Extension of the domain of classical logic gates to the complex numbers

Onna A, (Also Known under Lucy)

October 2020

1 Introduction

In this document I will seek to extend the domain of logic gates to complex numbers. There are two obvious ways to do this: Using NAND completeness, and the laws of probability. Both of these approaches have benefits and drawbacks and I cannot, at this point, determine a clear extension. The probabilistic approach allows one to compute the probability of a system having a certain output in some (but not all) cases. The completeness approach only requires two simple definitions to be able to describe all logical operations of a classical computer; while the probabilistic approach requires multiple.

1.1 Note on a desired property

There is property which is arguably the most important property for an extension like this to have. I am not sure if it is possible. Given the set of all systems of logic gates which produce the output of another system, S_{gate}, for example the following set of NAND gates:

$$\text{O} = (\text{A NAND A}) \text{ NAND } (\text{B NAND B})$$

produces the behaviour of an OR gate; so $O \in S_{\text{OR}}$. Said simply, the extension should produce the same truth table if it produces the same truth table for binary inputs.

The property mentioned, is that the function generated by all systems in S_x is the same function. This will be true when binary values are entered into the system, for all possible extensions. However it is not necessarily true for non-binary inputs. Imagine the extension that maps

$$\begin{aligned} \text{AND}\colon \mathbb{R}^2 &\to \mathbb{R} \\ (a,b) &\mapsto a^2b^2 \end{aligned}$$

$$\begin{aligned} \text{OR}\colon \mathbb{R}^2 &\to \mathbb{R} \\ (a,b) &\mapsto a+b-ab. \end{aligned}$$

This is a valid extension as all logic gates or systems of logic gates produce the correct binary output with a given binary input. However over the reals it does not satisfy the function property. The two systems of logic gates:

$$O_1 \in S_{\text{OR}} = \text{A OR B}$$
$$O_2 \in S_{\text{OR}} = \text{(A AND A) OR (B AND B)}$$

Are equivalent as they produce the same truth table for a given binary input. However if we look at their extensions

$$\text{A OR B} = a+b-ab \neq$$
$$\text{(A AND A) OR (B OR B)} = a^4+b^4-a^4b^4$$

which shows it does not satisfy this property as by definition $O_1 \in S_{\text{OR}}$ and $O_2 \in S_{\text{OR}}$. However also by definition all systems in a set must map to the same function, which they do not.

Unfortunately neither of the two possible extensions found in this document have this property, nor do I know if any possible extension has it. However I am sure, if such an extension exists, it is a front-runner for the most natural extension.

It may be helpful to think of a domain extension to be a mapping from a system of logic gates to a complex valued function. If this is the case one can simply verify for all sets of systems. Where o is some desired behaviour (such as the binary truth table for this system) $\forall o \exists S_o$, $\forall x, y \in S_o$, $x(\cdot) = y(\cdot)$. If this is true, the property is verified.

2 Probabilistic approach

The basic laws of probability are well developed and provide a pretty natural extension to the idea of logic gates. It just so happens that if you have two independent events, A and B. The probability of A and B happening is $A*B$. This fits into the idea of logic quite nicely in an unexpected way. If A and B are both guaranteed to happen, then A and B is guaranteed to happen. If B or A is guaranteed to happen, then A and B is guaranteed not to happen. If you continue with this logic and create a sort of truth table for probability. You will find NOT A, A AND B, A OR B, A XOR B, A NAND B, and so on all line up with traditional logic. However the probabilistic definitions of these functions are defined over the domain [0, 1] and can in fact be extended to all complex numbers, however outside the stated range it loses its tie to probability.

Unfortunately in some systems of logic gates, such as in a binary adder, (or simply something like A AND A = 1 no matter what A is), the gates rely on each others output and are so not independent events. As far as I can tell it is impossible to calculate the extended domain with dependant events included without knowing the set of all possible outputs before hand, thought statistics isn't my strong suit so this may be false..

To remedy this all inputs and all gates will be assumed independent (even when they are clearly not). This can be done because for the binary values 1 and 0, whether the gates are dependant is actually irrelevant in the calculation and so it possible to make this shortcut.

2.1 Definitions in the probabilistic approach

Throughout the document I will use S to be set of all possible combinations of logic gates and S_x to be a particular combination of logic gates. For example S_{OR} would be the equivalent of an or gate.

Note to self, look into: if you imagine x here to denote a binary table encoding the logical statement of the set of logic gates. Two natural questions come up.

1. With a continuous extension of the domain of Boolean logic, does every arbitrary valid(no repeated rows, no repeated definitions) rows encode some logical statement(i.e. set of logic gates).

2. If the above statement is true, is there an equivalent binary truth table for every extended truth table.

Secondly I will define P_l to be a mapping between some system of logic gates (defined below), S, and so $P_l(S)$ is the function mapped to by this extension given S. For brevity we can ignore the arguments of the functions, as different systems will end up with functions with a different amount of arguments. For example $P_l(S_{\mathbb{AND}}) = a * b$. Note that a and b are just dummy variables.

$$\begin{aligned} \text{AND}\colon \mathbb{C}^2 &\to \mathbb{C} \\ (a, b) &\mapsto ab. \end{aligned}$$

$$\begin{aligned} \text{OR}\colon \mathbb{C}^2 &\to \mathbb{C} \\ (a, b) &\mapsto a + b - ab. \end{aligned}$$

$$\begin{aligned} \text{XOR}\colon \mathbb{C}^2 &\to \mathbb{C} \\ (a, b) &\mapsto a + b - 2ab. \end{aligned}$$

$$\begin{aligned} \text{NAND}\colon \mathbb{C}^2 &\to \mathbb{C} \\ (a, b) &\mapsto 1 - a - b + 2ab. \end{aligned}$$

$$\begin{aligned} \text{NOT}\colon \mathbb{C} &\to \mathbb{C} \\ a &\mapsto 1 - a. \end{aligned}$$

The definitions above have the property that, for inputs in the range $[0, 1]$, and a subsection of all possible systems, Q the output given by $P_l(Q)$ is equivalent to the probability of the output of the system given a uniformly random input in the system.

2.2 Single variable inverse gates

From the gate definitions it is possible to come up with so called 'inverse gates'. They are very useful in determining the properties of some systems. Over the complex numbers there are infinitely many possible inverses of

course, this will be discussed later as I conjecture it is possible to determine all possible inputs given your output. I have proved it is possible in some cases.

If one has a logic gate L, on output O, and a set of complex numbers $Z|ZLZ = O$. We can call Z the set of single variable inverses to L, or L_i

$$\begin{aligned}
\mathrm{NOT}_i(O) &= \{z\colon O + z = 1\} \\
\mathrm{AND}_i(O) &= \{z\colon z^2 - O = 0\} \\
\mathrm{OR}_i(O) &= \{z\colon z^2 - 2z + O = 0\} \\
\mathrm{XOR}_i(O) &= \{z\colon 2z^2 - 2z + O = 0\} \\
\mathrm{NAND}_i(O) &= \{z\colon 2z^2 - 2z + 1 - O = 0\}
\end{aligned}$$

We can also define single variable inverse functions, which output a single, real, element of this set. These functions will be defined as follows:

$$\begin{aligned}
\mathrm{AND}^{(-1)}\colon \mathbb{C} &\to \mathbb{C} \\
O &\mapsto \sqrt{O}
\end{aligned}$$

$$\begin{aligned}
\mathrm{OR}^{(-1)}\colon \mathbb{C} &\to \mathbb{C} \\
O &\mapsto 1 - \sqrt{1 - O}.
\end{aligned}$$

$$\begin{aligned}
\mathrm{XOR}^{(-1)}\colon \mathbb{C} &\to \mathbb{C} \\
O &\mapsto \frac{1 - \sqrt{1 - 2O}}{2}.
\end{aligned}$$

$$\begin{aligned}
\mathrm{NAND}^{(-1)}\colon \mathbb{C} &\to \mathbb{C} \\
O &\mapsto \frac{1 - \sqrt{-1 - 2O}}{2}
\end{aligned}$$

$$\begin{aligned}
\mathrm{NOT}^{(-1)}\colon \mathbb{C} &\to \mathbb{C} \\
a &\mapsto 1 - O.
\end{aligned}$$

Using this information you can deduce an inverse of any system of logic gates.

Note that the inverse of OR is a quadratic, and I used the real solution that I did for the function because it comes to a nice answer: $\mathrm{OR}^{(-1)}(O) = \mathrm{NOT}(\mathrm{AND}^{-1}(\mathrm{NOT}(O)))$

Also note that the inverse of NOT is NOT.

*Note to self: TODO: one can use functional analysis here, and treat the logic gates as operators and do things like L^2, L^{-1}, $L^{0.5}$. This will allow for some interesting results and easier handling of some systems.

2.3 Multiple gate applications

Imagine a gate as an operator which can take two values and transform it into a single value. What would repeated application of this operator do. Here we take the example of AND. I will do a short proof to show this:

Base case:

$$\mathrm{AND}^{(1)}(a, b) = ab \tag{1}$$

N+1 case:

$$\mathrm{AND}^{(n+1)}(a, b) = \tag{2}$$
$$\mathrm{AND}(\mathrm{AND}^{(n)}(a, b), \mathrm{AND}^{(n)}(a, b)) = \tag{3}$$
$$\mathrm{AND}^{(n)}(a, b)^2 \tag{4}$$

and given repeated squaring is the same as exponentiation by a power of two it is clear that:

$$\mathrm{AND}^{(n)}(a, b) = (ab)^{2^{n-1}} \tag{5}$$

While this works for natural numbers, extending the domain of n is interesting in of itself. Negative integers coincide with repeated applications of the inverse *prove here*, while

$$\begin{aligned}\mathrm{AND}^{(-1)}(a,a) &= \\ (a^2)^{2^{-2}} &= \\ (a)^{2^{-1}} &= \\ \sqrt{a}\end{aligned}$$

gives us the single variable inverse defined earlier.

Interestingly, due to the nature of repeated application of multiple variable functions, the 0th application of the function is equal to the geometric mean of the two inputs. We will fix this in the next section with a unique mathematical structure, with properties allowing true repeated applications like those in the not gate.

All of the gates defined above can be given the same treatment. Below is the formula for the repeated application of the NOT gate.

prove later

$$\mathrm{NOT}^{(n)}(x) = \frac{1}{2} - \left(\frac{1}{2} + x\right) \cdot (-1)^n$$

– section not finished-

2.4 Polynomial encoding

The extension defined by the probabilistic approach can be thought of encoding arbitrary systems of logic gates into polynomials of N variables, where N is the number of inputs to the system. As an example The system of logic gates NOT(AND(NOT(a), OR(a, c))) is mapped to a polynomial as follows by directly using the definitions of the functions.

$$\begin{aligned}S_k &= \mathrm{NOT}(\mathrm{AND}(\mathrm{NOT}(a), \mathrm{OR}(a,c))) \\ S_k &\mapsto 1 - a - c - 2ac + a^2 - a^2c\end{aligned}$$

The most natural question arising from this is "Does there exist a mapping between all sets of logic gates and all integer polynomials of N variables. In more logical terms:

$$\exists k \mid S_k \mapsto P, P \in \mathbb{Z}[x_1, x_2, ..., x_n], n \in \mathbb{N}_+$$

This is false as for all systems of logic gates, their binary truth tables produce binary outputs. This would mean that the polynomial generated will always have a value at 0 or 1 when their variables are 0 or 1. This is not true for all polynomials. However this question can be narrowed to avoid this simple counterargument:

$$\begin{aligned} \exists k \mid S_k \mapsto P, P \in \mathbb{Z}[x_1, x_2, \ldots, x_n], n \in \mathbb{N}_+, & \\ P(0, 0, \ldots, 0) &= 0 \vee 1, \\ P(0, 0, \ldots, 1) &= 0 \vee 1, \\ &\vdots \\ P(1, 1, \ldots, 1) &= 0 \vee 1 \end{aligned}$$

This question is significantly harder to answer. . .