Advanced Computation

alp.ozen

1 Propositional logic

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:

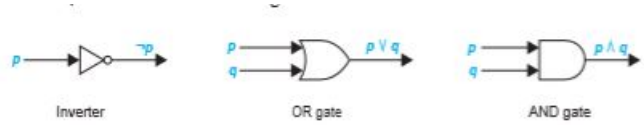


Figure 1: Logic gates

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. $p \wedge \neg p$ is a contradiction
$p \vee \neg p$ is a tautology

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

$$\begin{aligned} \neg(p \vee \neg p \wedge p \vee q) \\ \neg(T \wedge p \vee q) \\ \neg(p \vee q) \\ \neg p \wedge \neg q \end{aligned}$$

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. 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:

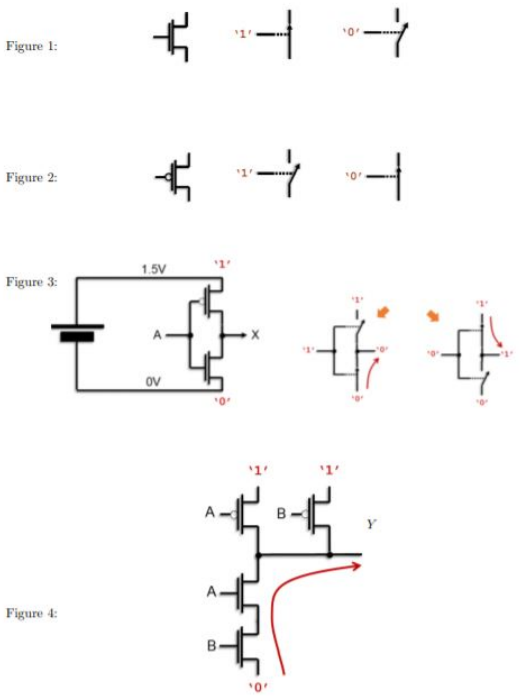
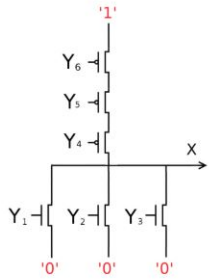


Figure 3: Caption



(*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 $x \in S \land P(x) \equiv Q(x)$ 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:

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

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:

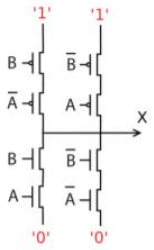
$$\begin{aligned} \equiv \neg\forall x \in S\ P(x) &\equiv \neg(\forall x \in S \to P(x)) \\ &\equiv \exists x \neg(x \in S \to P(x)) \\ &\equiv \exists x \neg(x \notin S \lor P(x)) \\ &\equiv \exists x(x \in S \land \neg P(x)) \\ &\equiv x \in S,\ \neg P(x) \end{aligned}$$

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 $A \land B$, lower part must evaluate $\neg(A \land B)$

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:

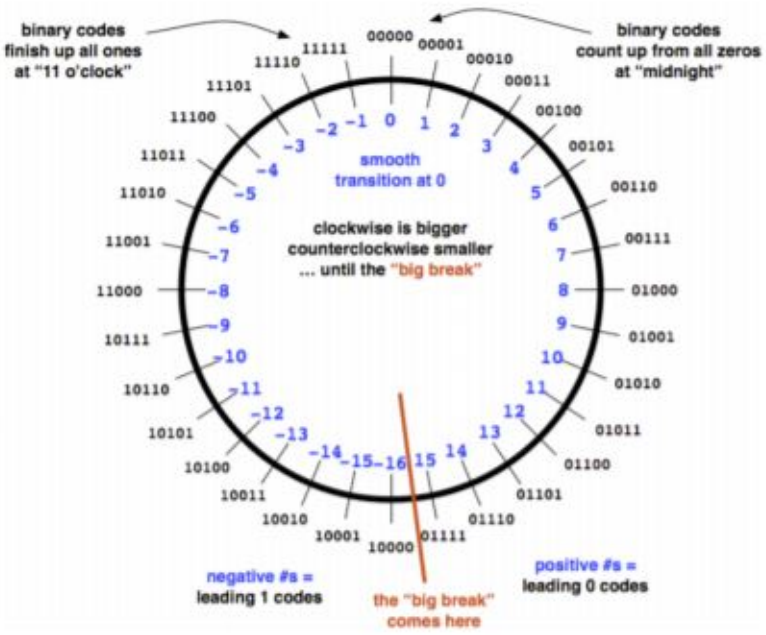
- ○ $X = \neg(A \leftrightarrow B)$
- ○ $X = A \leftrightarrow B$
- ○ $X = A \vee B$
- ○ the circuit may be shorted (connecting '0' to '1')

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 $0 \leq a < 2^{k-1}$ (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 $0 \leq a < 2^{k-1}$ 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 $-2^{k-1} < a \leq 1$ (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 $-2^{k-1} \leq a, b < 2^{k-1}$(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:

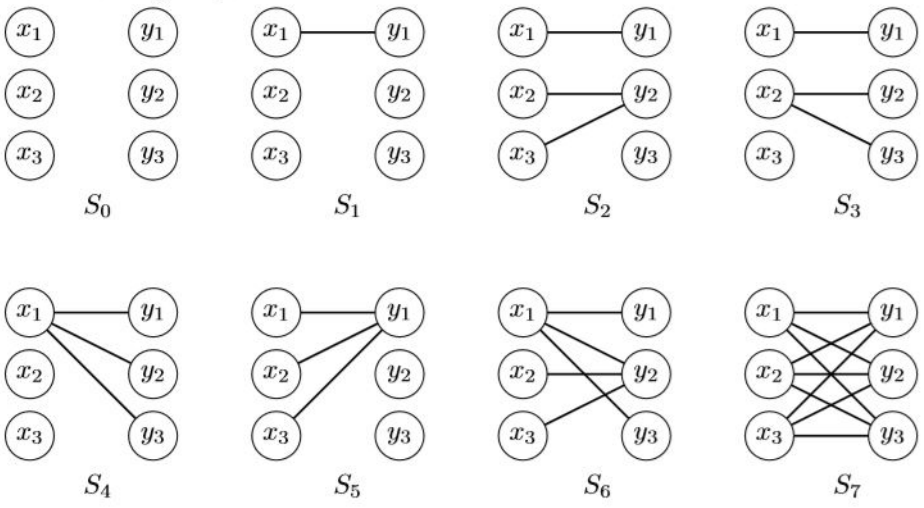
$$\exists! x P(x) \equiv \exists x (P(x) \wedge P(y) \rightarrow (x = y))$$

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

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

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 $0 \leq i < 8$ 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	$\exists x \exists y\, S(x,y)$	$\exists x \forall y\, S(x,y)$	$\rightarrow$	$\forall y \exists x\, S(x,y)$	$\exists y \forall x\, S(x,y)$	$\rightarrow$	$\forall x \exists y\, S(x,y)$	$\forall x \forall y\, S(x,y)$
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	$\rightarrow$	true	false		false	false
S_5	true	false		false	true	$\rightarrow$	true	false
S_6	true	true	$\rightarrow$	true	true	$\rightarrow$	true	false
S_7	true	true	$\rightarrow$	true	true	$\rightarrow$	true	true

Figure 8: Nesting order demo

And notice here how $\exists x \forall y S(x, y) \rightarrow \forall y \exists x S(x, y)$ 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 qunatifiers.

Definition 3. 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

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

1.11 Lec 08. notes

A **proof** is a vald argument establishing the truth of a statement.
We list some common proof methods and elaborate:
Direct proof: When trying to prove a statement like $p \rightarrow q$ we consider the only case that $p \rightarrow q$ 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 $p \rightarrow q \equiv \neg q \rightarrow \neg p$ 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 $\neg p \rightarrow q$ 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 $p \equiv a \rightarrow b$. We know that $\neg p \rightarrow a \wedge \neg b$ and get that $a \wedge \neg b$ is always false showing that p must be true.