Boolean Algebra much like the usual +, × say over $\mathbb{Z}$.

AND $\equiv$ ×

OR $\equiv$ +

Note: distributive law + such works:

X AND (Y OR Z)

$X(Y+Z) = XY + XZ$

(X AND Y) OR (X AND Z)

Can actually embed Boolean algebra in $\mathbb{Z}$:

Suppose $X, Y \in \{0, 1\} \subseteq \mathbb{Z}$.

X AND Y $\equiv$ XY

NOT X $\equiv 1 - X$

X OR Y $\equiv !(!X \text{ AND } !Y)$ (Demorgan)

$\equiv 1 - (1-X)(1-Y)$

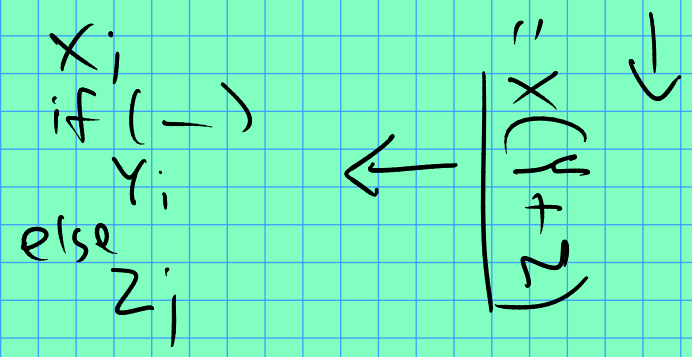
Algebra of program structure

$X_i\ Y_i \equiv$ AND. $\equiv$ ×.

```
if (__)
    X_i   ≡ OR ≡ +
else
    Y_i
```

```
if (__) {
    X_i
    Y_i
} else {
    X_i
    Z_i }
```

→ | × + (×) ↓



More complicated stuff works too.
Exercise: try FOIL

Continuing the analogy:

for (i=1; i≤n; i++)
  X_i; $\equiv \prod_{i=1}^{n} X_i$

Switch (w/ exclusive cases)
or an array of function pointers $\equiv \sum X_i$

Programming Pattern: "Reduce"
(aka "fold", "catamorphism"):

Setup: you have an array of values $X_1, \ldots X_n$
+ a function that takes 2 values to 1:
$f: V \times V \to V$. To "fold" the
X_i via f is to compute

$f(\ldots f(f(f(x_1, x_2), x_3), x_4) \ldots) \in V$

Example: sum: $f(x,y) = x+y$.

How to program it?

```
V r = x_1;
for(i=2; i<=n; i++)
    r = f(r, x_i);   <-
return r;
```

More examples: Computing max or min, or product.

Even your prime test can be seen as a reduce:

X_i are booleans, defined as

$X_i \equiv$ `n%i != 0`

Being prime means that for $i = 2, \ldots n-1$, X_i must be true:

$prime \equiv X_2$ AND X_3 AND ... AND X_{n-1}.

(The folding function f is just AND).