A Collection of Proof Tricks

ATRI RUDRA

atri@buffalo.edu

Last updated: August 31, 2014

Abstract

In this article, I will collect some of the proof tricks that we will use in CSE 331. This article is meant more of a reminder of tricks that you should have seen elsewhere and is *not* meant as a substitute for learning how to write proofs.

Contents

1 Introduction

In this article, I will collect the different proof tricks that we will see in CSE 331 this semester. A word of warning: this is just meant to collect the tricks with some examples. To prepare better, it would be better if you go back to your CSE 191 book and go through the chapters on proofs and logic. Or if you want another source, you can check out this book:

> Daniel J. Velleman, "How to Prove It: A Structured Approach (2nd Ed)." Cambridge University Press, 2006.

Or any other book on proofs that you like.

> **WARNING: This note is supposed to list some tricks that could be useful to you in writing proofs. This listing is not exhaustive and in particular you might need to use a trick not in this note to solve some problems in the course.**

Finally, there might be typos in the document. If you find any, please let me know: thanks!

2 Proving $P \implies Q$

Many of the proofs we will encounter in this course will be required to prove logical statements of the form

If *blah* then *blah'*,

which is logically equivalent to the form $P \implies Q$ (where in this case P stands for *blah* and Q stands for *blah*'). In this section we will see many ways to prove such a statement. To do that we will consider the following exercise throughout:

Exercise 1. *Prove the following. If every human who is active also has blue eyes, then every human who is active and is curious also has blue eyes and is curious.*

To solve the above problem, it would be useful to convert this into an implication. Let A denote the set of all humans who are active, B denote the set of all humans with blue eyes and C denote the set of all curious humans. Then Exercise 1 is exactly the same as proving

Exercise 2. *If $A \subseteq B$, then for any set C, $A \cap C \subseteq B \cap C$.*

Before we go on to present multiple ways to prove an implication, here are two remarks:

- Instead of using operators on sets, we could also have used logic. In particular, here is the equivalent first order logic statement

$$(\forall x(x \in A \implies x \in B)) \implies \big(\forall y(y \in A \cap C \implies y \in B \cap C)\big),$$

  or further

$$(\forall x(x \in A \implies x \in B)) \implies \big(\forall y(y \in A \wedge y \in C \implies y \in B \wedge y \in C)\big), \tag{1}$$

  where recall "$\wedge$" is the logical AND function.

- We remark that Exercise 2 is true for arbitrary sets A, B and C while Exercise 1 is for specific sets. However, note that Exercise 1 does not specify anything about the humans in A, B and C other than their attributes (i.e. humans who are active, are blue-eyed and are curious respectively). Even though in English being active, having blue-eyed or being curious have specific meanings, variables in mathematics have no specific meaning. So e.g. Exercise 2 would work even if A, B and C are goats who are active, are blue-eyed and are curious respectively. Or even if A, B and C were humans who were aggressive, are brown-skinned and cold-blooded respectively. This distinction between meanings in English and mathematical variables is an important distinction and something that you should keep in mind throughout the course.

We begin with a common mis-understanding.

2.1 A mis-understanding

Here is a common mistake when one encounters $P \implies Q$. Looking at Exercise 2, one might say "I know someone who is active and has brown eyes" and then conclude that $A \not\subseteq B$ and hence the statement in Exercise 2 is false.

The above happens because of a mis-understanding of the logical expression $P \implies Q$. If P is false then logically the statement $P \implies Q$ is *true*. Hence, we "only" need to worry about the case when P is true and then also prove that Q is true. This leads to the first way to prove implications.

2.2 Trick 1: Assume P. Then prove Q.

The "simplest" way to prove $P \implies Q$ is the following:

> Assume that the statement P is true. Then with this assumption, prove that Q is true.

Next, we use this trick to solve Exercise 2

Proof Idea: We will assume that $A \subseteq B$. Then we will argue that $A \cap C \subseteq B \cap C$. To do this, we will use the logical forms of these statements that are used in (1). □

Proof Details: Assume $A \subseteq B$. This implies for for every $x \in A$, we have $x \in B$. Now consider an arbitrary $y \in A \cap C$, which implies that $y \in A$ and $y \in C$. Now by the assumption $A \subseteq B$, we have $y \in B$, i.e. $y \in B$ and $y \in C$, which by definition, implies that $y \in B \cap C$. Since the choice of y was arbitrary, we have $A \cap C \subseteq B \cap C$, as desired. ■

2.2.1 Trick 1': Case Analysis

A special case of Trick 1, which is pretty useful is to break up the assumption into cases that cover all the possibilities. In particular,

> Let P be of the form $P_1 \vee P_2 \vee \cdots \vee P_m$. Then for $i = 1, \ldots, m$ do the following:
>
> - Assume P_i is true. Then prove Q.
>
> Conclude that $P \implies Q$.

In the above $P_1, \ldots, P_m$ are the cases. As an exercise,

Exercise 3. *Convince yourself that the case analysis above indeed proves* $(P_1 \vee P_2 \vee \cdots \vee P_m) \implies Q$.

We illustrate the trick above with the following exercise:

Exercise 4. *Prove the following. For any real number x, if $|x-3| \geq 5$, then either $x \geq 7$ or $x \leq -1$.*

Proof Idea: Define $P_1 = x \geq 8$ and $P_2 = x \leq -2$. Note that $|x-3| \geq 5$ is equivalent to $P_1 \vee P_2$. We will prove the result by a case analysis. □

Proof Details: For notational convenience define $Q = (x \geq 7) \vee (x \leq -1)$. We do a case analysis:

- (Case 1: $x \geq 8$, i.e. P_1 is true.) Note that since $x \geq 8$ it is also true that $x \geq 7$ and hence Q is true.
- (Case 2: $x \leq -2$, i.e. P_2 is true.) Note that since $x \leq -2$ it is also true that $x \leq -1$ and hence Q is true.

Since $|x-3| \geq 5$ implies either Case 1 or Case 2 is true, we have proved that $P_1 \vee P_2 \implies Q$, as desired. ■

2.3 Trick 2: Assume $\neg Q$. Then prove $\neg P$.

The next trick is to use the fact that $\neg Q \implies \neg P$ is logically *equivalent* to $P \implies Q$:

> First assume that $\neg Q$ is true (i.e. Q is false). Then prove that $\neg P$ is true (i.e. P is false).

Next, we use this trick to solve Exercise 2.
Proof Idea: We will assume that $A \cap C \not\subseteq B \cap C$. Then we will argue that $A \not\subseteq B$. To do this, we will use the logical forms of these statements that are used in (1). □

Proof Details: Assume that $A \cap C \not\subseteq B \cap C$. That is, there is an x such that $x \in A \cap C$ but $x \notin B \cap C$.[1] Note that $x \in A \cap C$ implies that $x \in A$ and $x \in C$. On the other hand, $x \notin B \cap C$ means that either $x \notin B$ or $x \notin C$. These two conclusions imply that $x \in A$ but $x \notin B$. This in turn implies that $A \not\subseteq B$, as desired. ■

2.4 Trick 3: Proof by Contradiction

Proof by contradiction is a very general (and useful) proof technique, where the idea is to assume the negation of what you're trying to prove and then come up with a contradiction (i.e. where you prove both a statement and its negation to be true). In the context of proving $P \implies Q$, this generally takes the following form

> Assume both P and $\neg Q$. Then arrive at a contradiction.

We now use this trick to solve Exercise 2.
Proof Idea: We will assume that both $A \subseteq B$ and $A \cap C \not\subseteq B \cap C$ and then arrive at a contradiction. To do this, we will use the logical forms of these statements that are used in (1). □

Proof Details: Let us assume that both $A \subseteq B$ and $A \cap C \not\subseteq B \cap C$ are true. The first assumption implies that for every x such that $x \in A$ it is true that $x \in B$. The second assumption implies that there exists a $y \in A \cap C$ such that $y \notin B \cap C$. The latter implies that

(i) $y \in A$

(ii) $y \in C$

(iii) Either $y \notin B$ or $y \notin C$.

Now note that (ii) and (iii) imply that $y \notin B$. This along with (i) implies that $y \in A$ but $y \notin B$ but this contradicts our first assumption. ■

2.5 Which trick should I apply?

You might have noted that the three solutions to Exercise 2 are similar. A natural question to ask is if there is a way to decide when one should use a particular trick. Unfortunately, the answer is that there is no mechanical way to do this. The more proofs you write the better you will get at picking the "correct" trick. (By correct I mean that sometimes a certain proof trick leads to a much simpler proof than what

[1] Note that if $S \not\subseteq T$ then one can only conclude that there is an $x \in S$ but not in T. However, we *cannot* conclude that there is an $y \in T$ but not in S (because the latter can be true even if $S \subseteq T$.

one gets by applying the other tricks and the “trick" is to figure out which one to pick.) When you are starting off pretty much the only choice you have is to try out all the tricks one by one and see which one works.

3 Prove $S = T$

A reasonably common problem we will encounter in this course it to prove that two sets S and T are in fact equal. Of course if both S and T have the exact description, this problem will be trivial. However, typically the descriptions of S and T will be different and hence, if we claim that $S = T$, then we need to present a proof for the claim. Below we will talk about one very common proof trick to prove set equality.

3.1 Trick 1: Prove $S \subseteq T$ and $T \subseteq S$.

This trick is based on the simple fact that $S = T$ if and only if $S \subseteq T$ *and* $T \subseteq S$:

> First prove $S \subseteq T$. Then prove $T \subseteq S$.

We will apply this trick on the following exercise:

Exercise 5. *You are an investigative journalist who is looking into two companies. The two companies are in the niche area which builds tiles with integral dimensions. WeLoveEvenSides claims to be the only company on earth to have all possible rectangular tiles where at least the length or the breadth is even. WeLoveEvenArea claims to be the only company on earth to have all possible rectangular tiles whose area is divisible by two. Your preliminary investigations have led you to believe that these two companies are indeed the same. Now deliver the knockout argument by proving that the set of rectangular tiles manufactured by WeLoveEvenSides is exactly the same as the set of rectangular tiles manufactured by WeLoveEvenArea (which by the claims of the companies shows that they are the same).*

As an aside:

Exercise 6. *Argue why you cannot deliver the knockout blow in Exercise 5 if the dimensions of the tiles can be real numbers.*

Whenever you see such a verbose problem statement, it is useful to abstract out the mathematical problem. I claim that the above is the same as the following problem (it is an exercise to convince yourself that my claim is indeed true):

Exercise 7. *Let R be the set of all rectangles with integral length and breadth. Let $R_1 \subseteq R$ be the set of rectangles with even length or breadth. Let R_2 be the set of all rectangles with even integral area. Prove that $R_1 = R_2$.*

We now use our trick to solve Exercise 7.

Proof Idea: We will first argue that $R_1 \subseteq R_2$. Then we will argue that $R_2 \subseteq R_1$. To do this we will use the fact that a rectangle with length ℓ and breadth b has an area of $\ell \cdot b$. □

Proof Details: We first prove $R_1 \subseteq R_2$. Consider an arbitrary $r \in R_1$ with length ℓ and breadth b. Since $r \in R_1$ either ℓ or b is even. Since multiplying an even number with another integer results in another even number, we conclude that $\ell \cdot b$ is even. This, by definition implies $r \in R_2$. Since the choice of r was arbitrary, we have $R_1 \subseteq R_2$.

We now prove that $R_2 \subseteq R_1$. Consider an arbitrary $r \in R_2$ with area a. This implies that a is even. Now let ℓ and b denote the length and breadth of r. Note that this implies that $\ell \cdot b$ is even. We claim that either ℓ or b is even. (We will prove this claim shortly.) Note that by definition this implies that $r \in R_1$. Since the choice of r was arbitrary, we have $R_2 \subseteq R_1$, as desired.

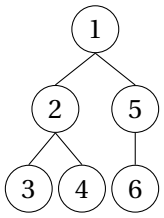
Finally, we prove the claim. For the sake of contradiction, let us assume that both ℓ and b are odd. Since the product of two off integers results in an odd integer, it must be the case that $a = \ell \cdot b$ is odd. This is a contradiction (since a is even), which proves the claim. ■

4 Induction

Induction is probably one of most widely used proof techniques. Perhaps the main difference from induction proofs that you might have seen in CSE 191 and the ones we will see in this course is that we will typically do induction on structures (such as graphs) instead of performing inductions on numbers. I will assume that you know the general structure of an induction proof and will present an example using trees.

4.1 Example with trees

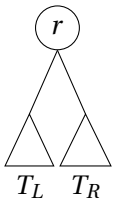
We will consider an inductive proof of a statement involving rooted binary trees. If you do not remember it, recall the definition of a rooted binary tree: we start with root node, which has at most two children and the tree is constructed with each internal node having up to two children. A node that has no child is a leaf. For example, below is a rooted binary tree with six nodes.



Recall that the depth of a rooted binary tree is the maximum number of edges it takes to go from the root to any leaf. In the example above, the tree has depth 2. Consider the following problem:

Exercise 8. *Prove that a rooted binary tree with depth d has at most $2^{d+1} - 1$ nodes in it.*

Note that in the above there is a parameter/number: the depth d. However, when we do induction, we will not just do induction d instead we will do induction on *all* binary rooted trees of depth d.
Proof Idea: We will induct on depth d (or more precisely on binary rooted trees of depth d). As usual, the interesting part is proving the inductive step, where we are in a situation as below:



In the above, the root r has up to two children: let the subtree rooted at the left child be called T_L and the subtree rooted at the right child be called T_R. The main observation is that if the original tree has depth d, then both T_L and T_R have depth at most $d-1$ and thus, we can apply induction on these subtrees. □

Proof Details: We will prove the statement by induction on (all rooted binary trees of) depth d.

For the base case we have $d = 0$, in which case we have a tree with just the root node. In this case we have 1 nodes which is at most $2^{0+1} - 1 = 1$, as desired.

For the inductive hypothesis, we will assume that any tree with depth $d \leq k$ has at most $2^{d+1} - 1$ nodes in it. For the inductive step, consider any rooted binary tree T of depth $k+1$. Let T_L denote the subtree rooted at the left child of the root of T and T_R be the subtree rooted at the right child of T (if it exists).[2] Since the path from the root of T to any leaf in T_L (T_R resp.) has to go through the root of T_L (T_R resp.), which is the left (right resp.) child of the root of T, it follows that the depth of both T_L and T_R are at most k. By the inductive hypothesis, both of these sub-trees have at most $2^{k+1} - 1$ nodes each.

Now the total number of nodes in T is the number of nodes in T_L and T_R plus one (for the root), which is at most

$$2 \cdot (2^{k+1} - 1) + 1 = 2 \cdot 2^{k+1} - 1 = 2^{k+2} - 1,$$

as desired. ■

5 Reduction

This is a trick that you might not have seen explicitly before. However, this is one trick that you have used many times: it is one of the pillars of computer science. In a nutshell, reduction is a process where you change the problem you want to solve to a problem that you already know how to solve and then use the known solution. Let us begin with a concrete non-proof examples.

5.1 Example of a Reduction

We begin with a popular elephant joke. There are many variants of this joke. The following one is adapted from here.[3]

Question 1. How do you stop a rampaging blue elephant?
Answer 1. You shoot it with a blue-elephant tranquilizer gun.

Question 2. How do you stop a rampaging red elephant?
Answer 2. You hold the red elephant's trunk till it turns blue. Then apply Answer 1.

Question 3. How do you stop a rampaging yellow elephant?
Answer 3. Make sure you run faster than the elephant long enough so that it turns red. Then Apply Answer 2.

In the above both Answers 2 and 3 are reductions. For example, in Answer 2, you do some work (in this case holding the elephant's trunk: in this course this work will be a mathematical argument) to

[2] Since the depth of T is at least 1, the root has at least one child. Pick one child and call it the left child and if there is another child, call it the right child.

[3] Thanks to Anna Gilbert and Martin Strauss for suggesting this example.

change Question 2 in a way so that you can map it to Question 1. Once you have the mapping, then you use the known Answer 1 to Question 1 to arrive at Answer 2 for Question 2.

When folks actually tell this joke, they do not say apply Answer 1: they spell it out but as CS folks we know we should just call existing functions instead of re-writing the function in full when we need it. This actually also points to where you have been using reductions when you program: whenever you use a library function, you are essentially doing reduction. Say in your program you need to sort numbers, then you just call a sorting routine from the library. The sequence of steps by which you went from your original problem to the point where you need to sort the numbers is the reduction in this case. Typically, this is not the end of the story since you might have to write more code to use the result of sorting to obtain the final desired output. Similarly, for general reductions, just using the solution to a known problem might not be enough: you might have to do some "post-processing" to convert the known solution into a format that is suitable for your problem. (In the elephant joke above, there is no need for post-processing.)

5.2 Exercise

We now present a more mathematical example for reduction. In fact, we are just going to re-use Exercise 4 from HW 0.

Exercise 9. *Deep in the Pacific ocean you discover the RapidGrowers bacteria. These are single cell organism that divide themselves into two (identical) organisms/cells in one second. When you start going back to the surface you store one organism in a special container that exactly replicates their habitat. From the moment you put the organism in, it takes you s seconds to come back to Buffalo to find your container bursting at its seams.*[4] *Prove or disprove the following:*

There are 2^s RapidGrowers in your container.

Proof Idea: There are two ways of solving this problem. (See HW 0 for the induction based proof.)

We will use the approach of reducing the given problem to a problem you have seen earlier.[5] Build the following complete binary tree: every internal node in the tree represents a "parent" RapidGrower while its two children are the two RapidGrowers it divides itself into. After s seconds this tree will have depth s and the number of RapidGrowers in the container after s seconds is the number of leaf nodes the complete binary tree has, which we know is 2^s. Hence, the claim is correct. □

Proof Details: Consider the complete binary tree with height s and call it $\mathcal{T}(s)$. Further, note that one can construct $\mathcal{T}(s+1)$ from $\mathcal{T}(s)$ by attaching two children nodes to all the leaves in $\mathcal{T}(s)$. Notice that the newly added children are the leaves of $\mathcal{T}(s+1)$. Now assign the root of $\mathcal{T}(0)$ as the original RapidGrower in the container. Further, for any interval node in $\mathcal{T}(s)$ ($s \geq 0$), assign its two children to the two RapidGrowers it divides itself into. Then note that there is a one to one correspondence between the RapidGrowers after s seconds and the leaves of $\mathcal{T}(s)$.[6] Then we use the well-known fact (cite your

[4] You may assume that your container is such that the RapidGrowers can continue replicating every second till you come back to Buffalo.

[5] In other words, we will cast the given problem into a problem for which we know the solution and then directly appeal to the previously known solution. In the present case the reduction is just a matter of changing the language: in later problems in the course, you will have to do more work in the reduction.

[6] Technically, you should prove this claim by induction too but here you can make this claim without a formal proof. In general, you will have to make such decisions at many points while writing your HW solutions. If you are not sure, take the conservative approach. That is, in the present case, go ahead and present the formal proof by induction.

191/250 book here with the exact place where one can find this fact): $\mathcal{T}(s)$ has 2^s leaves, which means that the number of RapidGrowers in the container after s seconds is 2^s, which means that the claim is correct. ■

6 Using a Progress Measure

This is another trick that you might not have studied formally but have used (implicitly) before. This trick is generally used to bound the number of times a loop is executed in an algorithm. Since most non-trivial algorithms have loops in them, this is a useful trick to remember when trying to bound the run time of an algorithm (which you will have to do frequently in this course). Most of the time you will need to use the trivial version of this trick.

6.1 A simple example

Let us begin with a prototypical example that you have already seen. Consider the following simple problem: given $n+1$ numbers $a_1, \ldots, a_n; v$, we should output $1 \le i \le n$ if $a_i = v$ (if there are multiple such i's then output any one of them) else output -1. Below is a simple algorithm to solve this problem.

Algorithm 1 Simple Search

INPUT: $a_1, \ldots, a_n; v$
OUTPUT: i if $a_i = v$; -1 otherwise

```
1: FOR every 1 ≤ i ≤ n DO
2:     IF a_i = v THEN RETURN i
3: RETURN −1
```

We will consider the run time analysis of this algorithm in the course but the main step is to compute an upper bound on the number of iterations of the loop in Step 1. In this case it is easy to come up with a bound of n: in each iteration of the loop, the value of i is incremented by 1. i is initialized to a value of 1 and the loop stops once $i \ge n+1$. This means i can be incremented at most n and since in each loop the value of i is incremented, the loop can be iterated in at most n times.

6.2 Trick: Bounding the Progress

One can generalize the above argument to bound the number of iterations of pretty much any loop we will encounter in this course. Here is general outline:

> Define a notion of *progress*, i.e. *after* the i'th iteration of the loop, let $\mathcal{P}(i)$ denote a integer with the following properties:
>
> 1. Argue that $\mathcal{P}(1) = \ell$ (here I am assuming that the loop starts with $i = 1$);
> 2. Argue that $\mathcal{P}(i+1) > \mathcal{P}(i)$ for every $i \ge 1$;
> 3. Argue that for every $i \ge 1$, $\mathcal{P}(i) \le u$.
> 4. Conclude that the number of iterations is bounded by $u - \ell + 1$.

As an exercise, convince yourself that the above is a valid trick:

Exercise 10. *Argue that the last step above follows from the first three arguments.*

Let us now formalize the argument for the number of iterations of the loop in Step 1. In this case we simply define $\mathscr{P}(i) = i$. Now consider the following four step argument:

1. Note that at the end of the first iteration of the loop, we have $\mathscr{P}(1) = 2$. (Here I am using the convention that in a FOR loop the value of the index gets incremented *after* the body of the loop is executed.)

2. Since the index i is incremented at the end of the i'th iteration, we have $\mathscr{P}(i) = i+1$. In particular, this implies that $\mathscr{P}(i+1) > \mathscr{P}(i)$.

3. Since the loop is not executed if $i > n$, this implies that $\mathscr{P}(i) \leq n+1$.

4. Conclude that the number of iterations is upper bounded by $n+1-2+1 = n$, as desired.

7 The Pigeon-hole principle

This is another trick that you might not have seen formally before but it formalizes something pretty obvious (and turns out to be pretty useful too). Let us start with the most common instantiation of this principle:

Exercise 11. *Consider any assignment of $n+1$ pigeons to n holes. Then there exists at least one hole with at least 2 pigeons in it.*

Next, we prove the above with a proof by contradiction.
Proof Idea: We will prove Exercise 11 with a proof by contradiction. We will assume that all holes have at most one pigeon in them and then conclude that there are at most n pigeons. □

Proof Details: For notational convenience, let h_i denote the number of pigeons in the ith hole for every $i \in [n]$. For the sake of contradiction, let us assume that $h_i \leq 1$ for every $i \in [n]$. Now since each pigeon is assigned to some hole, the total number of pigeons is

$$\sum_{i=1}^{n} h_i \leq \sum_{i=1}^{n} 1 = n,$$

where the inequality follows from our assumption that $h_i \leq 1$ for every $i \in [n]$. This implies that the total number of pigeons is at most n, which is a contradiction since there are $n+1$ pigeons. This implies[7] that there exists at least one $i \in [n]$ such that $h_i \geq 2$, as desired. ■

In fact, one can modify the proof above to obtain the more general version (proof is left as an exercise):

Exercise 12. *Consider any assignment of m pigeons to n holes, Then there exists at least one hole with at least $\lceil \frac{m}{n} \rceil$ pigeons in it.*

[7] We also use the fact that every h_i is an integer and $h_i \geq 0$ for every $i \in [n]$.

Finally, one can prove the “converse" of the pigeon-hole principle above (again the proof is left as an exercise):

Exercise 13. *Consider any assignment of m pigeons to n holes. Then there exists at least one hole with at most* $\lfloor \frac{m}{n} \rfloor$ *pigeons in it.*

7.1 An Application

We now see how we can use the pigeon-hole principle to prove another statement:

Exercise 14. *Consider a matching between n men and n women. If a man (or woman resp.) is not matched then there must exist a woman (man resp.) who is also not matched.*

Proof Idea: We will prove the above by the pigeon-hole principle (Exercise 13). Let us consider the case when a woman is not matched– then the matched women will be the pigeons and all the men will be the holes. Then we will conclude that there is at least one man who is not matched. □

Proof Details: We will prove the case when there is an unmatched woman (and we have to show that there exists an unmatched man). The proof for the case when there is an unmatched man is similar and is omitted.

Let w be the number of matched women. Note that since there is at least one woman who is not matched, we have $w \leq n-1$. Now think of each of the w woman to be a pigeon and the n men to be the n holes. The assignment of the pigeon to the holes is obvious: every matched woman is assigned to a unique man. Then by Exercise 13, there is one hole with at most $\lfloor \frac{w}{n} \rfloor \leq \lfloor \frac{n-1}{n} \rfloor = 0$ pigeons in it. This means that there is a man who is not assigned any woman, i.e. he is unmatched, as desired. ■

A Notation

Below are tables with (some) notations that we will use in this course (and is used in this article).

Logical Connectives

Notation	Meaning
$P \wedge Q$	The logical AND of boolean variables P and Q
$P \vee Q$	The logical OR of boolean variables P and Q
$\neg P$	Negation of the boolean variable P
$P \implies Q$	Logically equivalent to $\neg P \vee Q$
$\forall x P(x)$	For every x (in the appropriate domain), the boolean predicate $P(x)$ is true
$\exists x P(x)$	There exists at least one x_0 (in the appropriate domain), the boolean predicate $P(x_0)$ is true

Set Connectives

Notation	Meaning
$x \in S$	Element x belongs to set S
$\bar{S}$	Complement of S
$S \subseteq T$	S is a subset of T (they can be equal). Logically, $\forall x(x \in S \implies x \in T)$
$S \supseteq T$	$T \subseteq S$.
$S \cap T$	Intersection of sets S and T; i.e. $x \in S \cap T$ if and only if $x \in S \wedge x \in T$
$S \cup T$	Union of sets S and T; i.e. $x \in S \cup T$ if and only if $x \in S \vee x \in T$
$\emptyset$	The empty set, i.e. the set with no elements.
$S \setminus T$	Set difference of S and T. $S \setminus T = S \cap \bar{T}$
$S \times T$	Cartesian product of S and T. $S \times T = \{(a,b) \mid a \in S \text{ and } b \in T\}$
$[m]$	The set $\{1, 2, \ldots, m\}$

Other Notation

Notation	Meaning
$\lfloor \frac{m}{n} \rfloor$	m/n rounded down to the largest integer $\leq m/n$
$\lceil \frac{m}{n} \rceil$	m/n rounded up to the smallest integer $\geq m/n$

B Some useful mathematical identities

In no particular order:

1. $$\log_a b = \frac{\log_c b}{\log_c a} \text{ for any } a, b, c > 0.$$

2. $$a^b \cdot a^c = a^{b+c} \text{ for any } a, b, c.$$