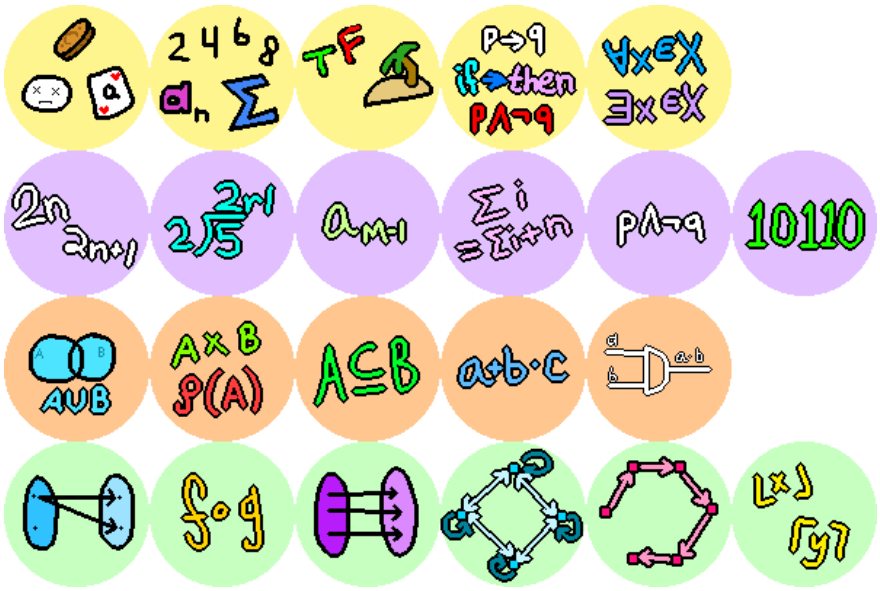
Rachel's Discrete Math Notes



Overview notes on Discrete Math topics by Rachel Morris

Last updated September 7, 2017

Contents

Chapter 1

Introduction & Logic

At the beginning of our course, we introduce some little word problems and games as a way to get you to start thinking methodically about the types of problems we will face. The important parts of Chapter 1 of our textbook is the propositions, predicates, and implications, so we will start here with our review, and come back to the sequences later on.

Propositions

Proposition: A proposition is a type of statement that is either *true or false*. These statements are standalone, and don't take any input. If you're familiar with programming, they are essentially like a boolean variable.

"4 is greater than 5."
"Pratik is taking Discrete Math."
"The moon is bigger than the sun."

Note that a proposition *doesn't have to be true* – the moon is not bigger than the sun – but it is still a proposition because we can identify it as either true or false.

Propositional variables: When we want to work with propositions, we will usually come up with a propositional variable. These are like variables in algebra, but instead of containing numbers, they contain propositional statements.

While working with propositions, we will write 'em both in mathematical terms, like:

p is the proposition, "length and width are equal."

As well as propositions that are just English statements, like:

q is the proposition, "Mariam likes cats."

Either way, we can build statements out of these!

Logic operators and compound statements

If you've spent some time programming, you've probably run across if statements that contain "and", "or", and "not" in the statements. In C++, this would look like:

```
if ( a > 0 && a < 10 )
```

For Discrete Math, we will also have these logic operators, but they look different.

English	**C++**	**Logic**
AND	`&&`	$\wedge$
OR	`||`	$\vee$
NOT	`!`	$\neg$

When we use these operators to combine two (or more) propositions, the result is a **compound statement**, like this:

p is the proposition, "Brandon is allergic to cats"
q is the proposition, "Brandon owns a cat"

$p \wedge q$: "Brandon is allergic to cats, and Brandon owns a cat."
$p \vee q$: "Brandon is allergic to cats, or Brandon owns a cat."
$p \wedge \neg q$: "Brandon is allergic to cats, and Brandon *does not* own a cat."

Negations

Using negations, we can turn a propositional statement into its logical opposite... This means that if a statement (or compound statement) is *true*, then the opposite is *false*, and if the statement is *false*, then the opposite is *true*.

If we create the proposition p is "the printer is functioning."
Then the negation, $\neg p$, is "the printer is **not** functioning."
We can also negate compound statements, which follow some rules...

The negation of $p \wedge q$ is written as $\neg(p \wedge q)$,
and is *logically equivalent* to $\neg p \vee \neg q$.

The negation of $p \vee q$ is written as $\neg(p \vee q)$,
and is *logically equivalent* to $\neg p \wedge \neg q$.

When two statements are **logically equivalent**, it means that their outcomes will be the same based on the values of their propositional variables - in this case, p and q.

Negating relational operators
You might not have thought about it before, but the negation of something containing a $>$ might not be what you think: The logical opposite of $>$ is $\leq$, and the logical opposite of $<$ is $\geq$.

a is the proposition, $x > 2$.	$\neg a \equiv x \leq 2$.
b is the proposition, $y < 10$.	$\neg b \equiv y \geq 10$.

And perhaps more intuitively, the negation of $=$ is going to be $\neq$.

c is the proposition, $z = 5$.	$\neg c \equiv z \neq 5$.

Just to drive the point home a little more, let's assume we're writing a program and using an *if, else* statement to check some variable.

```
if ( age < 18 )
{
    print( "Can't vote" );
}
else
{
    print( "Can vote" );
}
```

If the user enters some number less than 18, they will see the "Can't vote" message pop up. However, the else statement will catch any other cases - in this case, if age is equal to 18, or if it is greater than 18.

Contradictions and Tautologies

We can also end up writing propositional statements that always result to being *true* no matter what, or always result to being *false* no matter what. For the first case, if a statement is always true, then it is a **tautology**. On the other hand, if the statement is always false, then it is a **contradiction**.

For example, if we said "if both p and q are true, then p is true", then that will always be true. If we say $p \wedge \neg p$, then this will always be false - p cannot be both true and not-true at the same time.

Truth tables

How do we know if a compound statement is true or false? How can we tell if two compound statements are logically equivalent? We can use **truth tables** to diagram out all possible states in our propositional statements.

For a truth table, the left-hand side contains the propositional variables being used by the statement, and the right-hand side contains the compound statement itself (and perhaps some "in-between" statements to help us solve the final form.) Using the basic truth tables for $p \wedge q$, $p \vee q$ and $\neg p$, we can figure out the truth tables for more complex propositional statements.

Let's say we want to know what the result of $p \wedge \neg q$ is, for all possible values of p and q. We could diagram it like this:

Variables			Compound statement
p	q	$\neg q$	$p \wedge \neg q$
T	T	F	F
T	F	T	T
F	T	F	F
F	F	T	F

But first, let's look at our truth tables for AND, OR, and NOT.

And

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

Or

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

Not

p	$\neg p$
T	F
F	T

Practice

Predicates

Implications