n!$ is not $O(b^n)$
- $n!$ is $O(n^n)$ but n^n is not $O(n!)$, whereas n^n **is** $O((n!)^2)$
- $\log(n!)$ is $O(\log(n^n))$ **and** $\log(n^n)$ is $O(\log(n!))$

To prove the statements about n^d versus b^n use $n^d = b^{d\log_b(n)}$ and observe that the latter's logarithmic in n exponent $d\log_b(n)$ grows much slower than the exponent n of b^n.

From $n!$ is $O(n^n)$ (and the earlier discussion on logarithms) it follows that $\log(n!)$ is $O(\log(n^n)) = O(n\log(n))$. Conversely, from the fact that n^n is $O((n!)^2)$ (this follows from $n^n \leq (n!)^2$, which is easily seen to be the case[1]) it follows that $\log(n^n) = n\log(n)$ is $O(\log((n!)^2)) = O(\log(n!))$. We conclude that $n\log(n)$ is $\Theta(\log(n!))$. Here everything in purple is called *polynomial time* — traditionally "good" — and blue is *exponential time* or worse — traditionally "bad".

Computational complexity As we have seen during the lectures, the big-O notation is a succinct way to catch the essence of the number of operations to be carried out by an algorithm to solve a certain problem. Here it is assumed that a "problem" is interpreted as an infinite set of problem instances, where each problem instance has a certain well defined size. For instance: "sorting integers" is a problem, and any integer sequence of length n may be regarded as an instance of length n of the sorting integers problem; "integer addition" and "integer multiplication" are problems and any pair of n-bit integers may be regarded as an instance of length n of the integer addition or of the integer multiplication problem. Any instance of length n of the sorting problem can be solved in time $O(n\log(n))$, any instance of length n of the integer addition problem can be solved in time $O(n)$, any instance of length n of the integer multiplication problem can be solved in time $O(n^2)$ using schoolbook multiplication, in time $O(n^{\log_2(3)})$

[1] Because $(n!)^2 = \prod_{i=1}^{n} i(n-i+1)$, to prove that $n^n \leq (n!)^2$ it suffices to prove that $n \leq i(n-i+1)$ for $1 \leq i \leq n$. This follows from the fact that the parabola $-i^2 + i(n+1) - n$ is zero for $i = 1, n$ and positive on the interval $1 < i < n$.

using Karatsuba multiplication. Since earlier this year it is known that it can be done in time $O(n\log(n))$ (Harley and van der Hoeven, cf. September 27 lecture notes). No one knows if multiplication can be done faster.

We refer to the expressions in the big-O's as the "complexity" (or "time complexity", or "computational complexity") of the problem at hand, or we say that the problem belongs to the "complexity class" $O(\ldots)$ for whatever ... is applicable: "the complexity of sorting is $n\log(n)$", "the complexity of addition is linear, but no one knows yet what the complexity is of multiplication"[2]. Below a few common complexity classes are listed (where the Os may be replaced by Θs, except it is more common to use Os), along with common problems that belong to that complexity class:

- $O(1)$ ("constant") to retrieve the largest item in a sorted list of any size, or to compute the parity of a number given in its binary representation;
- $O(\log(n))$ ("logarithmic") for binary search in a sorted list of n items (if the list allows random access, for whatever that is worth);
- $O(n)$ ("linear") for linear search in a list of n items that is not necessarily sorted (requiring only sequential access);
- $O(n\log(n))$ ("n log n") to sort a list of n items using merge sort or heapsort[3] but
- $O(n^2)$ ("quadratic") to do it using bubble sort or in the worst case of quick sort;
- $O(n^3)$ ("cubic") to multiply two $n \times n$ matrices (note that the input length in that case is n^2, so as a function of the input length it is "only" a degree 1.5 algorithm and calling it cubic is a bit misleading – but traditional) and, a rather extreme example:
- $O(n^{12})$ ("exponent twelve") for the original 2002 AKS primality-proving method ("the three Indians method" – Indian indians, not American indians) to establish if an n-bit integer is prime or composite (this has in the meantime been improved to $O(n^6)$ ("exponent six")). This should be contrasted with the "practical" method that does not give absolute certainty but that, for randomly generated numbers of hundreds of bits, has not failed yet and that runs in time $O(n^3)$ if schoolbook multiplication is uses (or $O(n^2\log(n))$ with the latest $O(n\log(n))$ multiplication method).

The class P of polynomial time solvable problems Above we saw examples of problems that can be solved in time $O(n^d)$, i.e., for which there exists an algorithm and a fixed constant d such that for any problem instance of size n (i.e., the length of the input to the algorithm[4]), the algorithm will produce a solution in a number of operations that is $O(n^d)$. In the early 1970s it became customary to use the term *polynomial time* for such algorithms. If a problem can be solved by a polynomial time algorithm, the problem is polynomial time solvable (has *polynomial complexity*), and the problem is said to belong to the class **P** of polynomial time solvable problems: a problem X can be solved in polynomial time if there is an algorithm A and a fixed constant d such that for all n any instance of size n of problem X can be solved by A in $O(n^d)$ operations:

$$\exists A\, \exists d\, \forall n\, \forall \text{ instances } I_n \text{ of size } n \text{ of } X,\ A \text{ solves } I_n \text{ in time } O(n^d).$$

Note the importance of the order of the quantified variables A, d, n and I_n: reversing the order of the existentially quantified d and the universally quantified n and I_n turns it into a useless property (it would always hold because it would become possible to choose d depending on n or I_n).

The existence of a polynomial time algorithm for a problem does not imply that each algorithm to solve the problem runs in polynomial time: there are always plenty of ways to do things in a less clever manner

[2]Observe that each $\delta, k \in \mathbf{R}_{>0}$ gives rise to unique complexity classes $O(n^\delta)$, $O((\log(n))^k)$, and $O(n^\delta(\log(n))^k)$ and thus that there are uncountably many different complexity classes.

[3]If we can only make "binary decisions" such as comparing items, we need at least $\log(n!)$ steps to be able to distinguish between all $n!$ possible distinct outcomes of the problem of sorting a list of n items. Because $\log(n!)$ is of the same order as $n\log(n)$ sorting cannot be done faster than order $n\log(n)$.

[4]In our input length estimates we disregard the sizes of the numbers involved. For the problems and algorithms considered in this course that will be adequate. For more complex problems a more precise approach must be used.

that may take many more operations. But, once a polynomial time algorithm for a problem has been found, the problem is, more or less and at least from a theoretical point of view, considered to be *solved*, because it means that the number of operations required to find a solution can be nicely controlled by a "decent" polynomial function: of course solutions require more operations when larger problem instances are solved, but the growth of the number of operations is limited by a polynomial function and therefore thought to be reasonably well under control.

Everything that is purple in the "big-O hierarchy" above is polynomial time, assuming that the input length is n. Problems in the class **P** are traditionally referred to as *tractable* problems. But notice that the fixed exponent d in the expression that upper bounds the number of operations (up to a constant) may be any nonnegative constant: one may wonder how "tractable" a polynomial time solution is for which $d = 12$.

These nicely behaving polynomial time solvable problems are very much unlike nasty problems for which polynomial time solutions are not known to exist and for which the best algorithms known require at least *exponential time*: the blue bulleted expressions in the "big-O hierarchy" represent exponential time and worse (see below) run times, again assuming that the input length is n.

Problems not known to be in P A simple example of a problem for which no polynomial time solution is known to exist is the knapsack problem: given a set S of n items, each item in S with a value and a weight, the "best" subset of items in S must be found that satisfies a maximal weight restriction:

> let $S = \{s_1, s_2, \ldots, s_n\}$ be some set of items, let $v, w : S \to \mathbf{R}_{\geq 0}$ be value (v) and weight (w) functions, and let $W \in \mathbf{R}_{\geq 0}$ be a maximal weight restriction. Find a subset $T \subseteq S$ such that
>
> $$\left(\sum_{t\in T} v(t)\right) \text{ is maximized, while } \left(\sum_{t\in T} w(t)\right) \leq W$$

A solution would be to try all 2^n possible subsets T of S (while verifying the maximal weight W restriction) and selecting the one of highest value, but 2^n is exponential time. It is often more convenient to replace the above *optimization problem* by the — mostly equivalent[5] — *decision problem* of deciding if for some specified value $V \in \mathbf{R}_{\geq 0}$ there exists a subset $T \subseteq S$ such that $\left(\sum_{t\in T} v(t)\right) \geq V$ and $\left(\sum_{t\in T} w(t)\right) \leq W$ (and, if so, to provide such a subset T). Note that it can easily (i.e., in polynomial time, even in linear time) be checked if any proposed solution (i.e., some subset T) to the latter decision problem is correct (i.e., satisfies the two conditions). How to efficiently construct such a solution, however, is unclear. In particular it is not known if exponential time is required.

Another example is the traveling salesman problem, where a bounded length tour must be found among n cities: again a proposed solution can easily be checked, but finding one still escapes us, despite decades of intensive research. Note that a solution can be found in $O(n!)$ operations by trying all possible tours. Finding the chromatic index of a graph (the smallest number of colors required to color all vertices in the graph in a such a way that two neighbouring vertices have different colors) is – for the moment at least – not doable in polynomial time either; but checking if a given coloring satisfies a given bound is easily done (i.e., in polynomial time). These examples should be contrasted with problem such as "minimal spanning tree" or "shortest path" which are both easily solvable in polynomial time: minimal spanning tree even by a greedy algorithm that consistently selects a new edge of lowest weight without closing a cycle.

The class NP, and P versus NP Occasionally a polynomial time solution is found for one of the nasty problems, but after a promising start in the 1970s this soon happened less and less frequently (once the low-hanging fruit was gone) and occurs only very rarely these days (this should not be interpreted as a reason to stop looking!). The separation between the polynomial time solvable problems and those for which a polynomial time solution is not known to exist (yet?) persists to the present day. A particular question that is found to be of interest (and that will earn you a one million US$ prize if you solve it) is the **P** versus **NP** problem, namely if **P** is properly contained in **NP** or if **P** and **NP** are the same. But, what is **NP**? Probably not what you think it is, as explained in the next paragraph.

[5]Because the optimization problem can be solved by solving polynomially many decision problems (using binary search on potential V-values). The same comment applies to other optimization problems (such as traveling salesman or chromatic index).

The class **NP** consists of those problems for which the correctness of a proposed solution can be verified in polynomial time: for instance, the decision versions of the knapsack and traveling salesman problems belong to **NP**. Note that **NP does not stand for "not-P"**: it refers to the class of algorithms that can be solved in *nondeterministic polynomial time*. Here "nondeterminism" refers to the unspecified way in which a proposed solution is found: it may have been found by guesswork or any other way one sees fit (for instance, a "nondeterministic" one).

The difference between **P** and **NP** is the same as the difference between **finding** a solution to a problem (namely in polynomial time) and **verifying** (in polynomial time) that a given solution is correct – without worrying at all in the latter case how the solution was found. It is clear that all problems that belong to **P** also belong to **NP**. It has not been proved yet that there are problems that belong to **NP** but for which no polynomial time solution can exist (which would prove that $\mathbf{P} \neq \mathbf{NP}$ and thus that **P** is properly contained in **NP**) and neither has it been proved that all problems in **NP** allow a polynomial time solution (which would prove that $\mathbf{P} = \mathbf{NP}$). Some argue that based on our failure to prove that $\mathbf{P} = \mathbf{NP}$, despite the aforementioned decades of indeed intensive research, $\mathbf{P} \neq \mathbf{NP}$ is a safer bet than $\mathbf{P} = \mathbf{NP}$.

NP-complete problems The above statement "neither has it been proved that all problems in **NP** allow a polynomial time solution" sounds like a hopeless exercise, but this is what makes this field interesting: it has been shown that there are problems in **NP**, the so-called *NP-complete problems*, that have the following intriguing property: if a polynomial time solution is constructed for just a single NP-complete problem (not for a single problem instance, of course...), then all problems in **NP** are polynomial time solvable. The first problem shown[6] to be NP-complete was the *satisfiability problem* (finding an assignment of values to logical values such that a conjunction of disjunctions of the logical values (or their negations) evaluates to true), by showing that any problem X in **NP** can be *reduced in polynomial time* to the satisfiability problem (this means that solving X can be done in polynomial time, assuming the operations spent on solving one or more (but at most polynomially many) satisfiability problems are not counted). Other examples are the (decision versions of) the knapsack problem and the traveling **salesman problem.**

2.5 Lec 14. notes

2.5.1 Mathematical induction

Let's start by proving that mathematical induction is a valid proof technique. In our proof we use the well-ordering principle which states that:

Every non-empty subset of the natural numbers contains a least element.

Let's now begin our proof:

Proof. Suppose that we have the statement $P(1)$ true and also that $P(k) \to P(k+1) \forall k \in \mathbb{N}$. Now also suppose that $\exists n \in \mathbb{N}$ for which $P(n)$ is false. Let now $S = \{\text{all indexes for which P(n) is false}\}$. Now by the well-ordering property we have that S is non-empty. Let m be the least element of S. Now m can not be 1 since since $P(1)$ is true. Now notice that because m is a positive integer greater than 1, we have that $m-1$ must be a positive integer that is not in S. Hence we have that $P(m-1)$ is true. Now by our induction hypothesis(which is also true) we have that $P(m)$ must be true. Hence we have that $P(n)$ is true for all n. □

2.6 Lec. 15 notes

2.6.1 Worse than exponential time

We note that $n!$ is worse than exponential time b^d. And now let's derive an expression for subexponential time complexity. That is we need a function that lies between n^d and b^n where d, b are constants. We write n^d as:

$$n^d = b^{(\log_b n)^d}$$

From this we note that the expression:

$$b^{d \cdot n^r (\log_b n)^{1-r}}$$

[6] Independently by two researchers in the very early 1970s: Stephen Cook and Leonard Levin.

denotes subexponential time. The closer r is to 1, the closer we are to exponential time.
Another function that lies between polynomial and exponential time is quasi-polynomial time. Consider the expression:

$$b^{(\log_b(n))^c}$$

. For instance when $c = 1$ we have $n^{log_b n}$ which is worse than polynomial time as the exponent grows and is unbounded.

2.6.2 Some exemplary proofs

Example 4. We claim that any natural number greater than or equal to 8 can be represented as a combination of 3's and 5's.

Proof. We will use strong induction here(which we remind is equivalent to induction). Strong induction is the tautology that:

$$P(0) \wedge \forall k[\forall l \leq k \ P(l) \rightarrow P(k+1)] \rightarrow \forall k P(k)$$

Now our base case P(8) is clear. Let now $P(k-2) \wedge P(k-1) \wedge P(k)$ be our inductive hypothesis. We need to show $P(k+1)$. Well $P(k+1)$ clearly holds because given that $P(k-2)$ if we simply add 3 to the combination present in $P(k-2)$ we are done. Therefore we have shown that $P(k-2) \wedge P(k-1) \wedge P(k) \rightarrow P(k+1)$ is a tautology. And since we have that $P(8) \wedge P(9) \wedge P(10)$ as our basis step we are done. □

2.7 Lec 16. Notes

Consider the statement that everyone is equal. Let's tackle this with weak induction.

Example 5. Now our statement says $P(n)\ \forall n \geq 1$ As a basis step we take $P(1)$ which is trivially true. Now assume $\forall k\ P(k)$ Then we get that $P(k+1)$ also holds because persons through $2-k+1$ are the same(by IH) and persons through $1-k$. Yet, the problem lies in our basis. That is 1 person being the same does not imply that any two other persons are the same.

Example 6. Suppose we claim that $p^2+p+1|p^{n+1}+p^{2n-1}\ \forall n \geq 1$ The basis $P(1)$ is trivial. Now assume $\forall k P(k)$ Then we have that $P(k+1)$ is:

$$p^2 + p + 1|p^{n+2} + p^{2n+1}$$

Now we try to rewrite $p^{n+2} + p^{2n+1}$ as: $p(p^{n+1} + p^{2n-1}) + x$ where $x = (p^2 + p + 1)(p + 1)^{2k-1}$ hence we are done.

Example 7. Consider the claim that every $2 \leq n \in \mathbb{Z}$ is divisible by a prime number. Basis is trivial. Now assume that $\forall l\ 2 \leq l \leq kP(l)$ Given this, for $P(k+1)$ we have that either $k+1$ is prime or that $k+1 = a \cdot b$ where both $a, b \leq k$ hence IH applies to both and we are done.

And now we come to a more appealing example.

Example 8. We claim that a chocolate bar of m single blocks can be broken into m single blocks in $m-1$ breaks. Let's use strong induction again. Now base case $P(1)$ is trivial. Let now our IH be:

$$\forall l,\ 1 \leq l \leq k\ P(l)$$

Now given this consider we have a bar of $m-1$ blocks. Then suppose we break it into m and $k+1-m$ pieces at the cost of 1 break. Now our IH applies to both the blocks since their size is less than m. Through this we have that m will take $m-1$ and $k+1-m$ will take $k+1-m-1$ steps which in total gives $m-1+k+1-m-1+1 = k$ steps and we are done.

Let's now expand on example 8. Suppose that instead of defining one single break as a cost of 1, the cost of breaking m blocks into $m-L$ and L blocks is defined as $L(m-L)$. Now our goal is to find the optimal cost and prove it. As an example if we break n pieces into $n-L$ and L many pieces than net cost is $L(n-L)$. Notice that we may define the total cost $S(n)$ of breaking n pieces into n single units recursively as:

$$S(n) = \min_{1 \leq k \leq n} (k(n-k) + S(n-k) + S(k))$$

Now we conjecture that $S(n) = \frac{n(n-1)}{2}$ $\forall n \geq 1$

Proof. Basis is trivial. As for strong IH, we have that

$$\forall 1 \leq l \leq n,\ S(l) = \frac{l(l-1)}{2}$$

We now have to show that $S(k+1) = \frac{k(k+1)}{2}$ Now by definition we have that:

$$S(k+1) = \min_{1 \leq l \leq k+1} (l(k+1-L) + S(k+1-l) + S(l))$$

By our hypothesis and simplifications we get that $S(k+1) = \frac{(k+1)k}{2}$ as intended. □

(Useful analogy)

An intuition for our above result is the handshaking example. Suppose that a group of n people split up into a group of $n-k$ and k people. Hence at every splitting there are $(n-k)k$ handshakes. And at the end there are obviously $\underbrace{n-1+n-2+\ldots+1}_{\text{each person is a node}}$ many handshakes which is equal to $\frac{n(n-1)}{2}$(by summation for arithmetic series)

2.8 Lec 17. notes

We come to the subject of recursion. Recursion is similar to induction in the way that it exploits a base step called "bottom of recursion". Let's first see examples of where recursion is hidden.

Example 9. We may define summation $\sum_i^u x_i$ recursively as:

$$\sum_i^{u-1} x_i + x_u \text{ if } i \leq u \text{ and if } i > u \sum_i^u x_i = 0$$

We may do the same for multiplication, that is:

Example 10. We define $\prod_i^u x_i$:

$$x_u \cdot \prod_i^{u-1} \text{ if } i \leq u \text{ and else} \prod_i^u = 1$$

And notice that the above may be implemented programmatically:

```
sum(i,u,x_i) = if l>u then 0 else sum(i,u-1,x_i) + x_u
```

Similarly the factorial function is defined recursively as:

$$n! = n \cdot (n-1),\ 0! = 1$$

Now recursion becomes very useful for fast exponentiation. Observing the following:

$$a^{171} = (a^{85})^2 \cdot a = ((a^{42})^2 \cdot a)^2 \cdot a = \dots$$

We define p_l the exponentiation function as:

$$p_l(a, l) = \text{if } n \leq 0 \text{ then} 1 \text{ else } a^{n1} \cdot p_l(a, \lfloor \frac{n}{2} \rfloor)^2$$

where $a^{n1} = \{0, 1\}$ and is equal to 1 iff n is even.

3 Counting (Week 9 -

3.1 Lec 18. notes

Consider applications of counting principles. Suppose we create a password using characters and digits. In total we have 70 possibilities for each entry which follows from the **addition rule** as we may have upper case or lower case or digits or specials hence $26 + 26 + 10 + 8$ And now consider how many possible passwords we have of 12 entries. Using the product rule, we obtain 70^{12} And some more examples:

1. Not containing a digit is 60^{12}
2. Begin with a digit is $10 \cdot 70^{11}$
3. Begin with a digit and end with a character is $10 \cdot 26 \cdot 70^{10}$
4. Begin with a digit or special character is $(10 + 8) \cdot 70^{11}$

The last example follows from $|A \cup B| = |A| + |B| - |A \cap B|$ with $|A \cap B| = 0$ And now a slightly more complicated example. Consider the set of passwords that begin with a digit or end with a special character. Then $|A| = 10 \cdot 70^{11}$ and $|B| = 8 \cdot 70^{11}$ hence this gives $|A \cup B| = 10 \cdot 70^{11} + 8 \cdot 70^{11} - 10 \cdot 8 \cdot 70^{10}$ Also note that for $|A| \cup |B| \cup |C|$ we have that it equals: $|A| + |B| + |C| - |B \cup C| - |A \cup C| - |A \cup C| + |A \cap B \cap C|$

Complementarity: If we asked the number of passwords consisting of at least one digit, we could simply figure this using total number of passwords - number of passwords with no digits. More formally this is: $|A \cap B| = |A| - |A \cap B^*|$

Consider now all possible mappings $A \to B$ where $|A| = r$ and $|B| = n$. How many such mappings are there? Well it follows that for each $x \in A$ we have n many choices which means multiplying n, r many times hence $|B|^{|A|}$ many possible maps.

And now consider how many possible injective mappings we have from A to B. Well for first element we have $|B|$, for second $|B - 1|$ many and for last element $|B| - |A| + 1$ many choices. This yields:

$$|B| + \dots + |B| - |A| + 1 = \frac{n!}{(n-r)!}$$

which also happens to be the permutation formula.

Having introduced the permutation formula, the slight difference in combination formula is that because order does not matter, we must rid our count of all occurrences of the same combination. If we are picking r objects out of n then we know that there are $\frac{n!}{(n-r)!}$ many choices and $r!$ many repetitions hence we divide by $r!$ to get the combination formula:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

And yet more detail follows when we consider all possibilities of permutations:

- permutation with replacement: password selection
- permutation without replacement: sports competition winners

- combination with replacement: choosing cookies from jar
- combination without replacement: selecting members of a committee

An application of the permutation is a password without repetition. We have that for password of length r there are $P(n, r)$ many options.

And now we state the pigeonhole principle.

Theorem 3.1. If $k+1$ many objects are to be placed in k many boxes, then at least one box contains 2 objects.

3.2 Summary of counting principles

(Counting principles)

Product rule: Break a task into two subtasks of cost n_1 and n_2 then total cost is $n_1 n_2$

Example 11. Labeling a name tag with one character and one digit less than 100, then in total we have $26 \cdot 100$ choices

And a more elaborate example using nested loops:

```
int k = 0;
    for i=1 to a{ //loop 1
        for i=1 to b{ //loop 2
            k += 1;
        }
    }
```

For each traversal of length n, n is added to k and since for example loop 2 is done a many times we have that $k = a \cdot b$

Division rule: If there are n total ways to do a procedure and w of these procedures correspond to the same task d then there are $\frac{n}{d}$ ways of doing the task.

3.3 Lec 19. notes

We remind the following:

- Permutation with replacement from n to k with $n \leq k$ is k^n
- Permutation without replacement is $P(n, k) = \frac{n!}{(n-k)!}$
- Combination without replacement is $C(n, k) = \frac{n!}{k!(n-k)!}$

And now some examples of poker handouts:

Four of a kind(4 of same kind and one other)
13 ways to select a kind and 48 left after selection hence 13 ways to select first and 48 for last hence in total $13 \cdot 48$ ways.

Full house(3 and 2 of samekind)
13 ways to select kind and $\binom{4}{3}$ to select suit and then again 12 ways to select kind and $\binom{4}{2}$ ways to select suit. Alternatively one might see this as follows; there are 52 ways to pick a first card which determines a kind. Then, the remaining 3 cards of the card are taken as the kind of three. With 48 cards remaining now, we have $\binom{3}{2}$ choices to make for the final card. Yet realizing that this situation is symetric we divide by 2 to obtain: $\frac{3}{2} \cdot 48 \cdot 52$

Another interesting problem is how many choices there are when traversing a grid of size $(m+1, n+1)$ where say we may only travel north or east. It turns out that this is the combination $\binom{n+m}{n}$ because with always $n+m$ steps to take, we are deciding n many times whether to turn north or east.

Now consider how many n bit string there are with precisely r many ones. This is equal to $\binom{n}{r}$ and from this one sees that $\sum_{r=0}^{n} = 2^k$ as the question we asked is equivalent how many possible subsets of an n cardinality sets there are.

And now **Pascal's Identity**:

Theorem 3.2.

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$$

Proof follows from simple(based on who you are) algebra

Theorem 3.3.

$$r\binom{n}{r} = n\binom{n-1}{r-1}$$

This theorem is obvious through the following combinatorial argument: we are selecting r many committee members and appoint one of them as a chair. This yields $r\binom{n}{r}$ such ways. Similarly we first pick the chair and then pick committee members from $n-1$ members yielding $n\binom{n-1}{r-1}$

We left out earlier **combinations with replacement** and now come back to them. As it turns out the general formula for selecting r items out of n is:

$$C((r+(n-1),(n-1)) = \binom{n+r-1}{n-1}$$

Example 12. (**Stars and bars**) A famous problem in elementary combinatorics is to ask how many distinct 3-tuples are there such that $x_1 + x_2 + x_3 = 7$. This is easily resolved using the *stars and bars analogy*. We model it as follows:

$$xxxxxxx||$$

each time we move the | slider to the right it adds a value to the corresponding x_i for instance $xxx|xx|xx$ is the case where $x_1 = 3, x_2 = 2, x_3 = 2$ Thus the total number of slider positions is $\binom{7+3-1}{2}$

Now further motivating this example we may ask how many distinct integer n tuples are there such that $x_1+x_2+x_3 \leq 7$ The smart way to do this is to add a dummy variable x_4 and let $x_1+x_2+x_3+x_4 = 7$ as the two are equivalent. Then the solution to the latter is $\binom{7+4-1}{3}$.

And taking this even further what if we ask how many distinct integer n tuples are there such that $x_1+x_2+x_3 \leq 18$ and also that $x_i \geq i$. Well we make the problem easier by letting $y_i = x_i - i$ This works because if say $x_2 = 2$ then $y_2 = 4$ hence an equivalent but much simpler problem is:

$$y_1 + 1 + y_2 + 2 + y_3 + 3 + y_4 = 18$$

which trivially solves to $\binom{12+4-1}{3}$

Example 13. (The hardest integral solutions problem)

Suppose we are asked $x_1 + x_2 + x_3 \leq 12$ where $x_i \geq i\ \forall x = 1, 2$ and $0 \leq x_3 \leq 3$ Now using our previous method we obtain: $y_1 + 1 + y_2 + 2 + \underbrace{y_3}_{y_3 = x_3} \leq 12 \equiv y_1 + y_2 + y_3 \leq 9$ Now we divide this into cases $y_1 + y_2 + y_3 \leq k$ where $k = 9, 8, 7$ and solve as shown before.

Counting combinations with replacement

We have a set of n distinct items, say $\{1, 2, 3, \ldots, n\}$, and we have to select r of them with replacement (i.e., items may be selected multiple times) but we are not interested in the order in which the items are selected. Thus, with $r = 6$, selecting $1, 1, 3, 2, 1, 2$ counts as the same possibility as selecting $1, 1, 1, 2, 2, 3$. Denoting by x_i the number of times that item i is selected, we thus have $\sum_{i=1}^{n} x_i = r$. Given n and r we want to find the number of solutions to that equation for non-negative integers x_i.

We solve this by fixing some $r > 0$ and considering how we can find the number of solutions for increasing values of n. For $n = 0$ we find that there is no way to select r items. For $n = 1$ there is just a single way to select r items: just take $x_1 = r$. For $n = 2$, we note that once x_1 has been selected, the value for x_2 is fully determined. This is illustrated below, with in the first line a total of r ones to the left of the separator "$|$" at the right indicating that $x_1 = r$ and $x_2 = 0$. Now move the separator to the left: each time the separator $|$ moves a position to the left by jumping over a one, it changes the one into a two, so that x_1 decreases by one and x_2 increases by one. This continues until there is no one left for the separator to jump over: this means it is at the left of r twos (which all used to be ones), indicating that $x_1 = 0$ and $x_2 = r$:

$$
\begin{array}{ll}
11\ldots111| : & x_1 = r, x_2 = 0 \\
11\ldots11|2 : & x_1 = r-1, x_2 = 1 \\
11\ldots1|22 : & x_1 = r-2, x_2 = 2 \\
11\ldots|222 : & x_1 = r-3, x_2 = 3 \\
\quad\ldots & \\
1|\ldots2222 : & x_1 = 1, x_2 = r-1 \\
|2\ldots2222 : & x_1 = 0, x_2 = r-1
\end{array}
$$

Repeating the above with one additional separator $|$ at the far right does not change the varying x_1 and x_2 values, but indicates (for the same r-value) the potential presence of an additional variable x_3 which is so far just fixed at $x_3 = 0$ (because there is nothing to the right of the new rightmost separator):

$$
\begin{array}{ll}
11\ldots111|| : & x_1 = r, x_2 = 0, x_3 = 0 \\
11\ldots11|2| : & x_1 = r-1, x_2 = 1, x_3 = 0 \\
11\ldots1|22| : & x_1 = r-2, x_2 = 2, x_3 = 0 \\
11\ldots|222| : & x_1 = r-3, x_2 = 3, x_3 = 0 \\
\quad\ldots & \\
1|\ldots2222| : & x_1 = 1, x_2 = r-1, x_3 = 0 \\
|2\ldots2222| : & x_1 = 0, x_2 = r, x_3 = 0
\end{array}
$$

In each line above we now let the rightmost separator move to the left, each time jumping over a two while changing it into a three and as far it can go until bumping into the separator to its left (i.e., until there is no two left):

$$
\begin{array}{ll}
11\ldots111|| : & x_1 = r, x_2 = 0, x_3 = 0 \\
11\ldots11|2| : & x_1 = r-1, x_2 = 1, x_3 = 0 \\
11\ldots11||3 : & x_1 = r-1, x_2 = 0, x_3 = 1 \\
11\ldots1|22| : & x_1 = r-2, x_2 = 2, x_3 = 0 \\
11\ldots1|2|3 : & x_1 = r-2, x_2 = 1, x_3 = 1 \\
11\ldots1||33 : & x_1 = r-2, x_2 = 0, x_3 = 2 \\
11\ldots|222| : & x_1 = r-3, x_2 = 3, x_3 = 0 \\
11\ldots|22|3 : & x_1 = r-3, x_2 = 2, x_3 = 1 \\
11\ldots|2|33 : & x_1 = r-3, x_2 = 1, x_3 = 2 \\
11\ldots||333 : & x_1 = r-3, x_2 = 0, x_3 = 3 \\
\quad\ldots & \\
1|\ldots2222| : & x_1 = 1, x_2 = r-1, x_3 = 0 \\
1|\ldots222|3 : & x_1 = 1, x_2 = r-2, x_3 = 1 \\
1|\ldots22|33 : & x_1 = 1, x_2 = r-3, x_3 = 2 \\
1|\ldots2|333 : & x_1 = 1, x_2 = r-4, x_3 = 3 \\
1|\ldots|3333 : & x_1 = 1, x_2 = r-5, x_3 = 4 \\
\quad\ldots & \\
|2\ldots2222| : & x_1 = 0, x_2 = r, x_3 = 0 \\
|2\ldots222|3 : & x_1 = 0, x_2 = r-1, x_3 = 1 \\
|2\ldots22|33 : & x_1 = 0, x_2 = r-2, x_3 = 2
\end{array}
$$

|2. . .2 |333: $x_1 = 0, x_2 = r - 3, x_3 = 3$
|2. . . |3333: $x_1 = 0, x_2 = r - 4, x_3 = 4$
. . .
| |. . .33333: $x_1 = 0, x_2 = 0, x_3 = r$

Thus by considering all ways that two separators can be positioned over $r+2$ positions we get all possible non-negative integers x_1, x_2, x_3 such that $\sum_{i=1}^{r} x_i = r$: the number of positions to the left of the leftmost separator is x_1, the number of positions between the separators is x_2, and the number of positions to the right of the rightmost separator is x_3.

This generalizes in the obvious manner to $n = 4$: add to each line an additional separator on a new rightmost position (and thus a new variable $x_4 = 0$), and then let the new separator move to the left as far as it can go until bumping into the formerly rightmost separator, while changing threes into fours and while keeping track of the increasing x_4-values and decreasing x_3-values.

As a result we find that the number of r-combinations from n with replacement (and thus the number of non-negative integer solutions to $\sum_{i=1}^{n} x_i = r$) equals $C(r + (n-1), (n-1)) = \binom{n+r-1}{n-1}$, namely the number of ways $n-1$ separators can be positioned over $r+n-1$ positions. It follows (after introducing a *slack variable* x_{n+1}) that the number of non-negative integer solutions to $\sum_{i=1}^{n} x_i \leq r$ equals $\binom{n+r}{n}$.

Theorem 3.4. (Generalized pigeonhole principle)

If n objects are place in k boxes, then there is at least one box containing $\lceil \frac{n}{k} \rceil$ objects.

An alternative to this says that if n objects are put into k boxes, then there is at least one box containing at most $\lfloor \frac{n}{k} \rfloor$ many objects.

Proof. Proof by contraposition

Suppose none of the boxes contain more than $\lceil \frac{n}{k} \rceil - 1$ objects. Then we have that at most there are $k(\lceil \frac{n}{k} \rceil - 1)$ objects. Then:

$$k(\lceil \frac{n}{k} \rceil - 1) < k(\frac{n}{k} + 1 - 1)$$

□

Example 14. Constructive example of the pigeonhole principle

We are given a set of p desktops and k printers with $p > k$. We want to find a wiring arrangement such that for any subset of desktops with less than or equal to p desktops at hand, the desktops are able to print simultaneously. The question though is to do this most efficiently using least number of cables.

Proposed solution:
Connect first $d - p$ desktops to first p printers and connect each remaining desktop to each remaining printer resulting in $p + (d-p)p$ many cables.

Proof of simultaneous printing:
If we take a subset with the first k desktops we are done. Otherwise for a subset with first z where $z < k$ desktops and remaining $k - z$ laptops the first z go to first z printers and the remaining since wired to each printer sequentially go to $z+1, \ldots, k$

Proof of optimality:
Now supposing we use at most $p + (d-p)p - 1$ cables, we have that there is a printer with at most $\lfloor \frac{p+(d-p)p-1}{p} \rfloor = d - p$ cables. Now to all at least p desktops not connected to this printer, there are at most $p - 1$ available printers.

3.4 Lec 20. notes

Theorem 3.5. Vandemonde's idendtity

$$\binom{m+n}{r} = \sum_{i=0}^{r} \binom{m}{r-i}\binom{n}{i}$$

Proof. **A nice neat combinatorial proof for Mr.Vandemonde and his theorem** Now our goal is to reach a formula for $\binom{m+n}{r}$. Suppose we divide $m+n$ elements into two disjoint sets of size m and n. Now we want to pick a total of r elements with order being irrelevant. If we pick k elements from set of size n, then we must pick $r-k$ from the latter. Hence one such pick results in $\binom{m}{r-k}\binom{n}{k}$ choices **for each selection** of k. That is, the total number of ways to pick $\binom{m+n}{r}$ is clearly $\sum_{i=0}^{r}\binom{m}{r-i}\binom{n}{i}$. □

Theorem 3.6. Binomial theorem

$$(X+Y)^z = \sum_{r=0}^{z} \binom{z}{r}(X)^{z-r}(Y)^r$$

Proof. **Combinatorial proof** The expansion of the product $(X+Y)^z$ contains terms where each time X or Y is picked. For instance if $z=8$ and we want to find in how many ways we may pick X^5 this is equivalent to picking 5 X and 3 of Y from each of $(X+Y)$. In total, order doesn't matter hence we have that the term X^5Y^3 is picked $\binom{8}{5}$ many times. □

Proof. **Inductive proof** Now we simplify the binomial theorem to the statement $(1+Z)^n = \sum_{i=0}^{n}\binom{n}{i}Z^i$ as if we take $Z = \frac{Y}{X}$ then $X^n\sum_{i=0}^{n}\binom{n}{i}Z^i = \sum_{i=0}^{n}\binom{n}{i}(X)^{n-i}(Y)^i$
Inductive hypothesis: $(1+Z)^k = \sum_{r=0}^{k}\binom{k}{r}Z^r$
Need to show:$(1+Z)^{k+1} = \sum_{r=0}^{k+1}\binom{k+1}{r}Z^r$

$$\begin{aligned}
(1+Z)^{k+1} &= (1+Z)(1+Z)^k \\
&= (1+Z)\sum_{r=0}^{k}\binom{k}{r}Z^r \\
&= \sum_{r=0}^{k}\binom{k}{r}Z^r + Z\sum_{r=0}^{k}\binom{k}{r}Z^r \\
&= \sum_{r=0}^{k}\binom{k}{r}Z^r + \sum_{r=0}^{k}\binom{k}{r}Z^{r+1} \\
&= \sum_{r=0}^{k}\binom{k}{r}Z^r + \sum_{L=1}^{k+1}\binom{k}{L-1}Z^L \text{ letting } r-1=L \\
&= \sum_{\color{red}{r=0}}^{k}\binom{k}{r}Z^r + \sum_{r=1}^{\color{red}{k+1}}\binom{k}{r-1}Z^r \\
&= Z^0 + \sum_{r=1}^{k}(\binom{k}{r} + \binom{k}{r-1}Z^r) + \binom{k}{(k+1)-1}Z^{k+1} \\
&= Z^0 + \sum_{r=1}^{k}\binom{k+1}{r}Z^r + \binom{k+1}{k+1}Z^{k+1} = \sum_{r=0}^{k+1}\binom{k+1}{r}Z^r
\end{aligned}$$

□

3.5 Lec 21. notes

An unmentioned theorem about sequences is the following:

Theorem 3.7. Any sequence of n^2+1 distinct values contains an increasing or decreasing subsequence of length $n+1$.

Definition 9. The classical definition of a probability is that given by Laplace. This assumes that every event is as likely to occur. For a sample space P, probability of some E is $\frac{E}{P}$, namely the number of occurrences relative to the total number of occurrences.

Definition 10. Yet a better definition of probability is the probability distribution. For instance if we were flipping a biased coin, the classical definition would not work. The probability distribution assigns some $p(S)$ to each event such that:

$$0 \leq p(S) \leq 1$$
$$\sum p(i) = 1$$

Definition 11. A **uniform distribution** is the case when over some sample space, for $s \in S$, we have that $p(s) = \frac{1}{|S|}$ Rolling a dice, flipping a coin are all uniform distributions. Yet the maximum of rolling two dice is not a uniform distribution.

And now we present what is known as Boole's inequality:

Theorem 3.8.

$$p(\bigcup_{i=1}^{L} A_i) \leq p(\sum_{i=1}^{L} p(A_i))$$

Definition 12. Conditional probability

$$p(E|F) = \frac{p(E \cap F)}{p(F)}$$

Theorem 3.9. Independence We take a set of events to be mutually independent if:

$$\Pi_{i=1}^{L}(p(i)) = \bigcap_{i=1}^{L} p(i)$$

Note that while mutual independence implies pairwise independence, the latter doesn't apply.

Theorem 3.10. Bayes' theorem and two formulations for it The most common form of Bayes' theorem is:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

but through a simple derivation we also have that:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B|A)p(A) + p(B|A')p(A')}$$

3.6 Lec 22. notes

We may use polynomials to represent outcomes of a certain event. Suppose we are rolling a die. We let tails be represented by X^0 and heads by X^1 then $(1+X)^n = \sum_{i=0}^{n} \binom{n}{i} 1 \cdot X^n$

Similarly, if we wrote $(X + hdots + X^6)^2$ the coefficients of the expansion would tell us in how many ways we can achieve sums of rolling a die twice. Given this, we get the more general form for the possible sums when rolling a die n many times is obtained from first principles as follows:

for the $n = 1$ case we have: $X + \ldots + X^6 = X(1 + \ldots + X^5) = X\frac{X^6-1}{X-1}$ which means that for the general n case we have $X^n\frac{(X^6-1)^n}{(X-1)^n}$

Now a very crucial point is to realize the essence of a random variable which is neither variable nor random. It simply is a real-valued mapping $f : E \to \mathbb{R}$ And another subtlety is that $p(s) \neq p(X = x)$. The former is simply the probability of some $s \in E$ happening whereas the latter defined as:

$$p(X = x) = \sum_{\{s:s\in E:X(s)=x} p(s)\}$$

The simplest example is when flipping two dice we have that $P(X = 1) = P(TT) + P(HH)$

Definition 13. Expected value We define $E(x)$ as:

$$E(x) = \sum_{s\in E} p(s)X(s) = \sum_{x\in X(s)} xp(X = x)$$

where $X(s)$ is the random variable.

And we have that the excpected value of n independent Bernoulli trials is np

3.7 Lec 23. notes

Suppose we roll an n sided dice twice and define a random distribution $Y(i, j) = max(i, j)$ now we ask, what is $E(Y)$ Well realizing that $E(Y) = \sum_{s\in S} p(s)Y(s)$ we get the following:

$$\begin{aligned}
E(Y) &= \frac{\max(1,1)}{n^2} + \frac{\max(1,2)}{n^2} + \frac{\max(1,3)}{n^2} + \ldots + \frac{\max(1,n)}{n^2} + \\
&\quad \frac{\max(2,1)}{n^2} + \frac{\max(2,2)}{n^2} + \frac{\max(2,3)}{n^2} + \ldots + \frac{\max(2,n)}{n^2} + \\
&\quad \frac{\max(3,1)}{n^2} + \frac{\max(3,2)}{n^2} + \frac{\max(3,3)}{n^2} + \ldots + \frac{\max(3,n)}{n^2} + \\
&\quad \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
&\quad \frac{\max(n,1)}{n^2} + \frac{\max(n,2)}{n^2} + \frac{\max(n,3)}{n^2} + \ldots + \frac{\max(n,n)}{n^2} \\
&= \frac{1}{n^2} + \frac{2}{n^2} + \frac{3}{n^2} + \ldots + \frac{n}{n^2} + \\
&\quad \frac{2}{n^2} + \frac{2}{n^2} + \frac{3}{n^2} + \ldots + \frac{n}{n^2} + \\
&\quad \frac{3}{n^2} + \frac{3}{n^2} + \frac{3}{n^2} + \ldots + \frac{n}{n^2} + \\
&\quad \vdots \qquad\qquad\qquad \vdots \\
&\quad \frac{n}{n^2} + \frac{n}{n^2} + \frac{n}{n^2} + \ldots + \frac{n}{n^2}.
\end{aligned}$$

This shows that each $\frac{i}{n^2}$ occurs $2i - 1$ many times hence we get:

$$E(Y) = \sum_{i=1}^{n} \frac{i(2i-1)}{n^2}$$

And using our earlier knowledge of summations we have:

$$\begin{aligned}
\sum_{i=1}^{n} i(2i-1) &= \big(2\sum_{i=1}^{n} i^2\big) - \big(\sum_{i=1}^{n} i\big) \\
&= 2\frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2} \\
&= n(n+1)\big(\frac{2n+1}{3} - \frac{1}{2}\big) \\
&= \frac{n(n+1)(4n-1)}{6}.
\end{aligned}$$

It follows that:

$$E(Y) = \frac{(n+1)(4n-1)}{6n}$$

Remark 3.7.1. We note that $E(X)$ is a linear map. Namely $E(ax+b) = aE(x) + E(b)$

Theorem 3.11. Markov's inequality We ask what is the probability that some random variable deviates by a factor of M from $E(x)$. Markov answers this as:

$$P(X \geq M \cdot E(X)) \leq \frac{1}{M}$$

Definition 14. Variance

$$Var(x) = \sum_{s \in S} (X(s) - E(x))^2 p(s)$$

which is also:

$$E((x(s) - E(x))^2)$$

and simplifies to:

$$E(X^2) - E(X)^2$$

Chebyshev's inequality Let δ be some positive real number and let Δ be the event $\{s \in S : |X(s) - E(X)| \geq \delta\}$ that a random variable X deviates at least δ from its expected value $E(X)$. Because $V(X) = \sum_{s \in S}(X(s) - E(X))^2 p(s)$ can be written as

$$V(X) = \sum_{s \in \Delta} (X(s) - E(X))^2 p(s) + \sum_{s \in S \setminus \Delta} (X(s) - E(X))^2 p(s)$$

and the second sum is non-negative, it follows that

$$V(X) \geq \sum_{s \in \Delta} (X(s) - E(X))^2 p(s).$$

But $s \in \Delta$ implies that $|X(s) - E(X)| \geq \delta$ and thus that $(X(s) - E(X))^2 \geq \delta^2$, so that

$$V(X) \geq \delta^2 \sum_{s \in \Delta} p(s) = \delta^2 p(\Delta).$$

It follows that

$$p(\Delta) \leq V(X)/\delta^2$$

and thus that

$$p(|X(s) - E(X)| \geq \delta) \leq V(X)/\delta^2.$$

This is Chebyshev's inequality. With $\delta = \sigma(X) = \sqrt{V(X)}$ it becomes (if $V(X) \neq 0$)

$$p\big(|X(s) - E(X)| \geq \sigma(X)\big) \leq 1,$$

which is not saying much except that deviating by $\sigma(X)$ from $E(X)$ is something that should be considered to be "standard": Chebyshev's inequality provides no useful upper bound on the probability to be $\sigma(X)$ away from $E(X)$. On the other hand, Chebyshev's inequality also tells us that being "two (or four) standard deviations away from the expected value" (take $\delta = 2\sigma(X)$ (or $\delta = 4\sigma(X)$)) has a probability of at most $25\%(=\frac{1}{4})$ (or $6.25\%(=\frac{1}{16})$), which may be useful.

Summary In short, Markov tells us that the probability of being M away from the mean is $\frac{1}{M}$ and Chebyshev tells us that the probability of deviating by a factor of K from $\sigma(x)$ is $\frac{1}{K}$

We now note that variance is additive only if $E(x)$ is multiplicative. And we have that $E(x)$ is multiplicative iff. the random variables are. That is $P(X = x \wedge X = y) = P(X = x)P(X = y)$