Version of 26.1.01

CC112 Discrete Mathematics

D.H.Fremlin

This course is an introduction to formal logic; I will try to explain how to translate verbal arguments into 'argument forms', and then to test their validity. It is based on the first three chapters of

S.S.Epp, *Discrete Mathematics with Applications*, Brookes/Cole; QA 39.

A case study: `QuickSort`

In preparation for the ideas of this course I will describe an interesting and important algorithm. *Sorting* a list is one of the most fundamental operations of computation. The algorithm `quicksort` is short (at least in a language like C which allows recursion), exceedingly economical in space requirements, and fast on 'most' lists. Here is a version of it:

```
public class QuickSort

{static boolean goodorder(int i,int j,double num[])
  {return( num[i] <= num[j]);}

static int partition(int left,int right,int qslist[],double num[])
  {int pivot = qslist[left]; int lo = left; int hi = right;
    while( lo < hi )
      {if( goodorder(pivot,qslist[hi],num)) --hi;
        else{qslist[lo]=qslist[hi]; ++lo; qslist[hi]=qslist[lo];}}
    qslist[lo] = pivot;
    return(lo);}

static void quicksort(int left,int right,int qslist[],double num[])
  {if(left < right)
    {int dp = partition(left,right,qslist,num);
    quicksort(left,dp - 1,qslist,num);
    quicksort(dp + 1,right,qslist,num);}}

public static void main(String[] args)
  {double num[] = {0.53,0.12,0.98,0.32,0.18,0.72,0.80,0.46,0.32,0.21};
  int qslist[] = new int[num.length];
  for(int i = 0; i < num.length; ++i) qslist[i] = i;
  quicksort(0,num.length - 1,qslist,num);
  for(int i = 0;i < num.length;++i)
    System.out.print(qslist[i]+" "); System.out.println();}}
```

Now let's try to understand what's being done here. We are going to try to put numbers in a 'correct' order, as defined by the function `goodorder`; to make it easy without being trivial, I define an array `num[]` and say that i and j are to be counted as being in correct order if `num[`i`]` is less than or equal to `num[`j`]`.

```
{static boolean goodorder(int i,int j,double num[])
  {return( num[i] <= num[j]);}
```

Since

`num[1]`$= 0{\cdot}12$, `num[4]`$= 0{\cdot}18, \ldots ,$ `num[2]`$= 0{\cdot}98$,

the output list will be

$$1 \quad 4 \quad 9 \quad 3 \quad 8 \quad 7 \quad 0 \quad 5 \quad 6 \quad 2.$$

These notes are distributed to students on the understanding that they have not been properly checked for errors. If you spot a mistake, tell the author!

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

The main program is

```
public static void main(String[] args)
  {double num[] = {0.53,0.12,0.98,0.32,0.18,0.72,0.80,0.46,0.32,0.21};
  int qslist[] = new int[num.length];
  for(int i = 0; i < num.length; ++i) qslist[i] = i;
  quicksort(0,num.length - 1,qslist,num);
  for(int i = 0;i < num.length;++i)
    System.out.print(qslist[i]+" "); System.out.println();}}
```

All this does is to set up the array `num[]`, initialize `qslist[]` with `qslist[`i`]`$=i$ for all relevant i, call `quicksort`, and print out the final version of `qslist[]`.

Now for the function `quicksort`.

```
static void quicksort(int left,int right,int qslist[],double num[])
  {if(left < right)
    {int dp = partition(left,right,qslist,num);
    quicksort(left,dp - 1,qslist,num);
    quicksort(dp + 1,right,qslist,num);}}
```

What is this doing? It is calling on the function `partition`. We shall see shortly that `partition` does two things. It *returns* an integer value `dp`, but it also changes `qslist[]`. This is the key to the whole program. The integer parameters `left` and `right` refer to the segment of `qslist[]` which is being dealt with. (You can see from the main program above that on the first call to `partition`, we shall have `left` $=0$ and `right` $=9$, the two ends of the array `qslist[]`.) The idea of `partition` is that, at the end of the call, `qslist[left]`...`qslist[dp`-1`]` will carry numbers which will *precede* `qslist[dp]` in the final list, while `qslist[dp`$+1$`]`...`qslist[right]` will carry numbers which will *follow* `qslist[dp]` in the final list. If we can get this right, then we now only have to sort the left and right sections to get the whole list done; that's the last two lines in `quicksort`.

So now look at `partition`:

```
static int partition(int left,int right,int qslist[],double num[])
  {int pivot = qslist[left]; int lo = left; int hi = right;
    while( lo < hi )
      {if( goodorder(pivot,qslist[hi],num)) --hi;
        else{qslist[lo]=qslist[hi]; ++lo; qslist[hi]=qslist[lo];}}
    qslist[lo] = pivot;
    return(lo);}
```

The idea here is that we start by picking, not the returned value `dp` of `partition`, but the number `pivot` which will eventually appear as `qslist[dp]`. For simplicity, the program written here takes `pivot = qslist[left]`, the number at the bottom end of the segment being sorted. (If you ever want to write a version of `quicksort` for general use, you should try to be a little more intelligent at this point.) The algorithm now sets out to shuffle the numbers in the segment `qslist[left]`...`qslist[right]` in such a way that each comes out on the correct side of `pivot`, which we rather expect to finish up somewhere in the middle. The procedure is to 'squeeze' the list between the limits `lo` and `hi`, starting with the whole segment `left`, `right` under consideration. The economy of the algorithm derives from the fact that only one number needs to be moved at a time. At any moment, all the numbers in `qslist[left]`...`qslist[lo`-1`]` will be destined to end on the left of `pivot`; the numbers in `qslist[hi`$+1$`]`...`qslist[right]` will be destined to end on the right of `pivot`; the numbers in `qslist[lo`$+1$`]`...`qslist[hi]` are still to be arranged; and `qslist[lo]` has some random number in it. When squeezing, we look at one end of the section we are still working on – it doesn't matter much which – this program looks at `qslist[hi]`. If this is correctly at the high end, that is, if `goodorder(pivot,qslist[hi],num)` $=1$, then we just reduce `hi` by 1. Otherwise, we have to put `qslist[hi]` at the bottom of the segment, in the spare location `qslist[lo]`; increase `lo` by 1; and shift the number in the new location `qslist[lo]` to the position `hi` to be looked at next move, so that `qslist[lo]` is again spare.

(Have you noticed that I've been saying the whole time 'a' correct order not 'the' correct order? The point is that if two of the numbers in `num[]` are actually the same, `goodorder` will return `T` whichever order they turn up in, and figuring out which way round they will come in the final list will require a much more complicated analysis! If you had an application in mind in which it mattered – e.g., if you wanted to list students in the order of their marks in a test, and to put students with the same marks in alphabetical order – you would have to look carefully at `goodorder` and make sure it did the right thing.)

Now what would a 'proof' that this program works look like? What we want is an argument which we can rely on *without running the program.* Of course in practice all programmers rely heavily on testing their programs, running them with a variety of inputs and checking that the outputs are always what the customer would want. The trouble is that for any non-trivial program there are too many possible inputs. You could test a program like the one above indefinitely without finding out that if (for instance) the length of the list is a multiple of 113 the program fails ... (Remember the Pentium chip which gave incorrect answers to occasional double-precision divisions.) So it would be very nice if we could develop techniques for checking the *logic* of programs.

The discussion I have already written down is an attempt at such a check for the version of `quicksort` here. Like any human logic, it is full of jumps and assumptions. One of the aims of this course is to look at the structure of logical arguments with a view to *formalizing* them; that is, putting them in a language which could, in principle, be accessible to a computer to check that they're correct. (Of course, you would need a good deal of faith in your checking program ... but that's another story.)

At the end of last term, we spent quite a lot of time on 'proofs by induction', and these are certainly one of the basic elements of any proof that a program is correct. For instance, in the function `quicksort` above, if we believe that `partition` is doing its job, then we can argue as follows:

Let P(n) be the statement

'if `right`$-$`left` $= n$ then `quicksort` will put the segment `qslist[left]`...`qslist[right]` into a correct order.'

Base step If $n = 0$ then `right` $=$ `left` so `quicksort` does nothing, which is the right thing to do.

Inductive step Assuming that P(i) is true whenever $0 \leq i < k$, where $k \geq 1$, we have to prove P(k). Now if `right`$-$`left` $= k$, then `partition` will find `dp` such that `right`$\leq$`dp`$\leq$`left` and arrange that all the numbers in `list[left]`...`list[dp`$-$`1]` are numbers which are allowed to precede `list[dp]`, while all the numbers in `list[dp+1]`...`list[right]` are numbers which are allowed to follow `list[dp]`. Since `dp`$-$`1`$-$`left`$<$`right`$-$`left` $= k$ and `right`$-$`(dp+1)` $<$ `right`$-$`left` $= k$, the inductive hypothesis tells us that the calls `quicksort(left,dp-1,qslist,num)` and `quicksort(dp+1,right,qslist,num)` will arrange these sections correctly, so that we shall finish with the whole section `list[left]`...`list[right]` in a correct order. So the induction continues.

Now the argument above is a perfectly typical application of the Second Principle of Induction. But did you spot the gap? I wrote

Since `dp`$-$`1`$-$`left` $<$ `right`$-$`left` ...

If you look back at the arguments I wrote out last term, you will I hope see that I tried to take care to write things like 'since $0 \leq i < k$ and $0 \leq j < k$'. Does it matter that I left out the first '$0 \leq$' above? Well, it might. For `partition` can very well produce `dp` $=$ `left`, in which case `dp`$-$`1` $<$ `left`, and the call `quicksort(left,dp-1,qslist,num)` corresponds to the case `right`$<$`left`. So actually we have to start the induction at $n = -1$, not $n = 0$! But otherwise it's OK (I hope).

What I hope this last paragraph will persuade you of is the importance of *specification.* You need to be absolutely precise about exactly what can happen at any stage of the program. Now this is going to land us with some rather complicated sentences. Looking at the middle section of `partition`, we shall have to say things like

if 0 $\leq$`lo` and `lo`$\leq$`hi` and `hi`$<$`length`
and whenever $0 \leq i <$ `lo` then `num[list[`i`]]`$\leq$`num[pivot]`
and whenever `hi` $< i <$ `length` then `num[pivot]`$\leq$`num[list[`i`]]`
and $\{$`list[`i`]`$: 0 \leq i <$ `length`$, i \neq$ `lo`$\} = \{0, \ldots,$`length`$-1\} \setminus \{$`pivot`$\}$
then ...

So we already have quite an elaborate combination of elements 'and', 'if... then' and 'for every'. And it's compound sentences like this which I shall be talking about for the rest of the term.

[You may prefer to read `Quicksort` in C rather than in Java:]

```
void quicksort(int,int,int[],float[]);
int partition(int,int,int[],float[]);
int goodorder(int,int,float[]);

int goodorder(int i, int j, float num[])
  {if (num[i] <= num[j]) return 1; else return 0;}

main()
  {float num[10]
    ={0.53,0.12,0.98,0.32,0.18,0.72,0.80,0.46,0.32,0.21};
  int length=10; int qslist[10]; int i;
  for (i=0; i < length; i++) qslist[i]=i;
  quicksort(0,length-1,qslist,num);
  for (i=0; i < length; i++) cout << qslist[i] << " ";
  cout << endl;}

void quicksort(int left,int right,int qslist[],float num[])
  {int dp;
  if (left<right)
    {dp = partition(left,right,qslist,num);
    quicksort(left,dp-1,qslist,num);
    quicksort(dp+1,right,qslist,num);}}

int partition(int left,int right,int qslist[],float num[])
  {int pivot, lo, hi;
  pivot = qslist[left];
  lo = left;
  hi = right;
  while (lo < hi)
    {if (goodorder(pivot,qslist[hi],num)) hi = hi - 1;
    else {qslist[lo]=qslist[hi]; lo=lo+1;
        qslist[hi]=qslist[lo];}}
  qslist[lo] = pivot;
  return lo;}
```

Version of 26.1.01

Propositional Calculus: the Algebra of Logic

The ‘Propositional Calculus’ is the branch of logic which deals with the formal manipulation of compound statements. To make any sense out of that sentence, we had better start with some ‘simple’ or ‘atomic’ or ‘indecomposable’ statements:

It’s Tuesday.

The sun is shining.

It’s three o’clock.

It’s early in the morning.

$2+2=5$.

`left` $\leq$ `right`.

`goodorder`$(i,j,$`num`$)=1$.

(These two are taken from the program `quicksort` considered in the first lecture.)

38179 is a prime.

These are all sentences which are grammatically complete and for which it makes sense to say that they are ‘true’ or ‘false’ at any moment. We don’t have to know which; for instance, we may not know whether 38179 is a prime. What matters is that it makes sense to ask the question: is the sun shining? is 38179 a prime?

Compound propositions Starting from 'atomic' propositions like those above, we can put them together in various ways to form 'compound' propositions:

The sun is shining *and* it's three o'clock.

$\texttt{left} < \texttt{right}$ *or* $\texttt{hi} = \texttt{lo}$.

Actually there is something we can do with a single proposition; we can look at its opposite:

It *isn't* Tuesday.

Starting from any propositions p and q, we can form

— their **conjunction** $p \wedge q$, 'p *and* q',

— their **disjunction** $p \vee q$, 'p *or* q',

— the **negation** $\neg p$, 'not-p', $\sim p$ of one of them.

[We immediately run into a problem: how on earth are we to remember which of 'conjunction', 'disjunction', '$\vee$', '$\wedge$', 'and' and 'or' go together? This is what flashcards are for, of course. But there's another method of getting $\vee$ and $\wedge$ the right way up: look at the formulae

$$A \cup B = \{x : x \in A \text{ or } x \in B\} = \{x : (x \in A) \vee (x \in B)\},$$

$$A \cap B = \{x : x \in A \text{ and } x \in B\} = \{x : (x \in A) \wedge (x \in B)\}.$$

You should certainly not think of $\vee$ as being the same thing as $\cup$, but there will be many occasions in the work of this course in which logical formulae containing $\vee$ and $\wedge$ will correspond to formulae about sets containing $\cup$ and $\cap$. In the same way, we can match $\neg$ with complementation ^{c}. I'm afraid this still doesn't help decide which of 'conjunction' and 'disjunction' is which. The Latin prefix 'con'- commonly means 'together', so I suppose we can try to remember that the *con*junction of p and q is when they are true *together*. But these words are anyway less important, at least at first.]

Now the really important thing about compound propositions is that we can decide whether they're true by looking at their components. Thus the sentence

The sun is shining *and* it's three o'clock

is true if, and only if, the two sentences

The sun is shining

and

It's three o'clock

are both true. While the sentence

$\texttt{left} < \texttt{right}$ *or* $\texttt{hi} = \texttt{lo}$

is true if, and only if, at least one of the sentences

$\texttt{left} < \texttt{right}$

$\texttt{hi} = \texttt{lo}$

is true. (Remember that, unless explicitly told otherwise, we always interpret 'or' inclusively; maybe both are true at once.) And the sentence

It isn't Tuesday

is true if, and only if, the sentence

It's Tuesday

is false.

Truth tables

Generally we have the rules

$p \wedge q$ is true if p and q are both true, and is false if either is false;

$p \vee q$ is true if either of p and q is true, and is false if both are false;

$\neg p$ is true if p is false, false if p is true.

Now we can conveniently tabulate these rules in the following forms:

Conjunction

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

The idea of this table is that if we know whether p and q are true or false, we can look at the corresponding line of the table and read off 'T' or 'F' for $p \wedge q$; thus if p is false and q is true, we look at the third line, where F is in the p column and T is in the q column, and see that the $p \wedge q$ has an F, so that $p \wedge q$, 'p and q', is false.

Similarly, the table for $p \vee q$ is

Disjunction

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

and the table for $\neg p$ is

Negation

p	$\neg p$
T	F
F	T

(of course this is much simpler, with only two rows instead of four, because we've only got p to consider).

Note the top row of the table for $p \vee q$: all T's, saying that if both p and q are true then 'p or q' is true.

Compounds of compounds

We allow ourselves to look at compound propositions like $p \wedge q$ for *any* propositions p and q, which could perfectly well themselves be compounds. So we immediately get combinations like $p \wedge (q \vee r)$ for propositions p, q and r. To begin with, we are going to have to be awfully careful with the brackets; there is no reason why $\neg(p \vee q)$ should have anything to do with $(\neg p) \wedge q$, so we had better put brackets in every time. But one of the great virtues of truth tables is that they provide a method of sorting out compounds-of-compounds. For instance, starting from p and q, we can form $p \wedge q$, $\neg(p \wedge q)$, $\neg p$, $\neg q$, $(\neg p) \vee (\neg q)$, ... The corresponding truth table looks like this:

p	q	$p \wedge q$	$\neg(p \wedge q)$	$\neg p$	$\neg q$	$(\neg p) \vee (\neg q)$
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

This is easy to do if we go at it in the right order. The first two columns are just the standard listing of the four possibilities for p and q. (In a sense, it doesn't matter which order these are in, but I'll try to stick to the TT/TF/FT/FF pattern.) Now the column for $p \wedge q$ is just what it was before; a T if p and q are both T, and F otherwise. For the $\neg(p \wedge q)$ column, we just reverse every T or F in the $p \wedge q$ column, because $\neg(p \wedge q)$ is true iff $p \wedge q$ is false. Now turn to the $\neg p$ column. For this, we look left to the p column, and switch everything. Similarly, the $\neg q$ column is read off from the q column, again reversing every value. Finally, for the $(\neg p) \vee (\neg q)$ column, we read the $\neg p$ and $\neg q$ columns, putting a T if we find a T in either of these, and an F only when both $\neg p$ and $\neg q$ have an F.

Logical identities I hope you noticed that the $\neg(p \wedge q)$ and $(\neg p) \vee (\neg q)$ columns in the last truth table are identical. This is important. If you look at the truth tables for expressions in two logical variables p and q, there are only four rows, so there are only $2^4 = 16$ possible columns of T's and F's which can appear. On the other hand, there are infinitely many compound expressions, so the same columns are surely going to turn up again and again. One of the fundamental problems of logic is to decide, given two compound expressions, whether they produce identical truth tables.

I will use the symbol $\equiv$ to declare that this occurs; so that $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$. (Some authors write $\approx$ in place of $\equiv$.) In this case we say that the two compound propositions are **logically equivalent**. The simplest case is perhaps with $\neg(\neg p)$:

p	$\neg p$	$\neg(\neg p)$
T	F	T
F	T	F

We have $p \equiv \neg(\neg p)$; $\neg(\neg p)$ is true just when $\neg p$ is false, that is, when p is true. (Natural languages have trouble with this point. 'I never saw nobody' isn't received standard English, but it's a recognised dialect form for 'I didn't see anyone'. In Portuguese, 'eu não vi ninguem' has the double negative and is the standard form.) In English, it is normally accounted good style to avoid double negatives; that is, to replace $\neg(\neg p)$ by p whenever you can; this is more economical and reduces the risk of confusion.

Another important identity involves three variables: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$. Because we have three variables, the corresponding truth table has $2^3 = 8$ rows, corresponding to all possible allocations of T and F to each of p, q and r:

p	q	r	$q \wedge r$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r$	$(p \vee q) \wedge (p \vee r)$
T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	T	T	T	T
F	T	T	T	T	T	T	T
F	T	F	F	F	T	F	F
F	F	T	F	F	F	T	F
F	F	F	F	F	F	F	F

I hope you noticed that this looks very like the formula $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$. In fact it was recognised very early that we have laws of logic exactly paralleling the laws of sets which I laid out in CC111. Here they are:

Algebraic Laws of Logic

Let p, q and r denote arbitrary propositions (atomic or compound).

1. Idempotent Laws:
   1a: $p \vee p \equiv p$; 1b: $p \wedge p \equiv p$.
2. Associative Laws:
   2a: $(p \vee q) \vee r \equiv p \vee (q \vee r)$ 2b: $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$.
3. Commutative Laws:
   3a: $p \vee q \equiv q \vee p$ 3b: $p \wedge q \equiv q \wedge p$.
4. Distributive Laws:
   4a: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ 4b: $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$.
7. Involution Law: $\neg(\neg p) \equiv p$.
10. De Morgan's Laws:
   10a: $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$ 10b: $\neg(p \wedge q) \equiv \neg p \vee \neg q$.

From the numbering, if nothing else, you will see that I have missed some out. This is because I haven't yet discussed the logical concepts corresponding to the 'universal' and 'empty' sets. Actually these are fairly straightforward. Instead of the 'universal' set, to which every object under consideration is supposed to belong, we have the proposition $\top$ or `TRUE` or t which is always T in any truth table, so that, for instance, the table for $p \wedge \top$ is

p	$\top$	$p \wedge \top$
T	T	T
F	T	F

and $p \wedge \top \equiv p$. Similarly, we have the proposition $\perp$ or `FALSE` or c which is always F, so that the table for $p \vee \perp$ is

p	$\perp$	$p \vee \perp$
T	F	T
F	F	F

and $p \vee \perp \equiv p$. Now we can state the other laws:

5 & 6. Identity Laws:
5a: $p \vee \perp \equiv p$ 5b: $p \wedge \top \equiv p$
6a: $p \vee \top \equiv \top$ 6b: $p \wedge \perp \equiv \perp$.
8 & 9. Complement Laws:
8a: $p \vee (\neg p) \equiv \top$ 8b: $p \wedge (\neg p) \equiv \perp$
9a: $\neg\top \equiv \perp$ 9b: $\neg\perp \equiv \top$.

Please try to avoid the temptation to write $=$ in place of $\equiv$. When dealing with *sets*, the sets $A \cap (B \cup C)$ and $(A \cap B) \cup (A \cap C)$ really are equal; they have exactly the same members. Similarly, in the distributive law for multiplication, $2 \times (3 + 4)$ and $(2 \times 3) + (2 \times 4)$ are equal; they're both 14. But $p \wedge (q \vee r)$ and $(p \wedge q) \vee (p \wedge r)$ are not quite the same proposition. You should think of them rather as different ways of saying the same thing, just as the sentences

'no prime greater than 2 is even'

and

'no even number greater than 2 is prime'

say the same thing in different words. Sorting out the different ways of expressing the same facts is one of the central tasks of formal logic.

Note. I remarked above that we were going to have to be careful about brackets. But the 'associative' laws tell us that in one context we don't have to worry. Since the expressions $p \vee (q \vee r)$ and $(p \vee q) \vee r$ are logically equivalent, we can safely write $p \vee q \vee r$ ('either p or q or r') to mean either; its truth table is

p	q	r	$p \vee q \vee r$
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	F

Similarly $p \wedge q \wedge r$ can be interpreted as either $p \wedge (q \wedge r)$ or as $(p \wedge q) \wedge r$, and its truth table is

p	q	r	$p \wedge q \wedge r$
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Material Implication

In the 'algebraic laws of logic' above I have emphasized the correspondence between the set-theoretic operations $\cup$, $\cap$, ^{c} and the logical connectives $\vee$, $\wedge$ and $\neg$. But set theory and logic are not really the same (I think myself that set theory is a bit easier, which is why I did it first). In particular, some ideas which are important in set theory, like $\setminus$ and $\triangle$ ($A \setminus B = A \cap B^c$, $A \triangle B = (A \cup B) \cap (A \cap B)^c$) are more important than their logical analogues $p \wedge (\neg q)$, $(p \vee q) \wedge (\neg(p \wedge q))$. In the other direction, there is a particular connective which is of vital importance in logic but is not particularly interesting in set theory. This is **material implication**. We write $p \to q$ as an abbreviation for '$q \vee (\neg p)$', 'either q is true or p is false'. Its truth table is given by

p	q	$\neg p$	$q \vee (\neg p)$ $p \to q$
T	T	F	T
T	F	F	F
F	T	T	T
F	F	T	T

Remember $p \to q$ is 'usually' T; it is F only when p is T and q is F.

The formula '$p \to q$' can be interpreted as 'if p is true, then q is true', or, more briskly, 'if p then q' or 'p implies q'. In these forms it is natural to call p the *hypothesis* and q the *conclusion.* Like the mathematical use of 'or' (when I say 'either it's Monday or it's three o'clock', you have to remember that it might be three o'clock on Monday), this is difficult to get a proper handle on, because it doesn't quite match ordinary language. If we interpret the sentence

if pigs had wings they'd fly

in terms of 'material implication', p becomes the sentence 'pigs have wings', q becomes 'pigs fly', $\neg p$ becomes 'pigs don't have wings' and $p \to q$ becomes 'either pigs fly or they don't have wings', which I maintain is true. As long as you stick to truth tables, and don't try to think about what they mean, it's not too difficult. For instance, we can build the table

p	q	r	$q \to r$	$p \to (q \to r)$	$p \wedge q$	$(p \wedge q) \to r$
T	T	T	T	T	T	T
T	T	F	F	F	T	F
T	F	T	T	T	F	T
T	F	F	T	T	F	T
F	T	T	T	T	F	T
F	T	F	F	T	F	T
F	F	T	T	T	F	T
F	F	F	T	T	F	T

We see that the column for $q \to r$ has a T unless q has a T and r has an F, the column for $p \to (q \to r)$ has a T unless p has a T and $q \to r$ has an F, and the column for $(p \wedge q) \to r$ has a T unless $p \wedge q$ has a T and r has an F, that is, when p, q and r have T, T, F respectively. Observe that

$$p \to (q \to r) \equiv (p \wedge q) \to r;$$

this is one of the fundamental rules of 'natural deduction', which we shall be coming to sometime.

Version of 26.1.01

Contrapositives A particular feature of $\to$ is the way in which it can be turned around: $p \to q$ is logically equivalent to $(\neg q) \to (\neg p)$. To see this, look at the truth table:

p	q	$p \to q$	$\neg q$	$\neg p$	$(\neg q) \to (\neg p)$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

Here $(\neg q) \to (\neg p)$ is the **contrapositive** of $p \to q$. This is one of the commonest manipulations of logic. If I say

whenever it rains I get wet

that amounts to the same thing as saying

if I'm dry it can't be raining.

Remember: to get the contrapositive, we swap the hypothesis p and the conclusion q, and negate both.

Of course we can also take the contrapositive of $(\neg q) \to (\neg p)$. But this is just

$$(\neg(\neg p)) \to (\neg(\neg q)),$$

and since $p \equiv \neg(\neg p)$ and $q \equiv \neg(\neg q)$ we're effectively back where we started.

Contrapositives are especially attractive when one of the original parts p and q is in a negative form. For instance, the contrapositive of $p \to (\neg q)$ is $(\neg(\neg q)) \to (\neg p)$, which is logically equivalent to $q \to (\neg p)$. While the contrapositive of $(\neg p) \to q$ is $(\neg q) \to (\neg(\neg p))$, which is logically equivalent to $(\neg q) \to p$.

I cannot overemphasize the importance of the skill of finding contrapositives. I would almost go so far as to say that whenever you see a sentence 'if … then …' anywhere, you should turn it round just to see if it still makes sense.

Example Consider the sentence

'If n is prime, then, if $n > 2$, n is odd.'

This is an 'if … then …' with an 'if … then …' nested inside. If we take

p to represent the statement 'n is prime',

q to represent the statement '$n > 2$',

r to represent the statement 'n is odd'

then our original statement becomes

$$p \to (q \to r).$$

Now we can replace either implication by its contrapositive, so we get

$$p \to ((\neg r) \to (\neg q)), \quad (\neg(q \to r)) \to (\neg p), \quad (\neg((\neg r) \to (\neg q))) \to (\neg p)$$

as statements logically equivalent to the first. In words, the first of these becomes

'If n is prime, then, if n is even, $n \leq 2$'

(I am translating '$\neg$(n is odd)' into 'n is even', and '$\neg(n > 2)$' into '$n \leq 2$'.) Ordinary English can't really cope with '$\neg(q \to r)$, but of course this has another form, if we remember that '$q \to r$' is an abbreviation for '$r \vee (\neg q)$', because then we get

$$\neg(q \to r) \equiv \neg(r \vee (\neg q)) \equiv (\neg r) \wedge (\neg(\neg q)) \equiv (\neg r) \wedge q.$$

So '$(\neg(q \to r)) \to (\neg p)$' becomes '$((\neg r) \wedge q) \to (\neg p)$', which translates into

'If n is even and $n > 2$, then n is not prime'.

Bi-implication A further connective of great importance is $\leftrightarrow$, **bi-implication** or **biconditional**, defined by the truth table

	p	q	$p \leftrightarrow q$
	T	T	T
	T	F	F
	F	T	F
Bi-implication	F	F	T

that is, $p \leftrightarrow q$ is T if p and q have the *same* truth value, whether T or F, and $p \leftrightarrow q$ is F if p and q have *different* truth values. It's called 'bi-implication' because it's logically equivalent to $(p \to q) \wedge (q \to p)$:

p	q	$p \leftrightarrow q$	$p \to q$	$q \to p$	$(p \to q) \wedge (q \to p)$
T	T	T	T	T	T
T	F	F	F	T	F
F	T	F	T	F	F
F	F	T	T	T	T

The idea is that $\leftrightarrow$ should correspond to the mathematicians' 'if and only if'; '$2x$ and $2y$ are equal if, and only if, x and y are equal' becomes '$(2x = 2y) \leftrightarrow (x = y)$'.

Since

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p),$$

and since both $p \to q$ and $q \to p$ are logically equivalent to their contrapositives, we get

$$\begin{aligned} p \leftrightarrow q &\equiv (p \to q) \wedge (q \to p) \\ &\equiv ((\neg q) \to (\neg p)) \wedge ((\neg p) \to (\neg q)) \\ &\equiv (\neg p) \leftrightarrow (\neg q). \end{aligned}$$

So, for instance, the sentence

a prime is odd if and only if it is greater than or equal to 3

is logically equivalent to the sentence

a prime is even if and only if it is less than 3.

Tautologies

Consider the expression

$$p \vee (p \to q).$$

If we draw up a truth table for this, we get

p	q	$p \to q$	$p \vee (p \to q)$
T	T	T	T
T	F	F	T
F	T	T	T
F	F	T	T

with a column of T's. So $p \vee (p \to q)$ is *always* true, regardless of what p and q are. (This is peculiar. Try an example: take p to be 'it's Tuesday' and q to be 'we're in Belgium'. Then $p \vee (p \to q)$ becomes '*either* it's Tuesday *or* if it's Tuesday we're in Belgium'. Most of us avoid saying things like this. But remember that our ultimate aim is to teach a computer to sort out our mistakes, and it's got to be able to cope with silly sentences as well as sensible ones.) Anyway, sentences (sensible or not) which are always true, in this sense, are called **tautologies**. We've already had a couple. The simplest, in fact, is $\top$, the 'empty tautology', which is always T because we say it is. Another is $p \vee (\neg p)$ (or $q \vee (\neg q)$, of course).

Now suppose we take a logically equivalent pair of expressions – e.g., $p \vee (q \wedge r)$ and $(p \vee q) \wedge (q \vee r)$ – and consider the compound $(p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (q \vee r))$. Then we get a table

p	q	r	$p \vee (q \wedge r)$	$(p \vee q) \wedge (p \vee r)$	$(p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (q \vee r))$
T	T	T	T	T	T
T	T	F	T	T	T
T	F	T	T	T	T
T	F	F	T	T	T
F	T	T	T	T	T
F	T	F	F	F	T
F	F	T	F	F	T
F	F	F	F	F	T

Since the columns for $p \vee (q \wedge r)$ and $(p \vee q) \wedge (p \vee r)$ match exactly, and since the bi- implication $(p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (q \vee r))$ is T exactly when the two halves match, it's always T. This is a general rule: two expressions ϕ and ψ are logically equivalent exactly when $\phi \leftrightarrow \psi$ is a tautology.

Version of 26.1.01

Natural Deduction

Let me return for a moment to the question: what is this course for? We are trying to develop methods for proving that programs work. (Actually you need exactly the same ideas to prove that chips are correctly designed. The only difference is that it's harder to debug a chip than a program, so it's even more important to get it right first time.) The statement that 'a program works' can always be expressed as a compound expression with terms like 'if, at the nth step, `lo` $<$ `high` then, at the $(n+1)$st step, `lo` $\leq$ `hi`'. Each of these we can hope to check, one by one, and then, with a mixture of the manipulations treated so far, some more to come later this term, and a good few proofs by induction, assemble them into a truth table with a nice big T at the appropriate place in the column for 'the program works'. The problem is that it's going to be rather a big truth table. Remember that if we have k 'atomic' propositions $p_1, \ldots, p_k$ to work with, then we shall have 2^k lines in the table. Since any ordinary C program is going to have at least one proposition corresponding to each statement in the program, any program with enough lines in it to be a problem is going to lead to truth tables which are completely unmanageable, even by machine. I haven't given you any truth table with more than 16 lines in it to do by hand. With a fast PC you could imagine going up to 2^{20} lines, corresponding to 20 atomic propositions; but 2^{100} lines isn't on, and never will be; and we are already using programs with a million statements in them. So we have to find methods other than brute force. And this is where logical argument comes in.

I suppose people have been arguing logically ever since there have been people. But the systematic arrangement of logical arguments in such a way that they can be properly checked was one of the inventions of the ancient Greeks sometime between 500 and 300 B.C. They didn't use the formal language which we have now (that's mostly a twentieth-century development) but all the essential ideas are there. You divide the problem into little bits, take them one at a time and make sure they don't get up afterwards. Most of the little bits will be of the form

if p then q

where p and q are compound propositions of some kind. Typically p is of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_k$, so that we are looking at expressions of the form

$$(p_1 \wedge \ldots \wedge p_k) \to q.$$

We slowly build up a list of these and then bring them together to form the theorems we are trying to prove.

A logical argument, therefore, will be a vertical list

p_1 (premise)
p_2 (premise)
...
therefore ϕ_1 (intermediate assertion)
therefore ϕ_2 (intermediate assertion)
...
therefore ψ (conclusion)

in which the first few sentences $p_1, \ldots$ will be **premises**, things we believe we can check directly from our knowledge of the world ('the sun rises in the east', 'this computer has 2^{20} bytes of RAM'), and $\phi_1, \ldots$ will be things we can prove from these, culminating in the final assertion ψ we were trying to prove. The fundamental rule will be that every line, after the initial list of premises, will be deducible from the lines *above* it, without reference to the lines *below*.

Now what does it mean to say that a proposition is *deducible* from some string of propositions? The idea is that ϕ should be deducible from $\phi_1, \ldots, \phi_n$ if ϕ is true whenever $\phi_1, \ldots, \phi_n$ are. We'd better have some examples.

example 1 Consider the argument

If Socrates is a man, he is mortal.
Socrates is a man.
Therefore Socrates is mortal.

The form of this is

$p \to q$
p
therefore q,

where p stands for 'Socrates is a man' and q stands for 'Socrates is mortal'. Now look at the truth table for $\to$:

p	q	$p \to q$	
T	T	T	$<$
T	F	F	
F	T	T	
F	F	T	

We are supposing that both p and $p \to q$ are true. If we look at the table, we see that only in the top row do p and $p \to q$ both have `T`'s. If we delete any row in which either of the premises has an `F`, we are left with only the top row, marked $<$ in the table. So this is the only row which can happen in the real world. Since, in this row, q also has a `T`, q must be true in the real world, and the argument is valid.

example 2 Consider the argument

$2 < 3$.
$3 < 4$.
If $2 < 3$ and $3 < 4$ then $2 < 4$.
Therefore $2 < 4$.

What is the structure of this argument? The first two lines are simple assertions p ('$2 < 3$') and q ('$3 < 4$'). The third line is a compound proposition in the form '$(p \wedge q) \to r$', where r stands for '$2 < 4$', and the final line is the assertion r. If we dress this up as a truth table, we get

p	q	r	$p \wedge q$	$(p \wedge q) \to r$	
T	T	T	T	T	$<$
T	T	F	T	F	
T	F	T	F	T	
T	F	F	F	T	
F	T	T	F	T	
F	T	F	F	T	
F	F	T	F	T	
F	F	F	F	T	

The claim is that the final line is a consequence of the first three, which we are supposed to believe for other reasons. Now the first three lines of the argument correspond to the first, second and fifth columns of the truth table. So we believe that the only relevant lines of the truth table are those in which p, q and $(p \wedge q) \to r$ all have `T`'s; that is, the top line. None of the other lines can be relevant to the real world. And if we look at the top line, then we see a `T` in the column for r; so, in the real world, r must be true.

example 3 Consider the argument

Either John is older than Peter, or they are the same age, or Peter is older than John.
John and Peter are not the same age.
Therefore either John is older than Peter or Peter is older than John.

Here the form of the argument is

$p \vee q \vee r$
$\neg q$
therefore $p \vee r$,

where p is 'John is older than Peter', q is 'John is the same age as Peter' and r is 'Peter is older than John'. (Remember that I allow myself to write $p \vee q \vee r$ to mean either, or both, of $p \vee (q \vee r)$ or $(p \vee q) \vee r$.) Now the corresponding truth table is

p	q	r	$p \vee q \vee r$	$\neg q$	$p \vee r$	
T	T	T	T	F	T	
T	T	F	T	F	T	
T	F	T	T	T	T	$<$
T	F	F	T	T	T	
F	T	T	T	F	T	$<$
F	T	F	T	F	F	
F	F	T	T	T	T	$<$
F	F	F	F	T	F	

Since we are assuming that $\neg q$ and $p \vee q \vee r$ are both T, we need look only at the third, fourth and seventh lines; in all of these, $p \vee r$ has a T, so the argument is valid.

An invalid argument We had better have an example of an argument which *isn't* valid. By far the most dangerous is the following type. Consider the argument

If he were clever, he'd be rich,
he isn't clever,
therefore he isn't rich.

Writing p for 'he's clever' and q for 'he's rich', this becomes

$p \to q$
$\neg p$
therefore $\neg q$.

In the truth table we have

p	q	$p \to q$	$\neg p$	$\neg q$
T	T	T	F	F
T	F	F	F	T
F	T	T	T	F* $<$
F	F	T	T	T $<$

Looking at the premise '$p \to q$', we delete the second line; looking at the premise '$\neg p$', we delete the top two lines; so the bottom two lines are both allowed by the premises. But we see that in the third line the conclusion $\neg q$ is F. So the argument is **not** valid. If p is F and q is T, both the premises are T but the conclusion is F. In words: when we say 'if he were clever, he'd be rich', we are saying nothing at all about what would happen if he isn't clever, and it is entirely possible that he is rich and stupid.

Valid arguments and true conclusions

For a lecture or two now I've been going on about 'validity' of arguments, or rather of argument forms. It's very important to remember that the validity of an argument is not the same thing as the truth of a conclusion. We can be lucky by accident in lots of ways. For example:

Roses are red,
violets are blue,
therefore cabbages are green.

I hope this seems absurd. But the absurdity is not in my botany or my colour vision; it's in the shape of the argument, which is

p
q
therefore r.

The conclusion simply isn't related to the premises, and as far as the logic goes I might just as well have said

There are seven days in the week,
there are twelve months in the year,
therefore the moon is made of green cheese.

A *valid* argument from *true* premises must lead to a *true* conclusion; but *invalid* arguments can also blunder their way to true conclusions. Again, we can perfectly well have a valid argument leading to a false conclusion, if the premises are shaky:

if it's Friday today, it's Sunday tomorrow,
it's Friday today,
therefore it's Sunday tomorrow;

the argument form is Modus Ponens and is absolutely sound, but the major premise is false, so the conclusion is unreliable.

Standard forms of argument The truth-table method can cope, in principle, with arbitrarily complicated arguments. We need to be able to formalize the sentences we're dealing with in the language of the propositional calculus. Having done so, all we have to do is write out the truth table, look at the lines in which all the premises have T's, and check that the conclusion has a T in each of these lines. In practice, this is tiresome to do by hand if there are more than four or five atomic propositions or more than half a dozen premises involved, and impractical by machine on an argument of any real complexity. We have to learn to break arguments down into standard forms which we can use more or less without thinking. So for the next few lectures I will try to discuss some of these forms.

Modus Ponens This is the classic argument form

p
$p \to q$
therefore q

used in example 1 above.

Modus Tollens A very similar one is

$p \to q$
$\neg q$
therefore $\neg p$.

(For instance: 'if he were clever, he'd be rich; he isn't rich; therefore he isn't clever'.) We can easily do this directly, from the table

p	q	$p \to q$	$\neg q$	$\neg p$
T	T	T	F	F
T	F	F	T	F
F	T	T	F	T
F	F	T	T	T $<$

observing that the only row in which both $p \to q$ and $\neg q$ have T's is the bottom one, and in that row $\neg p$ has a T. But we can also do it by putting in an extra stage:

$p \to q$
$\neg q$
therefore $(\neg q) \to (\neg p)$
therefore $\neg p$.

In this form, we say first that, because $(\neg q) \to (\neg p)$ is logically equivalent to $p \to q$, we can deduce the third line from the first (since of course any line of the truth table with a T for $p \to q$ will also have a T for $(\neg q) \to (\neg p)$) and then deduce the bottom line from the second and third, using Modus Ponens.

For this kind of compound argument, we *must* put in comments! preferably in colour. Remember that the idea here is that people should be able to check our arguments, and they haven't a hope if we don't say what we're doing. So we might have something like

1.	$p \to q$	(premise)
2.	$\neg q$	(premise)
3.	$(\neg q) \to (\neg p)$	logically equivalent to (1)

4. $\neg p$ Modus Ponens from (3) and (2).

Disjunctive Addition
These are the forms

p
therefore $p \vee q$

and

q
therefore $p \vee q$.

('It's Friday, therefore it's either Friday or Saturday'.)

Conjunctive Simplification
Again, this comes in two forms of the same idea:

$p \wedge q$
therefore p

and

$p \wedge q$
therefore q.

('It's 10 o'clock on Friday, therefore it's 10 o'clock'). Note that in both of these we are in some sense discarding information. If we replace 'p' by '$p \vee q$', then we are forgetting that we know that p is true by itself. But quite often this is the way to apply some fact that we've learnt without bothering to go into details. Suppose we want to prove that

1999 isn't a leap year.

Well, we have to ask someone to tell us about leap years, and they might say 'it's a bit complicated, but if the year ends in 1, 3, 5, 7 or 9 it isn't a leap year'. So then we can argue

1. 1999 ends in a 9 (premise)
2. (1999 ends in a 9)$\vee$(1999 ends in a 7)$\vee$(1999 ends in a 5)$\vee$(1999 ends in a 3)$\vee$(1999 ends in a 1) (disjunctive addition from (1))
3. ((1999 ends in a 9)$\vee$(1999 ends in a 7)$\vee$(1999 ends in a 5)$\vee$(1999 ends in a 3)$\vee$(1999 ends in a 1))$\rightarrow$1999 is not a leap year (premise)
4. 1999 isn't a leap year (Modus Ponens from (3) and (2)).

Similarly, in the argument

It's Monday 15 February, so there's a CC112 lecture

we might use the form

1. (It's Monday)$\wedge$(it's February 15) (premise),
2. (it's Monday)$\rightarrow$(there's a CC112 lecture) (premise),
3. therefore it's Monday (conjunctive simplification from (1)),
4. therefore there's a CC112 lecture (Modus Ponens from (2) and (3)).

Human beings don't waste much time on this sort of thing unless someone is ordering them to. But if you want a machine to follow you, you have to explain points like this.

Disjunctive Syllogism Yet another idea which takes two forms:

$p \vee q$
$\neg q$
therefore p

and

$p \vee q$
$\neg p$
therefore q

If you are told that there are two possibilities, and you know that one of them can't happen, then you are left with the other.

Hypothetical Syllogism Here we are coming to something rather more substantial. The formal expression is

$p \to q$
$q \to r$
therefore $p \to r$.

Let's check this from the truth table:

p	q	r	$p \to q$	$q \to r$	$p \to r$	
T	T	T	T	T	T	$<$
T	T	F	T	F	F	
T	F	T	F	T	T	
T	F	F	F	T	F	
F	T	T	T	T	T	$<$
F	T	F	T	F	T	
F	F	T	T	T	T	$<$
F	F	F	T	T	T	$<$

so that the rows in which both $p \to q$ and $q \to r$ have `T`'s are the first, fifth, seventh and eighth, in all of which $p \to r$ also has a `T`.

This is one of the commonest forms of argument. It enables us to assemble chains, as in

if `goodorder` runs correctly, then `partition` will run correctly
if `partition` runs correctly, then `quicksort` will run correctly
therefore, if `goodorder` runs correctly, `quicksort` will run correctly.

What is more, the chains can go on indefinitely; we can argue in the form

1. $p \to q$ (premise)
2. $q \to r$ (premise)
3. $r \to s$ (premise)
4. $q \to s$ hypothetical syllogism from (2) and (3)
5. $p \to s$ hypothetical syllogism from (1) and (4).

Conjunctive Addition We can always argue as follows:

p
q
therefore $p \wedge q$.

What this means (together with Conjunctive Simplification) is that a list $p_1, \ldots, p_r$ of premises can always be interchanged with a single premise $p_1 \wedge p_2 \wedge \ldots \wedge p_r$ on one line. But the whole point of what we are doing at the moment is to break arguments into manageable fragments, so we assemble propositions only when we have some good reason to. For instance

1. It's February 1999 (premise).
2. If it's February 1999 and the date is divisible by 7, then it's a Sunday (premise).
3. It's the 14th of the month (premise).
4. If it's the 14th of the month, the date is divisible by 7 (premise).
5. Therefore the date is divisible by 7 (Modus Ponens from (4) and (3)).
6. Therefore (it's February 1999)$\wedge$(the date is divisible by 7) (conjunctive addition of (1) and (5)).
7. Therefore it's a Sunday (Modus Ponens from (2) and (6)).

(Note how ignorant logicians pretend to be. The statement 'if it's the 14th of the month, the date is divisible by 7' has to be brought in explicitly as a premise. Once again: we are devising arguments which a machine can be taught to check. If we want to use arithmetic facts, then we shall have to make sure the machine knows them too.)

Division into Cases

Consider the form

$p \vee q$
$p \to r$
$q \to r$
therefore r.

The truth table for this is

p	q	r	$p \vee q$	$p \to r$	$q \to r$	
T	T	T	T	T	T	$<$
T	T	F	T	F	F	
T	F	T	T	T	T	$<$
T	F	F	T	F	T	
F	T	T	T	T	T	$<$
F	T	F	T	T	F	
F	F	T	F	T	T	
F	F	F	F	T	T	

Only the first, third and fifth lines are relevant, and on all of these r is T.

We use this form of argument when we have two different lines we can follow. In a chess game we might say

I have to move either my king or my pawn
if I move my king I shall lose
if I move my pawn I shall lose
therefore I am going to lose.

Contradiction or **Reductio ad Absurdum** We shall have to return to this later, but in its simplest form it looks like this:

$(\neg p) \to \bot$
therefore p.

(Remember that $\bot$ is the 'false' proposition whose only truth value is F.) The truth table looks like

p	$\bot$	$\neg p$	$(\neg p) \to \bot$	
T	F	F	T	$<$
F	F	T	F	

The nine argument forms above (Modus Ponens, Modus Tollens, disjunctive addition, conjunctive simplification, conjunctive addition, disjunctive syllogism, hypothetical syllogism, division into cases, contradiction) can be thought of as 'elementary', not in the sense that they are 'easy' but in the sense that, like the chemical elements, they are components from which we can build more elaborate structures. There are plenty of other forms which are equally useful and have an equal right to be listed here.

The next thing it is worth doing is to list some further techniques which often come in useful.

Tautology We may introduce any tautology into an argument without regard to what has gone before. (Because a tautology is an expression which has truth value T on *every* line; in particular, it has truth value T on every line allowed by the premises; so it can be regarded as a consequence of *any* set of premises!) The most useful tautologies are actually the simplest: expressions of the form $\phi \vee (\neg\phi)$. This has to be T, because if ϕ is T then of course $\phi \vee (\neg\phi)$ is T, while if ϕ is F then $\neg\phi$ is T so again $\phi \vee (\neg\phi)$ is T. This is the principle of the **excluded middle**.

A particularly important type of argument using this principle is the following:

1. $p \to r$ premise
2. $(\neg p) \to r$ premise
3. $p \vee (\neg p)$ excluded middle
4. r division into cases, from (3), (1) and (2).

In the chess-playing example above, we might say

If I move my king I shall lose
if I don't move my king I shall lose
therefore I am going to lose.

Logical equivalences If ϕ appears in an argument, and $\psi \equiv \phi$, then we may introduce ψ at any later stage in the argument. (For ψ and ϕ have exactly the same truth tables, so if ϕ has a T in any line allowed by the premises, then ψ also will.)

Substitution If we have a valid argument with atomic propositions $p_1, \ldots, p_n$, and $\phi_1, \ldots, \phi_n$ are any expressions of the propositional calculus, then we can obtain another valid argument by replacing each p_i by ϕ_i wherever it appears in the original argument. Note that we have to change the *premises* as well as the *conclusions* here, so that in order to use this method, we shall have to check that the new premises are still valid. But, for instance, we can be sure that

$$\begin{array}{l} (p \to q) \to (r \to s) \\ p \to q \\ \text{therefore } r \to s \end{array}$$

is a valid argument form, because it's got by substituting the expressions $p \to q$ and $r \to s$ for the variables in Modus Ponens.

Conditional arguments If we have a valid argument of the form

$$\begin{array}{l} \phi_1 \\ \phi_2 \\ \ldots \\ \phi_n \\ \therefore\ \phi, \end{array}$$

then for any expression ψ the argument

$$\begin{array}{l} \psi \to \phi_1 \\ \psi \to \phi_2 \\ \ldots \\ \psi \to \phi_n \\ \therefore\ \psi \to \psi, \end{array}$$

will also be valid. (Once again, we shall be changing the premises as well as the conclusions when we make this transformation.)

Validation by extended argument I give a couple of examples of arguments which can be validated by combining the methods described above.

example Consider the argument form

$$\begin{array}{l} p \to (\neg p) \\ \text{therefore } \neg p. \end{array}$$

Of course this is very easy to do directly from a truth table:

p	$\neg p$	$p \to (\neg p)$	
T	F	F	
F	T	T	<

But it can also be done indirectly, as follows:

1.	$p \to (\neg p)$	premise
2.	$(\neg p) \to (\neg p)$	tautology
3.	$p \vee (\neg p)$	tautology
4.	$\neg p$	division into cases, from (3), (1) and (2).

example For a more interesting example, consider the following:

$$\begin{array}{l} (p \to r) \vee (p \to s) \\ (p \wedge r) \to q \\ s \to q \\ \text{therefore } p \to q. \end{array}$$

Here a truth table would have to have sixteen lines and up to ten columns (for p, q, r, $p \to r$, $p \to s$, $(p \to r) \vee (p \to s)$, $p \wedge r$, $(p \wedge r) \to q$, $s \to q$ and $p \to q$), so it's worth looking for alternative strategies if we want to do it by hand. One possible route uses the following steps. We need to know a couple of standard logical equivalences:

$$(p \to r) \vee (p \to s) \equiv p \to (r \vee s)$$

(this is part of question 4 on Problem Sheet 3), and

$$(p \wedge r) \to q \equiv p \to (r \to q)$$

(I gave this in the paragraph on 'material implication'). We also need to know that the following argument form is valid:

q
therefore $p \to q$.

There are lots of routes to this, but with only two variables the truth table is surely as efficient as any:

p	q	$p \to q$	
T	T	T	<
T	F	F	
F	T	T	<
F	F	T	

Or we can say

1.	q	premise
2.	$(\neg p) \vee q$	disjunctive addition
3.	$p \to q$	logically equivalent to (2).

Let's call this argument form 'weakening by a condition'. Now, equipped with these, we have

1.	$(p \to r) \vee (p \to s)$	premise
2.	$(p \wedge r) \to q$	premise
3.	$s \to q$	premise
4.	$p \to (r \vee s)$	logically equivalent to (1)
5.	$p \to (r \to q)$	logically equivalent to (2)
6.	$p \to (s \to q)$	weakening by a condition, applied to (3)
7.	$p \to q$	division into cases, conditional on p, from (4), (5) and (6).

The trouble with this approach is that you have to know which way to go! The idea of an argument as written out above is that you ought to be able to follow it line-by-line. If you don't happen to know the logical equivalences used in lines 4 and 5, you do know how to verify them (and the truth tables involved are reasonably small, in particular, much smaller than the 16×10 table you would need for brute-force validation of the main argument); while the trick in line 6 is in fact a standard 'elementary' form which didn't happen to make it into the list of nine forms I gave. So, using the comment lines, you should be able to see that the whole argument here is valid. But it gives no clue how you might be able to do it on your own.

Validation by intelligent consideration of cases I have so far described two types of validation: (i) exhaustive analysis by truth table, guaranteed to work in theory, but often far too big to be done even with the assistance of a machine; (ii) checking through an extended argument, which (if the argument is well designed) is manageable enough, but only if someone has shown you the way. For many arguments of intermediate size, however, we can find our own way through if we use our brains. As an example, I return to the same question dealt with above. The argument form is

$(p \to r) \vee (p \to s)$
$(p \wedge r) \to q$
$s \to q$
therefore $p \to q$.

As I've already said, the full truth table for this has 16 lines and 10 columns, which is getting daunting. But of course we don't need the whole truth table. What we want to know is

is there any row in which every premise is T but the conclusion is F?

because if there is such a row then the argument is invalid, while if there is no such row then the argument is valid. So let us try to understand what a row would have to look like if the premises

$$(p \to r) \vee (p \to s), \quad (p \wedge r) \to q, \quad s \to q$$

are all T but the conclusion $p \to q$ is F. Now in any such row we already know that p is T and q is F, because that's the only way in which $p \to q$ can be F. Next, because $s \to q$ is T but q is F, s must be F. Because p

is T and s is F, $p \to s$ is F. Because $(p \to r) \vee (p \to s)$ is T, while $p \to s$ is F, $p \to r$ is T. Because p and $p \to r$ are T, r is T. Because p and r are both T, $p \wedge r$ is T. But now we have $p \wedge r$ T and q F, so the premise $(p \wedge r) \to q$ is F; oops!

Thus there *isn't* any such row, and the argument is valid.

I hope that you can follow the proof I've just set out. Of course this is another 'extended argument'; I'm taking you along a path in which each step is supposed to be easy once you've been shown, but there's no promise that the path as a whole is easy to find. But I think in practice many people can find paths of this kind. The point is that we know just what we're looking for – a row in a truth table with certain properties – and each sentence in the argument amounts to saying a little more about what such a row would have to look like; starting from the truth value of $p \to q$, we deduce, in turn, the truth values of p, q, s, $p \to s$, $p \to r$, r, $p \wedge r$ and $(p \wedge r) \to q$. The point about these steps is that each question is an obvious one to ask; the only non-obvious thing is the order in which we should ask them. But the idea is that we don't spend very much time on any question; if we can't see the answer almost at once, we try something else. And this technique works in a great many cases.

The paradox of logic If you look at the last validation I gave, you'll see the word 'because' appearing no fewer than six times. This is a sign that I am *using* one logical argument to *validate* another logical argument. This is what happens as soon as we come to any non-trivial bit of logic: we have to use logic to talk about logic. We have to lift ourselves by pulling on our own bootlaces. The best we can claim for the process is that the bits of logic we are using are somehow more certainly true than the bits of logic we're validating. This is a real problem, and not just a matter of the way I've arranged these lectures.

Contradiction again

Many logical arguments involve arguments-within-arguments. One of the standard ways of doing this is the following. Suppose that we can't decide whether a proposition p is true. In this case, we can hunt for consequences of p *and* for consequences of $\neg p$, and get a collection of facts

$$p \to q_1, \dots, p \to q_m,$$

$$(\neg p) \to r_1, \dots, (\neg p) \to r_n.$$

Now, if we are lucky, one of these q_i or r_j we turn up in our search will turn out to be something we know is false. Suppose, for instance, that we find that $(\neg p) \to r$, where we know that r is false, that is, that $\neg r$ is true. Then we can argue as follows.

1. $(\neg p) \to r$ by previous argument
2. $\neg r$ by previous argument
3. p Modus Tollens from (1) and (2).

Similarly, if we can prove that $p \to q$, where q is false, then we have

1. $p \to q$ by previous argument
2. $\neg q$ by previous argument
3. $\neg p$ Modus Tollens from (1) and (2).

The thing you *must* remember is that

– you can sometimes prove p by looking at consequences of $\neg p$

– you can sometimes prove $\neg p$ by looking at consequences of p.

But you can *never* prove p by looking at its consequences. If you want to know whether an animal is a cow, you might be able to show that

if it's a cow, it has four legs,

if it's a cow, it has horns.

So you count the legs and look for horns. I don't say exactly that this is silly; but it can't prove that you've got a cow rather than a goat. It can only prove that you *haven't* got a cow. To prove that you *have* got a cow, you need to look at what the consequences of its *not* being a cow are.

Quite often the consequence of p (or of $\neg p$) which we know to be false is false because it's logically impossible, that is, is logically equivalent to $\perp$; that's the form of 'contradiction' which I listed as an elementary argument form. Another common pattern is that we find that we can prove

$$p \to (\neg p).$$

In this case we deduce $\neg p$, as in one of the examples above.

Validation again 'Proofs by contradiction' can be used for any type of logical analysis; in particular, they can be useful in validations. Suppose we have an argument form

$$\begin{array}{l} \phi_1 \\ \phi_2 \\ \dots \\ \phi_n \\ \therefore\ \psi, \end{array}$$

and wish to validate it. One method is to look at the string

$$\phi_1,\ \phi_2, \dots, \phi_n,\ \neg\psi$$

consisting of the premises of the argument and the *negation* of the conclusion, and see what we can deduce from this list. If we can find a valid argument

$$\begin{array}{l} \phi_1 \\ \dots \\ \phi_n \\ \neg\psi \\ \therefore\ \theta_1 \\ \therefore\ \theta_2 \\ \dots \\ \therefore\ \theta_m \\ \therefore\ \bot, \end{array}$$

then we can argue as follows. On any line of the truth table on which $\phi_1, \dots, \phi_n$ are all T but ψ is F, then all the premises $\phi_1, \dots, \phi_n, \neg\psi$ of *this* argument will be T, so all the conclusions $\theta_1, \dots, \theta_m, \bot$ of this argument will be T; but $\bot$ is never T, so there's no such line. Accordingly, on any line on which $\phi_1, \dots, \phi_n$ are all T, ψ is also T, and the original argument was valid.

The example again This is actually very close to the 'intelligent' validation given above. If we take our argument

$$\begin{array}{l} (p \to r) \lor (p \to s) \\ (p \land r) \to q \\ s \to q \\ \text{therefore } p \to q, \end{array}$$

negate the conclusion, and argue from the consequent list, we can get

1.		$(p \to r) \lor (p \to s)$	
2.		$(p \land r) \to q$	
3.		$s \to q$	
4.		$\neg(p \to q)$	
5.	$\therefore$	$p \land (\neg q)$	logically equivalent to (4)
6.	$\therefore$	$\neg q$	conjunctive simplification from (5)
7.	$\therefore$	$\neg s$	Modus Tollens from (3) and (6)
8.	$\therefore$	p	conjunctive simplification from (6)
9.	$\therefore$	$p \land (\neg s)$	conjunctive addition from (8) and (7)
10.	$\therefore$	$\neg(p \to s)$	logically equivalent to (9)
11.	$\therefore$	$p \to r$	disjunctive syllogism from (1) and (10)
12.	$\therefore$	r	Modus Ponens from (11) and (8)
13.	$\therefore$	$p \land r$	conjuntive addition from (8) and (12)
14.	$\therefore$	q	Modus Ponens from (2) and (13)
15.	$\therefore$	$q \land (\neg q)$	conjunctive addition from (14) and (6)
.	$\therefore$	$\bot$	logically equivalent to (15).

So once again the original argument was valid.

Of course this is rather long, and in particular is longer than the seven-line validation given a couple of lectures ago. But I think it's easier to find, because it deals only with very simple propositions; after the original premises, no line has more than two atomic propositions nor more than one pair of brackets. So you have some chance of finding it by following the rule that you won't look at anything complicated, but you will check all simple possibilities.

Example: Knights and Knaves

A traditional type of logical puzzle uses the Island of Knights and Knaves. On this island, everyone is either a Knight, and always tells the truth, or a Knave, and always lies. In most of the stories, they know which is which, but you don't. They are nearly always peaceful and talkative, but you sometimes have to listen very carefully to be sure you know which question they are answering. Now:

> On the Island of Knights and Knaves, you meet two persons, who speak as follows:
>
> *Amos* Barnabas is a knight.
>
> *Barnabas* Amos and I are of different types.

What can we deduce from this?

We know that

if Amos is a knight, then Barnabas is a knight,

if Amos is a knave, then Barnabas is a knave.

From this we can already deduce that Amos and Barnabas are of the same type; therefore Barnabas is lying and both are knaves.

Formally, the argument goes as follows. If p stands for 'Amos is a knight' and q for 'Barnabas is a knight', then we have premises

$$p \leftrightarrow q, \quad q \leftrightarrow (p \leftrightarrow (\neg q)),$$

and our conclusion is

$$(\neg p) \wedge (\neg q).$$

But this is easy, because we have the truth table

p	q	$p \leftrightarrow q$	$p \leftrightarrow (\neg q)$	$q \leftrightarrow (p \leftrightarrow (\neg q))$	$(\neg p) \wedge (\neg q)$	
T	T	T	F	F	F	
T	F	F	T	F	F	
F	T	F	T	T	F	
F	F	T	F	T	T	<

Valid arguments and tautologies

There is a general result which will be useful later for theoretical reasons. An argument form

ϕ_1
$\ldots$
ϕ_n
therefore ψ

is valid iff the expression

$$(\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n) \rightarrow \psi$$

is a tautology. To see this, consider the truth table. The argument form is valid iff ψ is T on every line on which $\phi_1, \ldots, \phi_n$ are all T. But these are just the lines on which the conjunction $\phi_1 \wedge \ldots \wedge \phi_n$ is T. So the argument is valid iff on every line we *either* have ψ T *or* we have $\phi_1 \wedge \ldots \wedge \phi_n$ F; but this just means that $(\phi_1 \wedge \ldots \wedge \phi_n) \rightarrow \psi$ must be T on every line, that is, is a tautology.

Version of 26.1.01

Predicate Calculus

We come now to the third and final part of this course. If you look at the sentences considered in the first two parts, they are all straightforward assertions like '$2 \leq 3$', 'if Amos is a knight, Barnabas is not a knave' and so on. But many of the things we were looking at last term had phrases like 'for every n', 'there is some x'. Sentences of this form are the subject of the **predicate calculus**.

The essence of these new sentences is that

(i) they contain *variables*, generally labelled x or n or A to give a hint of what kind of object they refer to, but not normally specified to the point that you know exactly which number or set is being considered,

(ii) they contain *quantifiers*, either informal ('for every set A', 'whenever n is an integer', 'we can find a real number x') or formal (using the symbols $\forall$ or $\exists$) introducing the variables.

So instead of the argument

if Socrates is a man, Socrates is mortal,
Socrates is a man,
therefore Socrates is mortal,

we can say

for every x, if x is a man, x is mortal,
Socrates is a man,
therefore Socrates is mortal,

or, in less formal language,

all men are mortal,
Socrates is a man,
therefore Socrates is mortal.

I am going to try to explain the rules of these new sentences, and to incorporate them into the structure of the propositional calculus and the rules of natural deduction we have already seen.

Universal statements are of the form

$$\text{`}\forall\, x,\ Q(x)\text{'}, \qquad \text{`for every } x,\ Q(x)\text{'}.$$

The idea is that the proposition $\forall\, x,\ Q(x)$ will be T if the proposition $Q(a)$ is T for every a. If you like, the sentence $\forall\, x,\ Q(x)$ is logically equivalent to a conjunction

$$Q(a) \wedge Q(b) \wedge Q(c) \wedge \ldots$$

But this way of looking at it, though perhaps useful to begin with, is very dangerous if you don't have an extremely clear notion of what sort of a, b, c to look at. If there are infinitely many possibilities for x, then we seem to have an infinitely long sentence; and if there are uncountably many possibilities, even an infinitely long sentence will miss most of them out. So part of what I want to do here is to set out rules which don't depend on any particular listing of the relevant x's.

Actually there is a question about whether we need to try to think about truly universal sentences $\forall\, x,\ Q(x)$. Every universal sentence relevant to computer science, and nearly all the sentences of interest to mathematicians, have some restrictions on x built into them. Ordinarily we have sentences like

$$\text{for every natural number } n,\ n^2 \geq n,$$

or

$$\forall\, n \in \mathbb{N},\ n^2 \geq n,$$

in which we are told at the start that the variable is a particular type of object, or belongs to a particular set. So from now on I'll regularly mention on these, since it's generally easier to form a mental picture of what's happening, even though the formulae come out a few symbols longer.

Existential statements are of the form

$$\text{`}\exists\, x,\ Q(x)\text{'}, \qquad \text{`there is some } x \text{ such that } Q(x)\text{'}.$$

Here, the proposition $\exists\, x,\ Q(x)$ will be T if the proposition $Q(a)$ is T for some a. If you like, the sentence $\exists\, x,\ Q(x)$ is logically equivalent to the disjunction

$$Q(a) \vee Q(b) \vee Q(c) \vee \dots$$

where a, $b, \dots$ run through the possible things which x could be. Once again, this is dangerous if you don't know exactly what's going on. And once again, we do usually have a definite notion of what kind of x we're looking for, so our sentences are normally of the form

$$\exists\, x \in \mathbb{R},\ x^2 = 2,$$

with some set declared as the domain of the quantifier $\exists$.[1]

Note that *every* 'for all' or 'there is a' must be attached to some variable; and in a complete sentence, every variable should be introduced by a quantifier. If I just write

$$x \neq 2 \text{ or } x = 3$$

you are left wondering whether I mean

$$\text{for every } x, \text{ either } x \neq 2 \text{ or } x = 3$$

(starting with a universal quantifier) or

$$\text{there is some } x \text{ such that } x \neq 2 \text{ or } x = 3$$

(starting with an existential quantifier).

Negation The first rule to establish concerning these new sentences is the form which *negations* take. Recall that

$$\neg(p \wedge q) \equiv (\neg p) \vee (\neg q).$$

If we interpret $\forall\, x,\ Q(x)$ as a conjunction

$$Q(a) \wedge Q(b) \wedge \dots,$$

then it becomes natural to say that $\neg(\forall\, x,\ Q(x))$ ought to be logically equivalent to the disjunction

$$(\neg Q(a)) \vee (\neg Q(b)) \vee \dots,$$

which corresponds to the existential statement

$$\exists\, x,\ \neg Q(x).$$

In English, sort of: the negation of the sentence 'for every x, $Q(x)$' is logically equivalent to the sentence 'there is some x such that $\neg Q(x)$'. The negation of the sentence

all cows are brown

is equivalent to

there is a cow which isn't brown.

Similarly,

$$\neg(\exists\, x,\ Q(x))$$

is logically equivalent to

$$\forall\, x\, \neg Q(x),$$

just as

$$\neg(p \vee q)$$

is logically equivalent to

$$(\neg p) \wedge (\neg q).$$

The rule is:

> to negate any sentence starting with a quantifier $\forall$ or $\exists$, change the quantifier to the other one and negate the rest of the sentence.

[1] A new word here: $\forall$ and $\exists$ are **quantifiers**.

This works equally well for restricted quantifiers. The negation of the sentence

$$\forall\, x \in \mathbb{R},\ x^2 \geq 0$$

is the sentence

$$\exists\, x \in \mathbb{R},\ x^2 < 0.$$

You may have noticed that I haven't been talking about truth tables here. One of the very important philosophical differences between the predicate calculus and the propositional calculus is that all the logical implications of the propositional calculus, and all the corresponding rules of natural deduction, can be supported by truth tables, which may be uncomfortably large but are finite. The predicate calculus doesn't deal with finite truth tables, and the rules we use are based on analogies like the one above (treating a universal statement as a conjunction and an existential statement as a disjunction). I am not going to discuss such philosophical issues in this course. But it means that there is at least a possibility that there might be alternative logical rules for the predicate calculus; 'intuitionism' is one such alternative approach which has attracted computer scientists.

Universal conditional statements A very common kind of universal statement is of the form

$$\text{for every } x, \text{ if } P(x) \text{ then } Q(x).$$

In fact any sentence with a restricted quantifier can be thought of in this way: '$\forall\, n \in \mathbb{N},\ n^2 \geq n$' can be interpreted as '$\forall\, n$, if $n \in \mathbb{N}$ then $n^2 \geq n$'. The negation of such a sentence is, or is equivalent to,

$$\text{there is some } x \text{ such that } \neg(P(x) \to Q(x)).$$

Now applying the ordinary rule that

$$\neg(p \to q) \equiv \neg(q \vee (\neg p)) \equiv (\neg q) \wedge p,$$

we can transform this into

$$\text{there is some } x \text{ such that } P(x) \text{ and } \neg Q(x).$$

Vacuous truth Consider the sentences

All Prince Charles' daughters are brunettes,

All Prince Charles' daughters are blondes.

Are these true or false? If we look at their negations, we get

Prince Charles has a daughter who is not a brunette,

Prince Charles has a daughter who is not a blonde.

Now both of those (so far as I know) are false. On the principle that for *any* sentence p, exactly one of p and $\neg p$ must be true, we are forced to conclude that both the original sentences are true. So, putting them together, we get

All Prince Charles' daughters are both brunette and blonde.

This is yet another peculiarity of formal logic. If we have a universal statement

$$\forall\, x,\ P(x) \to Q(x)$$

where $P(x)$ is never true, then the statement is true; because

$$\forall\, x,\ P(x) \to Q(x)$$

means just that

$$\forall\, x,\ Q(x) \vee (\neg P(x)),$$

and since

$$\forall\, x,\ \neg P(x),$$

this is true. Similarly

$$\forall\, x \in \emptyset,\; Q(x)$$

is necessarily true for any grammatically correct predicate $Q(x)$. Since this tells us rather less about $Q(x)$ than we might have hoped to learn, it's helpful to remark that the sentence is 'vacuously' true; it's true because you can't catch any x's to check. Of course the sentence

All Prince Charles' daughters are both brunette and blonde

(if true) is telling us something, not about brunettes and blondes, but about Prince Charles.

Multiple quantifiers As soon as we let ourselves have quantifiers at all, we are likely to want more than one. (If we want to talk about `quicksort`, we can expect to have to deal with sentences like 'for all m, $n \in \mathbb{N}$, if `left` $= m$ and `right` $= n$, then for all k, $l \in \mathbb{N}$, if `lo` $= k$ and `hi` $= l$, ...'.) Typically we get sentences like

for every real number x there is a natural number n such that $x \leq n$,

or

$$\forall\, x \in \mathbb{R},\; \exists\, n \in \mathbb{N},\; x \leq n.$$

These give rise to some new problems. Many of these can be dealt with by rigorously writing in the brackets I've left out:

$$\forall\, x \in \mathbb{R},\; \big(\exists\, n \in \mathbb{N},\; x \leq n\big).$$

You have to remember which variable was introduced first, and be very careful not to shuffle them around without careful thought. For instance, it is true that for every real number x there is a natural number n such that $x \leq n$. But if we swap the quantifiers, we get

$$\exists\, n \in \mathbb{N},\; \big(\forall\, x \in \mathbb{R},\; x \leq n\big),$$

which is false. *Changing the order of quantifiers is dangerous.* Think of it as a game. In the sentence

$$\forall\, x \in \mathbb{R},\; \exists\, n \in \mathbb{N},\; x \leq n,$$

the x-player moves first and the n-player, having seen x, moves second; of course he can win. But in the sentence

$$\exists\, n \in \mathbb{N},\; \forall\, x \in \mathbb{R},\; x \leq n,$$

the n-player has to make a move which can defeat any possible reply by the x-player, and there's no such move. Changing the order of quantifiers is like changing the order of the moves in a game; it's liable to change the result.

Yet another way of thinking of the same thing: if you think of

$$\exists\, y,\; P(x, y)$$

as a shorthand for

$$P(x, \alpha) \vee P(x, \beta) \vee \dots,$$

then

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

becomes

$$(P(a, \alpha) \vee P(a, \beta) \vee \dots) \wedge (P(b, \alpha) \vee P(b, \beta) \vee \dots) \wedge \dots,$$

while

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

becomes

$$(P(a, \alpha) \wedge P(b, \alpha) \wedge \dots) \vee (P(a, \beta) \wedge P(b, \beta) \wedge \dots) \vee \dots.$$

Now even if x can only be either a or b, and y can only be either α or β, then we have expressions like

$$(p \vee q) \wedge (r \vee s), \quad (p \wedge r) \vee (q \wedge s),$$

and these are not logically equivalent; the second implies the first, but not the other way about.

Apart from this, a lot of the rules we've already had still work. If we want to negate a sentence with a string of quantifiers, for instance, we can do it as follows

$$\neg\big(\exists\, n \in \mathbb{N},\ (\forall\, x \in \mathbb{R},\ x \leq n)\big)$$

is logically equivalent to

$$\forall\, n \in \mathbb{N},\ \neg\big(\forall\, x \in \mathbb{R},\ x \leq n\big)$$

which is logically equivalent to

$$\forall\, n \in \mathbb{N},\ \big(\exists\, x \in \mathbb{R},\ \neg(x \leq n)\big)$$

which is equivalent to

$$\forall\, n \in \mathbb{N},\ \big(\exists\, x \in \mathbb{R},\ n < x\big).$$

You just work along the quantifiers in order, changing every $\forall$ into $\exists$ and every $\exists$ into $\forall$, and finishing up with a negation of the core sentence at the end. With a little practice this goes very smoothly, though I must say I still find that I like to write it out in full sometimes.

Formal translations In the propositional calculus 'translations' of English sentences into formal expressions are not trivial but at least can be done according to fairly well-defined rules, and two different versions of the same original are generally recognisable as such. In the predicate calculus it's a good deal harder. For instance, the sentence

'all flying pigs have wings'

can equally well be represented as

$$\forall\, x,\ FP(x) \to W(x),$$

where $FP(x)$ means 'x is a flying pig' and $W(x)$ means 'x has wings', or as

$$\forall\, x \in P,\ F(x) \to W(x),$$

where P is the set of pigs and $F(x)$ means 'x can fly'. When we come to multiple quantifiers, there may be a real problem in deciding what the original sentence means. For instance, the sentence

'you can fool all the people some of the time'

might mean either

'everybody can be fooled sometimes'

or

'there are times when you can fool everybody'.

The first is

$$\forall\, x \in P,\ \exists\, y \in T,\ F(x, y),$$

while the second is

$$\exists\, y \in T,\ \forall\, x \in P,\ F(x, y),$$

writing P for the set of people, T for the set of times, and $F(x, y)$ for 'you can fool x at time y'. Of course the second is much harder; you might easily be able to fool people one at a time, without ever being able to fool them all at once. I think myself that it's the second version which Lincoln had in mind.

There are also some difficulties with the way ordinary English handles negations. Consider the four sentences

(i) Smith can outrun every man on the team.

(ii) Smith can outrun any man on the team.

(iii) Smith cannot outrun every man on the team.

(iv) Smith cannot outrun any man on the team.[2]

Of these, (i) and (ii) are saying the same thing; they can both be represented as

$$\forall\, x \in T,\, P(S, x),$$

where T is the set of men in the team, S stands for Smith and $P(y, x)$ means 'y can outrun x'. But (iii) and (iv) are different: (iii) just says that (i) is false, that is,

$$\neg(\forall\, x \in T,\, P(S, x)),$$

or

$$\exists\, x \in T,\, \neg P(S, x),$$

'there is a man on the team who can keep up with Smith'. (iv), however, says something much stronger, that is, '*every* man on the team can keep up with Smith', or

$$\forall\, x \in T,\, \neg P(S, x).$$

Thus the particle 'not' has quite different effects in the two cases. You will see how this sort of problem makes computer translation so difficult!

Exchanging quantifiers of same type I said that you had to be awfully careful about changing the order of two quantifiers. $\forall\, x\, \exists\, y$ is hardly ever the same thing as $\exists\, y\, \forall\, x$. But if you have $\forall\, x\, \forall\, y$ things are much more comfortable, and you can safely move to $\forall\, y\, \forall\, x$ if you like it better. Similarly, $\exists\, x\, \exists\, y$ comes to the same thing as $\exists\, y\, \exists\, x$. You can get some idea of what's happening here by looking at the conjunctions and disjunctions involved. If we say

$$\text{for all } x,\ \big(\text{for all } y,\ Q(x, y)\big),$$

and interpret this as a conjunction

$$(\text{for all } y,\ Q(a, y)) \wedge (\text{for all } y,\ Q(b, y)) \wedge \dots,$$

and then as a conjunction

$$(Q(a, i) \wedge Q(a, j) \wedge \dots) \wedge (Q(b, i) \wedge Q(b, j) \wedge \dots) \wedge \dots,$$

the associative and commutative laws for $\wedge$ allow us to rearrange this as

$$(Q(a, i) \wedge Q(b, i) \wedge \dots) \wedge (Q(a, j) \wedge Q(b, j) \wedge \dots) \wedge \dots$$

corresponding to

$$\text{for all } y,\ \big(\text{for all } x,\ Q(x, y)\big),$$

In the same way, the equivalence of $\exists\, x\, \exists\, y$ and $\exists\, y\, \exists\, x$ is related to the associative and commutative laws for $\vee$. Whether this really the right way of looking at it I leave to the philosophers. But remember that this trick works only if the brackets are right! If you start from

$$\forall\, m \in \mathbb{Z}\ \big((\forall\, n \in \mathbb{N},\ m^2 \leq n^2) \to (m = 0)\big)$$

(which is true, because if $m^2 \leq n^2$ for every $n \in \mathbb{N}$ then $m^2 \leq 0$ so $m = 0$), and try to switch the quantifiers, you are in danger of getting

$$\forall\, n \in \mathbb{N}\big(\forall\, m \in \mathbb{Z},\, (m^2 \leq n^2) \to (m = 0)\big)$$

which is false (if you put $n = 1$, then it isn't true that $m^2 \leq n^2$ implies that $m = 0$). The problem is in the brackets $\big((\quad)\quad\big)$. If we change these to

$$\forall\, m \in \mathbb{Z}, \forall\, n \in \mathbb{N},\ (m^2 \leq n^2) \to (m = 0),$$

[2]This example is taken from 'Mathematical Logic', by W.V.O.Quine, one of the founders of the subject. You may notice that the language now seems very old-fashioned! If you can imagine that you are teaching at Harvard in 1930 it makes sense; students were always 'men' in those days, and (in America, at least) they were expected to take both running and 'teams' seriously. *Eheu fugaces!* But logic is eternal.

$$\forall\, n \in \mathbb{N},\, \forall\, m \in \mathbb{Z},\, (m^2 \leq n^2) \to (m = 0)$$

then the sentences are logically equivalent (and both are false; try $m = n = 1$).

So take care!

Arguments with quantified statements Just as with 'natural deduction' in the propositional calculus, there are valid argument forms for arguments with quantified statements. Just as before, there are infinitely many such valid forms. But the position now is much worse. In the propositional calculus, there is an accepted algorithm (validation by full truth table) to determine whether an argument form is valid. The actual computation may be beyond us for practical reasons, but in theoretical terms we know what to do; moreover, even in cases which are computationally intractable, brains and persistence and luck may lead us to a step-by-step validation we can check. But for mathematics as a whole there is no such accepted algorithm, and (in my view) 'Gödel's theorem',[3] as it's called, tells us that no acceptable algorithm will ever be found. So we really are faced here with something deeper than logic.

Nevertheless, there are in practice a large number of arguments which are accepted by the great majority of working mathematicians, who can mostly agree among themselves about which methods are valid and which aren't.

What we *can* sometimes say is that certain argument forms must be *invalid*. For instance the form

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

cannot possibly be valid, because

for every man there is a woman who is (or was) his mother

is true, while

there is a woman who is the mother of every man

is false.

This is a one-sided test: some arguments we know we must reject, because we have found a case in which they lead from true premises to a false conclusion. But it's not at all clear how we can be sure that an argument is valid. In the last hundred years, methods have been developed for validating arguments in the predicate calculus, but these belong to a different course (on 'model theory') and I shall not try to describe them here.

Various people have tried to give lists of the most important argument forms used in practical reasoning. These lists are even less convincing than the list of nine 'standard arguments' I produced for the propositional calculus. I will not try to do anything of the sort. Instead, I will proceed with 'case studies', looking at some valid arguments, translating them into argument forms, and seeking to classify these forms according to general principles.

Case study 1 Consider the argument

All men are mortal,
Socrates is a man,
therefore Socrates is mortal.

The argument form is

$\forall\, x,\, P(x) \to Q(x)$,
$P(a)$,
therefore $Q(a)$,

where $P(x)$ stands for 'x is a man', $Q(x)$ for 'x is mortal', and a represents Socrates. There is more than one way of looking at this, but one is the following. From

$$\forall\, x,\, P(x) \to Q(x)$$

we first deduce the special case

$$P(a) \to Q(a),$$

[3] K.Gödel, 1906-1978; http://history.math.csusb.edu/Mathematicians/Godel.html

and then we just have Modus Ponens:

$$\begin{array}{l} P(a) \to Q(a) \\ P(a) \\ \text{therefore } Q(a). \end{array}$$

So the new idea is that of 'taking a particular case'; the official word for this is **instantiation**. Starting from any sentence $\forall x,\ \phi(x)$ and any a, we have the valid argument form

$$\begin{array}{l} \forall\, x,\ \phi(x) \\ \text{therefore } \phi(a). \end{array}$$

Perhaps this really is the most fundamental of all the principles of the predicate calculus, since it goes to the heart of what $\forall$ means; if we really believe that $\phi(x)$ is true for every x, then surely we believe it of any particular value.

In terms of restricted quantifiers, instantiation takes a slightly different form:

$$\begin{array}{l} \forall\, x \in A,\ \phi(x) \\ a \in A \\ \text{therefore } \phi(a). \end{array}$$

(If our original universal statement was applicable only to members of A, we'd better check that $a \in A$ before applying it.)

In fact the 'Socrates syllogism' can be simplified into this form, if we interpret A as the set of men and $\phi(x)$ as 'x is mortal'.

Case study 2 Consider the three formulae

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

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

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

Can any of these be regarded as a consequence of one of the others? I have already remarked that the argument

$$\begin{array}{l} \forall\, x,\ \exists\, y,\ P(x,y), \\ \therefore\ \exists\, y,\ \forall\, x,\ P(x,y) \end{array}$$

is invalid ('every man has a mother' is true, but 'there is a woman who is the mother of every man' isn't). The same example shows that

$$\begin{array}{l} \forall\, x,\ \exists\, y,\ P(x,y), \\ \therefore\ \forall\, y,\ \exists\, x,\ P(x,y) \end{array}$$

is invalid, because

'every man has a mother'

is true, while

'every woman has a son'

is false. However, the argument

$$\begin{array}{l} \exists\, y,\ \forall\, x,\ P(x,y), \\ \therefore\ \forall\, x,\ \exists\, y,\ P(x,y) \end{array}$$

is valid; if the premise is true, then we have a single y which works for every x at once, and in particular works for any particular x.

Case study 3 Consider the theorem 'If A, B and C are sets, and $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$'. This is a good deal more complicated. The first thing to do is to get an *informal* proof written down. (Not one person in a thousand can write a formal proof without first getting an informal proof clear in their mind.) Here the informal proof might run

> '$A \subseteq B$ means just that every member of A belongs to B. So the hypothesis is that (i) every member of A belongs to B (ii) every member of B belongs to C. Now, in this case, if $x \in A$, we

must have $x \in B$, therefore $x \in C$. So every member of A belongs to C, which is what we mean by saying that $A \subseteq C$.'

Note that (as is very common in mathematical arguments) quite a lot of the argument is just a matter of unwrapping shorthand statements like '$A \subseteq B$' into extended forms like 'for every x, if x belongs to A then it belongs to B'; the point being that these are what can be expressed in the formulae of the predicate calculus, as '$\forall x, (x \in A) \to (x \in B)$'.

Now look at the way the argument is running. The central section is

if $x \in A$ then $x \in B$,
if $x \in B$ then $x \in C$,
therefore if $x \in A$ then $x \in C$.

This is just Hypothetical Syllogism:

$$\begin{array}{l} p \to q \\ q \to r \\ \therefore\ p \to r. \end{array}$$

So the idea seems to be: given that

$$\begin{array}{l} (x \in A) \to (x \in B) \\ (x \in B) \to (x \in C) \\ \therefore\ (x \in A) \to (x \in C) \end{array}$$

is a valid argument, we want to say that

$$\begin{array}{l} \forall x, (x \in A) \to (x \in B) \\ \forall x, (x \in B) \to (x \in C) \\ \therefore\ \forall x, (x \in A) \to (x \in C) \end{array}$$

is a valid argument.

This is a special case of **universal natural deduction**. Generally, if we have any valid argument

$$\begin{array}{l} \phi_1 \\ \phi_2 \\ \dots \\ \phi_n \\ \therefore\ \psi, \end{array}$$

then we can stick a 'for every x' at the beginning of each line and get another valid argument

$$\begin{array}{l} \forall x, \phi_1 \\ \forall x, \phi_2 \\ \dots \\ \forall x, \phi_n \\ \therefore\ \forall x, \psi. \end{array}$$

(Of course this is worth doing only when the variable x turns up in at least some of the formulae $\phi_1, \dots, \psi$.) What we are saying here is that since every premise is valid whatever x is, so must the conclusion be, because we can run the argument for each x separately. The relativised version is equally straightforward: if

$$\begin{array}{l} \phi_1 \\ \phi_2 \\ \dots \\ \phi_n \\ \therefore\ \psi, \end{array}$$

is valid, then for any set A the form

$$\begin{array}{l} \forall x \in A, \phi_1 \\ \forall x \in A, \phi_2 \\ \dots \\ \forall x \in A, \phi_n \\ \therefore\ \forall x \in A, \psi \end{array}$$

will be valid.

So the middle section of our proof becomes

$$\begin{array}{l} \forall\, x,\ (x \in A) \to (x \in B) \\ \forall\, x,\ (x \in B) \to (x \in C) \\ \therefore\ \forall\, x,\ (x \in A) \to (x \in C) \end{array}$$

which we recognize as 'universal hypothetical syllogism' and therefore valid. Using the standard abbreviations, we see that

$$\begin{array}{l} A \subseteq B \\ B \subseteq C \\ \therefore\ A \subseteq C \end{array}$$

is a valid argument form.

We are now very near the theorem we wanted. But not quite there. The theorem itself says

$$\forall A, \forall B, \forall C, ((A \subseteq B) \wedge (B \subseteq C)) \to (A \subseteq C).$$

We have somehow got to translate a claim about the validity of an argument form into a statement about sets. Of course practically all mathematical arguments work this way. Nearly all theorems are general assertions saying that all numbers, or all sets, or whatever, have such-and-such properties, and the proofs proceed by giving names to the objects involved and looking for arguments which will deal with them when they are thought of as specified individuals. One way of looking at the process is through what I will call the **tautology principle**: if we have an argument form

$$\begin{array}{l} \phi_1 \\ \phi_2 \\ \dots \\ \phi_n \\ \therefore\ \psi, \end{array}$$

then this is valid iff

$$(\phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n) \to \psi$$

is a tautology. Now I claim that, because

$$\begin{array}{l} A \subseteq B \\ B \subseteq C \\ \therefore\ A \subseteq C \end{array}$$

is a valid argument form, then

$$((A \subseteq B) \wedge (B \subseteq C)) \to (A \subseteq C)$$

is a tautology, that is, can be entered into any argument without other justification, that is, the argument form

$$\therefore\ ((A \subseteq B) \wedge (B \subseteq C)) \to (A \subseteq C)$$

in which there are *no* premises and this conclusion, is valid. Now applying Universal Natural Deduction to this argument we get

$$\therefore\ \forall\, A, \forall\, B, \forall\, C, ((A \subseteq B) \wedge (B \subseteq C)) \to (A \subseteq C)$$

as a valid argument; that is,

$$\forall\, A, \forall\, B, \forall\, C, ((A \subseteq B) \wedge (B \subseteq C)) \to (A \subseteq C)$$

is itself a tautology and necessarily true.

> **Warning!** I am using the word 'tautology' here in a dangerously elastic way. In particular, if you ever want to go farther into the theory of logical arguments, you will have to re-arrange many of the ideas here into different language. All I am trying to do at the moment is to identify some of the manipulations mathematicians use, and have found to be safe.

Case study 4 Consider the argument

if x is real and $x^3 < 0$, then $x^5 < 0$,
therefore there is a real number x such that $x^5 < 0$.

Is this a valid argument? The conclusion is certainly true; $(-1)^5 = -1 < 0$. But to say that the *argument* is valid is to say that any argument of that logical form, starting from a true premise, will lead to a true conclusion. Now the form of the argument is

$$\forall x \in A,\ P(x) \to Q(x),$$
$$\therefore\ \exists x \in A,\ Q(x),$$

where A is the set of real numbers, $P(x)$ stands for '$x^3 > 0$', and $Q(x)$ stands for '$x^5 > 0$'. And put like that it won't do at all. For a start, A could be the empty set, so that any sentence starting '$\forall x \in A, \ldots$' would be true, and any sentence starting '$\exists x \in A, \ldots$' would be false. Or, closer to the original expression (since we know very well that $\mathbb{R}$ is non-empty) we might have

if x is real and $x^4 < 0$, then $x^6 < 0$,
therefore there is a real number x such that $x^6 < 0$.

Here the premise is true but the conclusion is false; so the argument is untrustworthy.

Case study 5 Consider the argument form

$$\forall x \in A,\ \exists y \in A,\ P(x,y),$$
$$\forall x \in A,\ \forall y \in A,\ \forall z \in A,\ (P(x,z) \wedge P(y,z)) \to (x = y),$$
$$\therefore\ \forall y \in A,\ \exists x \in A,\ P(x,y).$$

Is this valid?

If you really can't guess the answer to a question like this, it can be very hard indeed. To *validate* the argument you will either have to find some more or less philosophical approach to make it 'obvious', or reduce it to one of the forms you have already seen. To *invalidate* it you will have to find some example in which the premises are true and the conclusion false, and you have very little idea where to look. What makes things much worse is not even knowing which of these you have to do!

Usually, faced with such a dilemma, the thing to do is to try the two approaches alternately. Just occasionally one finds that one's mind can engage a problem with parallel processing, tackling it from both sides at once, and seeing which side wins. But this rarely works with anything which requires a new idea. Normally, when you can see two different ways of tackling a problem, and don't know which to use, you should make a serious effort with one of them, and turn to the other when the first seems to be getting nowhere.

As it happens, the argument here is invalid, so any attempt to validate it is bound to fail, and I won't try to explore what a validation would look like if there were one. Instead, I will give just the approach which starts from the assumption that the argument is invalid and hunts for an example to demonstrate this.

The first step is to use a technique which we have already seen in 'natural deduction'. The argument is invalid if there is an example in which the premises are true but the conclusion is false, that is, the negation of the conclusion is true. So write down what the negation of the conclusion is:

$$\neg(\forall y \in A,\ \exists x \in A,\ P(x,y)).$$

Now we can move the $\neg$ past the $\forall$ and the $\exists$, as usual, to get

$$\exists y \in A,\ \forall x \in A,\ \neg P(x,y),$$

and we are hunting for an example in which this is true, together with the original premises

$$\forall x \in A,\ \exists y \in A,\ P(x,y),$$

$$\forall x \in A,\ \forall y \in A,\ \forall z \in A,\ (P(x,z) \wedge P(y,z)) \to (x = y).$$

At this point it is worth noting the following. For any statement $P(x,y)$ about two things, there is a corresponding relation R, where we write $x\,R\,y$ if $P(x,y)$ is true. If we re-write the hypotheses in this language, we have

there is some $y \in A$ such that $x \not R\, y$ for every $x \in A$,

for every $x \in A$ there is some $y \in A$ such that $x\,R\,y$,

for all x, y and z in A, if $x\,R\,z$ and $y\,R\,z$ then $x = y$.

This is worth while because it gives us a chance to use ideas from CC111. For instance, the hypothesis

$$\text{for every } x \in A \text{ there is some } y \in A \text{ such that } x\,R\,y$$

was actually given a name there; it says that A is included in the 'domain' of the relation R. Spotting this kind of correspondence may not solve a problem, but it will help a great deal in telling you what sort of ideas may be useful.

Now if we think of R as a relation, we can try to look at its incidence matrix M_R. What does M_R have to look like? Consider the negated conclusion:

$$\text{there is some } y \in A \text{ such that } x \not R\, y \text{ for every } x \in A.$$

What this says is that there is some column of M_R in which every entry is 0. (Remember that the (x, y)-entry of M_R lies in the x-row and the y-column, and is 1 if $x\,R\,y$, 0 if $x \not R\, y$.) Since there is at least one column of zeroes, we might as well put it at the beginning:

$$\begin{pmatrix} 0 & . & . & . & . \\ 0 & . & . & . & . \\ 0 & . & . & . & . \\ 0 & . & . & . & . \\ 0 & . & . & . & . \end{pmatrix}$$

Next, the premise

$$\text{for every } x \in A \text{ there is some } y \in A \text{ such that } x\,R\,y$$

declares that in every *row* there is at least one 1. In particular, there's a 1 in the top row; let's put it in the first available spot:

$$\begin{pmatrix} 0 & 1 & . & . & . \\ 0 & . & . & . & . \\ 0 & . & . & . & . \\ 0 & . & . & . & . \\ 0 & . & . & . & . \end{pmatrix}$$

If we now look at the other premise

$$\text{for all } x,\, y \text{ and } z \text{ in } A, \text{ if } x\,R\,z \text{ and } y\,R\,z \text{ then } x = y,$$

this says that in any column (corresponding to a fixed z) there can be at most one 1. So we can fill in some more of our matrix: the rest of the second column must be zeroes:

$$\begin{pmatrix} 0 & 1 & . & . & . \\ 0 & 0 & . & . & . \\ 0 & 0 & . & . & . \\ 0 & 0 & . & . & . \\ 0 & 0 & . & . & . \end{pmatrix}$$

Returning to the premise

$$\text{for every } x \in A \text{ there is some } y \in A \text{ such that } x\,R\,y,$$

we see that there has to be a 1 in the second row; put it in the first blank slot:

$$\begin{pmatrix} 0 & 1 & . & . & . \\ 0 & 0 & 1 & . & . \\ 0 & 0 & . & . & . \\ 0 & 0 & . & . & . \\ 0 & 0 & . & . & . \end{pmatrix}.$$

Remember that no column can have more than one 1, so that the rest of the third column's entries are 0,

$$\begin{pmatrix} 0 & 1 & 0 & . & . \\ 0 & 0 & 1 & . & . \\ 0 & 0 & 0 & . & . \\ 0 & 0 & 0 & . & . \\ 0 & 0 & 0 & . & . \end{pmatrix},$$

that there must be a 1 in the third row,

$$\begin{pmatrix} 0 & 1 & 0 & . & . \\ 0 & 0 & 1 & . & . \\ 0 & 0 & 0 & 1 & . \\ 0 & 0 & 0 & . & . \\ 0 & 0 & 0 & . & . \end{pmatrix},$$

the rest of the fourth column must be zeroes,

$$\begin{pmatrix} 0 & 1 & 0 & 0 & . \\ 0 & 0 & 1 & 0 & . \\ 0 & 0 & 0 & 1 & . \\ 0 & 0 & 0 & 0 & . \\ 0 & 0 & 0 & 0 & . \end{pmatrix},$$

there must be a 1 in the fourth row,

$$\begin{pmatrix} 0 & 1 & 0 & 0 & . \\ 0 & 0 & 1 & 0 & . \\ 0 & 0 & 0 & 1 & . \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & . \end{pmatrix},$$

and the rest of the fifth column must be zeroes, and look at what we've got:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Is this a relation of the type we're looking for? It's got a blank column, and no column has more than one 1 in it; that is, the two requirements

$$\text{there is some } y \in A \text{ such that } x \not R\, y \text{ for every } x \in A,$$

$$\text{for all } x,\ y \text{ and } z \text{ in } A, \text{ if } x\,R\,z \text{ and } y\,R\,z \text{ then } x = y$$

are satisfied. But there's also a blank row, so the requirement

$$\text{for every } x \in A \text{ there is some } y \in A \text{ such that } x\,R\,y$$

hasn't been achieved. But remember at this point that we're not limited to a 5×5 matrix. If we give ourselves an extra row and column we can get a 1 into the fifth row, as follows:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Of course this just shifts the problem to the sixth row, and we have to do it again:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

At this point we need a flash of light. *What happens if we go on for ever?* Why not have an *infinite* set A? I know that in CC111 I never tried to draw an incidence matrix for a relation on an infinite set, but that's no reason to refuse to do it now. Imagine that we simply carried on:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}.$$

The pattern is clear enough: a diagonal line of 1's and the rest 0's.

With the pattern established, we can try to construct an actual formula to match it. Since the set A is infinite, let's make it the simplest infinite set, $\mathbb{N}$. Then we can annotate our matrix:

$$\begin{array}{c|c} & \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & \dots \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ \vdots \end{matrix} & \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \end{array}.$$

What relation does this correspond to? We have $0\,R\,1$, $1\,R\,2$, $2\,R\,3$, $\dots$; that is, $m\,R\,(m+1)$ for every $m \in \mathbb{N}$, while $m \not R\, n$ for all other m and n. Going back to the predicate $P(x,y)$, what this seems to mean is that $P(x,y)$ if, and only if, $y = x+1$. In this case we have

$$\text{there is some } y \in A \text{ such that } (x,y) \notin P \text{ for every } x \in A,$$

because $0 \neq x+1$ for every $x \in \mathbb{N}$,

$$\text{for all } x,\ y \text{ and } z \text{ in } A, \text{ if } (x,z) \in P \text{ and } (y,z) \in P \text{ then } x = y$$

because if $z = x+1 = y+1$ then $x = y$, and

$$\text{for every } x \in A \text{ there is some } y \in A \text{ such that } (x,y) \in P$$

because we can take $y = x+1$ every time. Thus

$$\forall\, x \in A,\ \exists\, y \in A,\ P(x,y)$$

and

$$\forall\, x \in A,\ \forall\, y \in A,\ \forall\, z \in A,\ (P(x,z) \wedge P(y,z)) \to (x = y)$$

are true, but (because of the empty first column)

$$\forall\, y \in A,\ \exists\, x \in A,\ P(x,y)$$

is false, and the argument is invalid.

I have written this out at such length to try to show you that there is a route to the answer which depends on determination rather than inspiration. You have to be absolutely clear about what you are looking for, but building examples of this kind is rather like writing programs: once you have the specification clear and you have mastered the language, you can hope to get there in the end. The other feature of the argument is the fact that there is no *finite* set A which will do. Any example will necessarily involve an infinite set. However the set $\mathbb{N}$, the simplest infinite set, is enough; and as it happens this is true for *any* example in the predicate calculus[4].

> I ought to remark, for the sake of anyone who might wish to go farther with these ideas, that the formula '$x = y$' which appears in the formal statements above gives a good deal of trouble if you look at it too hard. However there are no real traps in it, so it will do no harm if you take it as 'obvious' what it means.

Case study 6 Here is another theorem from elementary set theory:

'For any relations R and S, $(R \circ S)^{-1} = S^{-1} \circ R^{-1}$'.

What would a proof of this in the language of the predicate calculus look like?

First, as usual, let's try to set out a mathematical argument without forcing ourselves into rigid formulae. First, we have to remember what $\circ$ and $^{-1}$ mean in this context:

$$x(R \circ S)y \text{ iff there is a } z \text{ such that } x\,R\,z \text{ and } z\,S\,y,$$

$$x\,R^{-1}\,y \text{ iff } y\,R\,x.$$

But there's also the question of what we mean when we say that two relations are equal: we say that $R = S$ if

$$R = S \text{ if } x\,R\,y \text{ just when } x\,S\,y.$$

So a mathematical proof might go something like this:

$$\begin{aligned}
x(R \circ S)^{-1}y &\iff y(R \circ S)x \\
&\iff \text{there is some } z \text{ such that } y\,R\,z \text{ and } z\,S\,x \\
&\iff \text{there is some } z \text{ such that } z\,R^{-1}\,y \text{ and } x\,S^{-1}\,z \\
&\iff \text{there is some } z \text{ such that } x\,S^{-1}\,z \text{ and } z\,R^{-1}\,y \\
&\iff x(S^{-1} \circ R^{-1})y.
\end{aligned}$$

Now all the translations of the basic terms are all expressible in the predicate calculus, so we have some hope of sorting things out according to the rules of this course.

After translation, the formula is something like

$$\forall R, \forall S, (\mathcal{R}(R) \wedge \mathcal{R}(S)) \to ((R \circ S)^{-1} = S^{-1} \circ R^{-1}),$$

where I write '$\mathcal{R}(R)$' to mean 'R is a relation'. Now this expands to

$$\forall R, \forall S, (\mathcal{R}(R) \wedge \mathcal{R}(S)) \to (\forall x, \forall y, x(R \circ S)^{-1}y \leftrightarrow x(S^{-1} \circ R^{-1})y),$$

which is itself shorthand for

$$\begin{aligned}
&\forall R, \forall S, (\mathcal{R}(R) \wedge \mathcal{R}(S)) \to (\forall x, \forall y, \\
&\qquad y(R \circ S)x \leftrightarrow (\exists z, x\,S^{-1}\,z \wedge z\,R^{-1}\,y)),
\end{aligned}$$

and expands again to

$$\begin{aligned}
&\forall R, \forall S, (\mathcal{R}(R) \wedge \mathcal{R}(S)) \to (\forall x, \forall y, \\
&\qquad (\exists z, y\,R\,z \wedge z\,S\,x) \leftrightarrow (\exists z, z\,S\,x \wedge y\,R\,z)).
\end{aligned}$$

Put like this it seems obvious. Surely the two things on either side of the $\leftrightarrow$, $\exists z, y\,R\,z \wedge z\,S\,x$ and $\exists z, z\,S\,x \wedge y\,R\,z$ are exactly the same? Well, not *exactly*; we are relying on the commutativity of $\wedge$, that is, the logical equivalence of $p \wedge q$ and $q \wedge p$. Now we have known for ages that

[4]This is a form of the **Löwenheim-Skolem theorem**, which is the first basic theorem in model theory.

$$(p \wedge q) \leftrightarrow (q \wedge p)$$

is a tautology, that is, is necessarily true whatever p and q happen to be. But the expression we have in the middle of the last formula is

$$(\exists\, z,\, P(z) \wedge Q(z)) \leftrightarrow (\exists\, z,\, Q(z) \wedge P(z)),$$

and we ought to check that this is equally universally true.

What in fact is true is that for any tautology

$$\phi(z) \rightarrow \psi(z)$$

of the predicate calculus, the expression

$$(\exists\, z,\, \phi(z)) \rightarrow (\exists\, z,\, \psi(z))$$

is also a tautology. For if $\psi(z)$ is necessarily true, for logical reasons, whenever $\phi(z)$ is true, and if there is some z such that $\phi(z)$ is true, then $\psi(z)$ will be true for that z, possibly among others. Similarly, if

$$\phi(z) \leftrightarrow \psi(z)$$

is a tautology, that is, if $\phi(z)$ is true exactly when $\psi(z)$ is true, then there will be some z making one true iff there is a z making the other true, and

$$(\exists\, z,\, \phi(z)) \leftrightarrow (\exists\, z, \psi(z))$$

will be a tautology. Applying this with $\phi(z)$ standing for $y\,R\,z \;\wedge\; z\,S\,x$ and $\psi(z)$ standing for $z\,S\,x \;\wedge\; y\,R\,z$, we see that the central formula

$$(\exists\, z,\, y\,R\,z \;\wedge\; z\,S\,x) \leftrightarrow (\exists\, z,\, z\,S\,x \;\wedge\; y\,R\,z)$$

is a tautology. Since $p \rightarrow q$ is true whenever q is true, we can be sure that

$$(\mathcal{R}(R) \wedge \mathcal{R}(S)) \rightarrow (\forall\, x, \forall\, y,\, (\exists\, z,\, y\,R\,z \;\wedge\; z\,S\,x) \leftrightarrow (\exists\, z,\, z\,S\,x \;\wedge\; y\,R\,z))$$

is also universally true, and therefore that

$$\begin{aligned}\forall\, R, \forall\, S, (\mathcal{R}(R) \wedge \mathcal{R}(S)) \rightarrow (\forall\, x, \forall\, y,& \\ (\exists\, z,\, y\,R\,z \;\wedge\; z\,S\,x) \leftrightarrow (\exists\, z,\, z\,S\,x \;\wedge\; y\,R\,z))&\end{aligned}$$

is universally true, that is, is a theorem.

Version of 26.1.01

Induction again At last we have a language capable of expressing the principles of induction. The First Principle is the argument form:

$$\begin{array}{l} P(0) \\ \forall\, n \in \mathbb{N},\; P(n) \rightarrow P(n+1) \\ \therefore\; \forall\, n \in \mathbb{N},\; P(n). \end{array}$$

While the Second Principle has two quantifiers in it:

$$\begin{array}{l} P(0) \\ \forall\, n \in \mathbb{N},\; \big(\forall\, i \in \mathbb{N},\; ((0 \le i) \wedge (i < n)) \rightarrow P(i)\big) \rightarrow P(n) \\ \therefore\; \forall\, n \in \mathbb{N},\; P(n). \end{array}$$

Watch the brackets in the second line! I am trying to say

if $P(i)$ is true whenever $0 \le i < n$, then $P(n)$ is true.

So the big brackets go round '$P(i)$ is true whenever $0 \le i < n$', and I have

$$\big(P(i) \text{ is true whenever } 0 \le i < n\big) \rightarrow P(n).$$

Now the 'whenever' signals a universal quantifier; we are saying something about every i, so it comes out as a $\forall\, i$ at the front:

$$\big(\forall\, i, \text{ if } 0 \le i < n \text{ then } P(i)\big) \rightarrow P(n).$$

Taking $0 \le i < n$ apart as a conjunction $(0 \le i) \wedge (i < n)$ and remembering to say explicitly that we only want to look at natural numbers i, I get the form

$$\big(\forall\, i \in \mathbb{N},\ ((0 \le i) \wedge (i < n)) \to P(i)\big) \to P(n)$$

which now has to have a '$\forall\, n$' put in front.

The 'second principle' especially seems to be hard to get to grips with; I think this really is because its logical form is more complex. In fact practising mathematicians use it quite as often as the first, and also use a couple of allied principles which I will now describe.

Peano's[5] principle of induction says

$$\forall A,\ \big((A \subseteq \mathbb{N}) \wedge (0 \in A) \wedge (\forall\, n,\ (n \in A) \to (n+1 \in A)\big) \to (A = \mathbb{N}).$$

I put it like this so that you can practice deciphering the formula for yourselves before reading further! But I had better give the translation as well. What it says is that

> If A is a subset of $\mathbb{N}$, containing 0, such that $n+1 \in A$ whenever $n \in A$, then A is the whole of $\mathbb{N}$.

Note that the expression '$A \subseteq \mathbb{N}$' is an abbreviation for '$\forall x,\ (x \in A) \to (x \in \mathbb{N})$'.

The well-ordering principle (for $\mathbb{N}$) says

$$\forall\, A,\ ((A \subseteq \mathbb{N}) \wedge (A \ne \emptyset)) \to (\exists\, n \in A, \forall\, m \in A,\ n \le m).$$

What this expression is saying is that if A is any non-empty subset of $\mathbb{N}$ then it has a least member, that is, there is an $n \in A$ such that $n \le m$ for every $m \in A$.

> (I have written '$A \ne \emptyset$' above as if it were an expression of the predicate calculus. Actually it is here an abbreviation for '$\exists\, n,\ n \in A$'.)

I think it is worth looking at the connexions between these four principles. If you think of them in the right ways, they all say very nearly the same thing.

(i) For instance, assuming the First Principle, you can prove the Second Principle, as follows. Suppose that P satisfies the conditions of the Second Principle, that is,

> $P(0)$ is true,
>
> whenever $n \in \mathbb{N}$ is such that $P(i)$ is true whenever $0 \le i < n$, then $P(n)$ is true.

In this case, write $Q(n)$ for

$$P(i) \text{ is true whenever } 0 \le i \le n.$$

Then $Q(0)$ is true, because the only i such that $0 \le i \le 0$ is 0 itself, and we are supposing that $P(0)$ is true. Next, if $Q(n)$ is true, then $P(i)$ is true whenever $0 \le i \le n$, because that's what $Q(n)$ says; but this means that $P(i)$ is true whenever $0 \le i < n+1$, so that $P(n+1)$ is true, and in fact $P(i)$ is true whenever $0 \le i \le n+1$, and $Q(n+1)$ is true. Thus $Q(n)$ implies $Q(n+1)$, for every $n \in \mathbb{N}$, and (by the first principle of induction) $Q(n)$ is true for every $n \in \mathbb{N}$. But as $Q(n) \to P(n)$, $P(n)$ is true for every $n \in \mathbb{N}$, which is what we wanted to know.

(ii) Next, assuming the First Principle, you can prove Peano's principle, as follows. Given $A \subseteq \mathbb{N}$ such that $0 \in A$ and $n+1 \in A$ whenever $n \in A$, write $P(n)$ for '$n \in A$'. Then our hypotheses tell us that $P(0)$ is true and that $P(n+1)$ is true whenever $P(n)$ is true, so $P(n)$ is true for every $n \in \mathbb{N}$, and $A = \mathbb{N}$. Since A is arbitrary, Peano's principle is true.

(iii) And assuming the Second Principle you can prove the well-ordering principle, as follows. Suppose that $A \subseteq \mathbb{N}$ is any non-empty set. Write $P(n)$ for '$n \in \mathbb{N} \setminus A$' (that is, '$(n \in \mathbb{N}) \wedge (\neg(n \in A))$'). Because A is non-empty, we know that '$P(n)$ for every $n \in \mathbb{N}$' is false. So one of the hypotheses of the Second Principle is false. (a) If the hypothesis '$P(0)$' is false, then $0 \in A$, and of course $0 \le m$ for every $m \in A$. (b) If the hypothesis 'whenever $P(i)$ for $0 \le i < n$, then $P(n)$' is false, then there must be some $n \in \mathbb{N}$ such that $P(i)$ for every $i < n$ but $\neg P(n)$; that is, $n \in A$ but $i \notin A$ whenever $i < n$. But this means that if $m \in A$ then $m \not< n$, that is, $n \le m$. Thus in either case (a) or case (b) we have an $n \in A$ such that $n \le m$ for every $m \in A$.

[5] G.Peano, 1858-1932; `http://history.math.csusb.edu/Mathematicians/Peano.html`

Second principle again It is worth noting that the expression of the Second Principle can be simplified. I have written it out as

$P(0)$
$\forall\, n \in \mathbb{N},\ \big(\forall\, i \in \mathbb{N},\ ((0 \le i) \wedge (i < n)) \to P(i)\big) \to P(n)$
$\therefore\ \forall\, n \in \mathbb{N},\ P(n).$

But of course the formula

$$\forall\, i \in \mathbb{N},\ ((0 \le i) \wedge (i < n)) \to P(i)$$

is clumsy, because if $i \in \mathbb{N}$ then we don't have to say separately that $0 \le i$, so we can replace it by

$$\forall\, i \in \mathbb{N},\ (i < n) \to P(i).$$

Next, the first hypothesis '$P(0)$' can simply be omitted. Because if we know that the second hypothesis

$$\forall\, n \in \mathbb{N},\ \big(\forall\, i \in \mathbb{N},\ (i < n) \to P(i)\big) \to P(n)$$

is true, then, in particular ('instantiating')

$$\big(\forall\, i \in \mathbb{N},\ (i < 0) \to P(i)\big) \to P(0).$$

But

$$\forall\, i \in \mathbb{N},\ (i < 0) \to P(i)$$

is necessarily `T`, because for any $i \in \mathbb{N}$, '$i < 0$' is `F` so '$(i < 0) \to P(i)$' is `T`. So we can argue

$\big(\forall\, i \in \mathbb{N},\ (i < 0) \to P(i)\big) \to P(0)$
$\forall\, i \in \mathbb{N},\ (i < 0) \to P(i)$
$\therefore\ P(0)$

by Modus Ponens.

Thus the Second Principle of Induction can be rewritten

$\forall\, n \in \mathbb{N},\ \big(\forall\, i \in \mathbb{N},\ (i < n) \to P(i)\big) \to P(n)$
$\therefore\ \forall\, n \in \mathbb{N},\ P(n).$

Proofs of correctness of programs Nearly all proofs of 'correctness' involve at least one appeal to an induction. The most fundamental reason is that when a program is running, then it will move through a variety of states, and you will have to describe the operation of the program in terms of a sequence $S_0, S_1, S_2, \ldots$ of states, where S_n is a description of the machine, including all its memory registers and any files it's referring to, after n operations. In order to prove anything about its behaviour, you will probably have to identify some general characteristic $P(S_n))$ of the states S_n, and prove, by induction on n, that $P(S_n)$ is true for every n. In a very large proportion of cases, there are some 'internal' inductions as well; the proof that $P(S_n) \to P(S_{n+1})$ is likely to involve an induction on the size of some variable in the program. This is especially true because you will probably want to use n as a count, not of the number of machine operations which have been performed, but of the number of lines of code which have been executed; and if some of those lines are subroutine calls, you will have some hard work to do in checking that they've done the right things.

As an example of the kind of process involved, I will return to the program `quicksort` which I introduced in the first lecture. The centre of this program is the subprogram `partition`:

```
static int partition(int left,int right,int qslist[],double num[])
  {int pivot = qslist[left]; int lo = left; int hi = right;
    while( lo < hi )
      {if( goodorder(pivot,qslist[hi],num)) --hi;
        else{qslist[lo]=qslist[hi]; ++lo; qslist[hi]=qslist[lo];}}
    qslist[lo] = pivot;
    return(lo);}
```

Remember what this is doing. We have a list of numbers in the array `qslist[]` which we are trying to sort. The idea is that at the end of the process they should be arranged in an order determined by the subprogram `goodorder` and the data array `num[]`, and that we count `qslist[i]`, `qslist[j]` as being in an acceptable order if `goodorder(qslist[i],qslist[j],num)` is `T`. The `partition` subprogram is supposed to split the section from `qslist[left]` to `qslist[right]` into two parts; the 'low' section, which will run from `qslist[left]` to `qslist[dp−1]`, and the 'high' section, which will run from `qslist[dp+1]` to `qslist[right]`, with a 'pivot' at `qslist[dp]` in the middle, and `dp` the 'returned value' of the subprogram. (`quicksort` will later sort each of these sections in turn, so that in the end the whole section from `left` to `right` has been sorted.)

Now during the execution of `partition`, quite a lot of things are being left strictly unchanged. The array `num`, for instance, isn't touched. (You can see from the program code that it isn't touched by `partition`. But since there's a call to `goodorder`, you have to look at the code for that to make sure that it also doesn't mess about with `num`.) Similarly, `left` and `right` don't move. Next, you can see what happens to `lo` and `hi`. At the beginning, `lo` is set equal to `left` and `hi` to `right`; so we start with `left` = `lo` < `hi` = `right`, at least if `left` < `right`. (What happens if `left` ≥ `right`? In this case, `lo` ≥ `high` at the start, so the loop `while (lo < hi)` never acts, and the subprogram closes having done nothing except waste a moment setting `pivot` equal to `qslist[left]`, `lo` equal to `left` and `qslist[lo]` equal to `pivot`, which is what it started as; if it matters, the returned value is `left`.)

Suppose, then, that we enter `partition` with `left` < `right`. What happens to `lo` and `hi` as the `while` loop runs? The loop has an `if ... else ...`, and on one side `hi` drops by 1, while on the other side `lo` rises by 1. So if at the beginning of the loop, `left` ≤ `lo` < `hi` ≤ `right`, then at the end of the loop we shall have `left` ≤ `lo` ≤ `hi` ≤ `right`; while if we approach the loop with `lo` ≥ `hi`, then we go to the end of the subprogram. Thus `lo` and `hi` stay between `left` and `right`; what is more, the difference `hi` − `lo` shrinks by 1 each time through, until it reaches 0 and we exit.

What this means is that we can set up a little proof by induction of the following fact:

> If we enter the `while` loop of `partition` with `left` ≤ `lo` ≤ `hi` ≤ `right` and `hi` − `lo` = n, then after n repetitions of the loop we shall leave the subprogram with `left` ≤ `lo` = `hi` ≤ `right`. Moreover, `left` ≤ `lo` ≤ `hi` ≤ `right` throughout the operation of the subprogram.

For this is certainly true if $n = 0$, and if it is true for $n = k$, then, starting from `left` ≤ `lo` ≤ `hi` ≤ `right` and `hi` = `lo`+$k + 1$, then after one run through the loop we shall have `left` ≤ `lo` ≤ `hi` ≤ `right` and `hi` = `lo`+k, so (by the inductive hypothesis) k further repetitions of the loop will lead us to exit in the approved manner.

Of course `lo` and `high` here are just bookkeepers; they're nothing to do with the real purpose of the subprogram, but they're essential as markers of where we are. But we do see from this at least that the program will terminate! which is surely the first thing to make sure of.

The next stage of the analysis must be to check that `qslist[]` is being dealt with correctly. The first step is I think to note that while `qslist[]` is certainly altered, it's altered only in certain ways. Specifically, the only things done during the loop are in the instructions

```
qslist[lo]=qslist[hi]; qslist[hi]=qslist[lo];
```

and since neither `lo` nor `hi` leaves the interval [`left`,`right`], `qslist[`i`]` is unchanged if either $i <$ `left` or $i >$ `right`.

In between `left` and `right`, of course, things are moved around. It's perhaps easier to see what the line

```
qslist[lo]=qslist[hi]; ++lo; qslist[hi]=qslist[lo];
```

does if we think of it as if it were the exactly equivalent code

```
qslist[lo]=qslist[hi]; qslist[hi]=qslist[lo+1]; ++lo;
```

This shifts the value `qslist[hi]` down to the location `qslist[lo]` and the value `qslist[lo+1]` up to the location `qslist[hi]`, before incrementing `lo`. Now the clever thing about this is that the *old* `qslist[lo]` is overwritten and the *new* `qslist[lo]` has been copied into `qslist[hi]`. The point is that `qslist[lo]` is treated as a vacant slot. It's not carrying anything we ever look at after the original definition of `pivot` as `qslist[left]`. One way to think of this is to regard `pivot` as the 'real' value of `qslist[lo]`, and the actual number stored in the location `qslist[lo]` as accidental junk. If we look at it this way, we can formulate an assertion

> after n executions of the `while` loop, if we replace `qslist[lo]` by `pivot`, then the numbers in the string `qslist[left]`, ..., `qslist[right]` are exactly what they were when we entered `partition`, but possibly in a different order.

Now this is something we can set out to prove by induction on n. It's obviously true when $n = 0$, because up to the first entry to the `while` loop we don't touch `qslist[]` at all. For the inductive step, we look at what happens as we pass through the `while` loop. It may be that we don't change `qslist[]`, so there's nothing more to check. If we do change `qslist[]`, it's by moving `qslist[hi]` to the old blank space `qslist[lo]`, moving `qslist[lo+1]` to `qslist[hi]`, and incrementing `lo`, so that the loose number `pivot` is correctly pointed at the new blank space. Since on exiting `partition` we put `pivot` back into place in `qslist[lo]`, we can say that

on exiting `partition` the numbers in the string `qslist[left]`, ..., `qslist[right]` are exactly what they were when we entered, but possibly in a different order.

Oof. Next: in what way is the 'new' order an improvement on the 'old' order? The target was to ensure that, for `left` $\le i <$ `dp`, `qslist[`i`]` and `qslist[dp]` should be in the right order, that is, that `goodorder(qslist[`i`],qslist[dp],num)` is `T`; it will make the rest of this argument much simpler if I write $i \preccurlyeq j$ to mean `goodorder(`i`,`j`,num)` is `T`(in the execution I gave of `quicksort`, this just means `num[`i`]` $\le$ `num[`j`]`). We also want to leave `partition` with `qslist[dp]` $\preccurlyeq$ `qslist[`i`]` whenever `dp` $< i \le$ `right`. Now `dp` is the 'returned value' of `partition`, that is, the final value of `lo`; and as the penultimate line of `partition` sets `qslist[lo]` to be `pivot`, the objective is that

on exiting `partition`,
`qslist[`i`]` $\preccurlyeq$ `pivot` whenever `left` $\le i <$ `dp`,
`pivot` $\preccurlyeq$ `qslist[`i`]` whenever `dp` $< i \le$ `right`.

How is this achieved? In fact what is done is that, at each entry to the `while` loop,

`qslist[`i`]` $\preccurlyeq$ `pivot` whenever `left` $\le i <$ `lo`,
`pivot` $\preccurlyeq$ `qslist[`i`]` whenever `hi` $< i \le$ `right`.

To see this, induce on the number of times we've gone through the loop. At the beginning, `lo` = `left` and `hi` = `right`, so there are no i such that `left` $\le i <$ `lo` or `hi` $< i \le$ `right`, and the requirement is vacuously satisfied. For the inductive step, we have to look at two cases. (a) If `pivot` $\preccurlyeq$ `qslist[hi]`, then we don't touch `qslist[]`, but we bring `hi` down by one. Since at the beginning `pivot` $\preccurlyeq$ `qslist[`i`]` whenever `hi` $< i \le$ `right`, by the inductive hypothesis, and also `pivot` $\preccurlyeq$ `qslist[hi]`, we actually have `pivot` $\preccurlyeq$ `qslist[`i`]` whenever `hi` $\le i \le$ `right`, and (after decreasing `hi`) `pivot` $\preccurlyeq$ `qslist[`i`]` whenever `hi` $< i \le$ `right`. On the other side, `qslist[`i`]` $\preccurlyeq$ `pivot` whenever `left` $\le i <$ `lo`, again by the inductive hypothesis, and this is unchanged by the action of the loop. (b) If `pivot` $\not\preccurlyeq$ `qslist[hi]`, then `qslist[hi]` $\preccurlyeq$ `pivot`. (This depends on the structure of `goodorder`, of course, and must be checked.) In this case, at the end of the loop, `qslist[lo`$-$`1]` is the old `qslist[hi]`, so $\preccurlyeq$ `pivot`. Thus since (by the inductive hypothesis) we had `qslist[`i`]` $\preccurlyeq$ `pivot` whenever i was less than the old `lo`, that is, less than the new `lo`-1, and these `qslist[`i`]` haven't been changed, we shall have, at the end of the loop, `qslist[`i`]` $\preccurlyeq$ `pivot` whenever `left` $\le i \le$ `lo`-1, that is, whenever `left` $\le i <$ `lo`. At the other end, `hi` hasn't moved (though of course `qslist[hi]` has been changed), and `pivot` $\preccurlyeq$ `qslist[`i`]` whenever `hi` $< i \le$ `right`, by the inductive hypothesis.

Thus this induction continues.

At the end of the operation, therefore, when `partition` has come to its end because `lo` = `hi`, we still have

`qslist[`i`]` $\preccurlyeq$ `pivot` whenever `left` $\le i <$ `lo`,
`pivot` $\preccurlyeq$ `qslist[`i`]` whenever `hi` $< i \le$ `right`;

and as we now set `dp` = `lo` = `hi`, we shall have, on exiting the program,

`qslist[`i`]` $\preccurlyeq$ `qslist[dp]` whenever `left` $\le i <$ `dp`,
`qslist[dp]` $\preccurlyeq$ `qslist[`i`]` whenever `dp` $< i \le$ `right`,

and `partition` has done its stuff.

(Of course we should remember to check what this looks like if we enter `partition` with `left` $\ge$ `right`. Once again, `partition` does nothing at all and returns `dp` = `left`. In this case, there are no values of i such that `left` $\le i <$ `dp` or `dp` $< i \le$ `right`, so the same formulae are vacuously satisfied.)

This more or less completes the analysis of the subprogram `partition`. There is still a corresponding analysis of `quicksort` itself to do! But this is easier; I sketched some of the ideas in Lecture 1.

CC112: Summary of Rules of Inference

Modus Ponens

$$\begin{array}{rc} & p \to q \\ & p \\ \therefore & q \end{array}$$

Modus Tollens

$$\begin{array}{rc} & p \to q \\ & \neg q \\ \therefore & \neg p \end{array}$$

Disjunctive addition

$$\begin{array}{rc} & p \\ \therefore & p \vee q \end{array}$$

Conjunctive simplification

$$\begin{array}{rc} & p \wedge q \\ \therefore & p \end{array}$$

Conjunctive addition

$$\begin{array}{rc} & p \\ & q \\ \therefore & p \wedge q \end{array}$$

Disjunctive syllogism

$$\begin{array}{rc} & p \vee q \\ & \neg q \\ \therefore & p \end{array}$$

Hypothetical syllogism

$$\begin{array}{rc} & p \to q \\ & q \to r \\ \therefore & p \to r \end{array}$$

Division into cases

$$\begin{array}{rc} & p \vee q \\ & p \to r \\ & q \to r \\ \therefore & r \end{array}$$

Proof by contradiction

$$\begin{array}{rc} & (\neg p) \to \perp \\ \therefore & p \end{array}$$

Tautology We may introduce any tautology into an argument without regard to what has gone before.

Substitution If we have a valid argument with atomic propositions $p_1, \dots, p_n$, and $\phi_1, \dots, \phi_n$ are any expressions of the propositional calculus, then we can obtain another valid argument by replacing each p_i by ϕ_i wherever it appears in the original argument.

Logical equivalences If ϕ appears in an argument, and $\psi \equiv \phi$, then we may introduce ψ at any later stage in the argument.

Conditional arguments If we have a valid argument in the form $\phi_1, \dots, \phi_n, \therefore \phi$, then for any expression ψ the argument $\psi \to \phi_1, \dots, \psi \to \phi_n, \therefore \psi \to \phi$ will also be valid.