Boolean Reductions to Regular Algebra

Ethan Holland Simmons

May 30, 2020

*Abstract*

*It is often said computer science is intertwined with math. Boolean algebra is called algebra for a reason, but it only works for 1 or 0 in its normal state. These ideas of and, or and not can be extended to normal mathematics. All boolean expressions within programming can be done with algebra, returning a 0 or 1.*

# 1.) Introduction

With boolean expressions in programming, the end result is true or false. In boolean algebra this is 1 and 0 respectively. A brief recap on logic gates if you’re not familiar:

Each ‘gate’ is a sort of magic box. It takes some number of inputs that are either true or false. It returns (or gives out) a value of true or false. There are 3 basic gates that can scale up and build incredibly complex functions and do amazing things (short term memory is one of them). They are as follows:

AND: returns true if both inputs are true, false otherwise.

OR: returns true if any of the inputs are true, false otherwise.

NOT: inverts an input. True becomes false, false becomes true.

These are represented by fancy symbols, but that’s really not important to understand this. All you need to know is it takes inputs and returns true or false (remember these can be seen as 1 and 0 respectively).

# 2.) The story

I really wanted to create a way for these logic gates to be used in normal mathematics to complete and justify my theory that simple math could be turing complete. The primary issue I ran into was reducing all numbers to the range of 0 and 1. 0 has some special properties, but also tends to become undefined if you’re not careful. Rounding also seemed to look promising for reducing values from a huge domain to a smaller one. One fateful day in government class (sorry mr. duff, it’s not you, it’s me) I chanced on the correct equation to make conditionals.

# 3.) The equations

|  |  |  |
| --- | --- | --- |
| Meaning | Expression | Alternative |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 4.) Expanding

The above expressions are universally expandable. For example:

* To test equality of 2 numbers, the desired expression would have to be reduced to a known expression.
* ->
* To do this, subtract b from both sides and negate the expression, resulting in
* Fully expanded is

These expressions can be chained with others to create any expression desired. Equality, inequalities and the 3 basic logic gates can all be used together to create any expression.

# 5.) Use cases

Evaluating to only a 0 or 1 can have powerful uses. A chain of conditionals can be condensed into a single expression. For example, take the following pseudo code (with some input n):

integer x;

if( n == 1 ) x=20;

else if( n == 2) x=30;

else if(n == -10) x=0;

Can be reduced to:

x = (n=?1)\*20 + (n=?2)\*30;

Or fully expanded:

x =

## 5.1) Programming

The ability to evaluate expressions into boolean values allows programs to (sort of) be built inside of a list of normal calculations. This allows a calculator with little to no programming ability to do some sort of higher computations. There are calculators that have programming built in (TI-84 with some odd basic and assembly for example), but for the vast majority of “basic” calculators, programming is now achievable.

### 5.1.1) Obfuscation

While the readability of the above example just dropped, the readability of it just dropped! It completely flattened a control flow graph and killed any flow visualization. To further emphasize this, the above expressions are converted into popular programming syntax.

|  |  |  |
| --- | --- | --- |
| Expression | Python | Java |
|  | math.ceil(-1/(abs(a)+1)+1) | Math.ceil(-1/(Math.abs(a)+1)+1) |
|  | math.ceil(-1/(abs(a0+a1…)+1)+1) | Math.ceil(-1/(Math.abs(a0+a1...)+1)+1) |
|  | a0\*a1... | a0\*a1... |
|  | math.ceil(-1/(abs(a)+a+1)+1) | Math.ceil(-1/(Math.abs(a)+a+1)+1) |
|  | 1-math.ceil(-1/(abs(a)-a+1)+1) | 1-Math.ceil(-1/(Math.abs(a)-a+1)+1) |

The major selling point for Java and Python is readability! The structure of the languages is similar here, but no matter what, those expressions look nothing like the expression on the far left. For those that do not see the use of obfuscation, that is not the scope of this paper. Needless to say, it’s very practical (and a sort of diabolical fun).

# 6.) Conclusion

Reducing integer math to work in a boolean domain (0, 1) has practical uses. It adds functionality to regular expressions, flattens branching, and obfuscates very plain commands. It makes programming just a bit more exciting and gives calculators power they didn’t have before. It most certainly makes the time complexity go through the roof (calculate that and tell me if you care to) and is horrible to write, but it amused me enough to write this and hopefully you can get something out of it.