Generating Ternary Logic from One Operation

Lucilla

Abstract

Any n-ary function for all $n \geq 1$ from a three-valued set to itself can be represented in terms of a single binary function. This paper outlines how.

0 Introduction

As is commonly known, both the functions `NAND` and `NOR` have the property that any n-ary Boolean function can be represented in terms of it: that is, both the sets {`NAND`} and {`NOR`} are *functionally complete*. (They are also the only functionally complete singletons.) A proof (for `NAND`) would go something like this:

$$\begin{aligned} \texttt{NOT}\ a &= a\ \texttt{NAND}\ a; \\ a\ \texttt{AND}\ b &= \texttt{NOT}\ (a\ \texttt{NAND}\ b); \\ a\ \texttt{OR}\ b &= (\texttt{NOT}\ a)\ \texttt{NAND}\ (\texttt{NOT}\ b), \end{aligned}$$

and then by reducing every n-ary Boolean function to disjunctive normal form.

The fact that functionally complete singletons for Boolean logic exist implies that the "complexity", the minimal cardinality of a functionally complete set of binary functions, for Boolean logic is 1. A question naturally arises: is the complexity of *any* m-valued logic also 1?

Putting aside the fact that Webb 1935 answers this question affirmatively as early as nearly a century ago, let's imagine how a math-obsessed hermit might tackle the case $m = 3$. Throughout this paper, three-valued logic will be taken to have the values {`r`, `b`, `g`}. It turns out that then the following binary function is sufficient:

$$@^*(a,b) = \begin{cases} \text{if } a \text{ or } b \text{ is } \texttt{r}: & \texttt{b}; \\ \text{else if } a \text{ or } b \text{ is } \texttt{b}: & \texttt{g}; \\ \text{else}: & \texttt{r}, \end{cases}$$

making {$@^*$} functionally complete.

1 Completeness of @*

A technique similar to the disjunctive normal form in Boolean logic will be employed to construct any arbitrary n-ary three-valued function. First, we will need some helper functions, particularly unary ones. Throughout this paper, the unary function $(\mathtt{r} \mapsto x; \mathtt{b} \mapsto y; \mathtt{g} \mapsto z)$ will be denoted as xyz.

It is immediately apparent that @* is the composition of @, the (associative and commutative) min function given by the total order $\mathtt{r} < \mathtt{b} < \mathtt{g}$, i.e.

$$@(a,b) = \begin{cases} \text{if } a \text{ or } b \text{ is } \mathtt{r}: & \mathtt{r}; \\ \text{else if } a \text{ or } b \text{ is } \mathtt{b}: & \mathtt{b}; \\ \text{else:} & \mathtt{g}, \end{cases}$$

with the "rotation", "swizzle", or (this is the best name) "three-way switcheroo" $\mathtt{bgr} = (\mathtt{r} \mapsto \mathtt{b};\ \mathtt{b} \mapsto \mathtt{g};\ \mathtt{g} \mapsto \mathtt{r})$. This is no coïncidence, as both @ and bgr can be reconstructed from @*; indeed, since $x\ @\ x = x$, we have

$$\begin{array}{lcll} \mathtt{bgr}\ x & = & x\ @^*\ x; & \\ \mathtt{grb}\ x & = & \mathtt{bgr}\ \mathtt{bgr}\ x; & \text{(reverse three-way switcheroo);} \\ a\ @\ b & = & \mathtt{grb}\ (a\ @^*\ b). & \end{array}$$

Next, we will need "discarding" or "coërcion" functions; that is, ones whose image has cardinality 2. One of them can be constructed rather easily:

$$\mathtt{rbr}\ x = x\ @\ (\mathtt{bgr}\ x).$$

By pre-switcherooing the argument before passing it to rbr, we can rotate the order of the outputs, and by post-switcherooing, we can change the colors of the outputs. Overall, this allows us to generate any "positively oriented" coërcion function, that is, one where the "odd one out" output is one switcheroo further than the remaining two; e.g. we can generate bbg, but not (yet) ggb.

We can obtain also the remaining ("negatively oriented") coërcion functions using "two-way switcheroos": functions that swap two of the values and keep the last one unchanged. One two-way switcheroo can be obtained as follows:

$$\mathtt{brg}\ x = (\mathtt{bbg}\ x)\ @\ (\mathtt{grg}\ x),$$

(where both of the necessary coërcion functions are positively oriented, and thus already constructible). Three-way switcherooing the output of brg produces the remaining two-way switcheroos, and finally, two-way switcherooing the positively oriented coërcion functions produces the negatively oriented ones.

The only missing unary functions are now the constant functions, the ones whose image has cardinality 1. const r is $x \mapsto (x\ @\ (\mathtt{bgr}\ x)\ @\ (\mathtt{grb}\ x))$, and the other constant functions can be obtained by switcherooing its output.

We are now ready to tackle a general n-ary three-valued function through a strategy analogous to that of the disjunctive normal form in Boolean logic.

Imagine the construction of such a function through a circuit board, analogous to how Boolean functions are implemented in electronics using logic gates, except here each wire can be in one of the three ternary states $\{\mathtt{r}, \mathtt{b}, \mathtt{g}\}$.

Start with the n inputs; $n \geq 1$. Split each input into 3^n branches and collect the n inputs for each branch. Each of these branches will represent one of the 3^n combinations that the n inputs may have. For each branch, add three-way switcheroos in such a way that their desired combination will, after switcherooing, produce all $\mathtt{g}$ values; e.g. for the combination $(\mathtt{r}, \mathtt{g}, \mathtt{g}, \mathtt{b})$, one would switcheroo the inputs twice, zero times, zero times, and once, respectively. Now, for every possible combination of inputs, exactly one of the 3^n branches will have all $\mathtt{g}$ values.

Then, merge all the n inputs in each branch using @. Since @ only produces $\mathtt{g}$ if both (or all, if extended to n arguments, by associativity) of its inputs are $\mathtt{g}$, this means that after merging, exactly one of the 3^n output wires will have the value $\mathtt{g}$. Switcheroo all of them once, so that now exactly one of them will have the value $\mathtt{r}$.

Next, collect the 3^n branches into three sets, one corresponding to those input combinations that should return $\mathtt{r}$, those that should return $\mathtt{b}$, and those that should return $\mathtt{g}$. Merge each of these sets into a single wire using @. Since @ produces $\mathtt{r}$ if at least one of its inputs is $\mathtt{r}$, after merging, exactly one of the three group branches will have the value $\mathtt{r}$; specifically, exactly the group branch that corresponds to the "correct" output will be $\mathtt{r}$. Coërce every group branch with $\mathtt{rgg}$, so that now the other two group branches will be $\mathtt{g}$.

Now all we need is a ternary function that satisfies

$$\begin{aligned} (\mathtt{r}, \mathtt{g}, \mathtt{g}) &\mapsto \mathtt{r}; \\ (\mathtt{g}, \mathtt{r}, \mathtt{g}) &\mapsto \mathtt{b}; \\ (\mathtt{g}, \mathtt{g}, \mathtt{r}) &\mapsto \mathtt{g}, \end{aligned}$$

and $(a, b, c) \mapsto (a \mathbin{@} (\mathtt{brg}\ b))$ does the trick. ■

2 Addendum

The mapping of *polynomials* (as arbitrary-length finite sequences of coefficients) over a field $\mathbb{F}$ to their corresponding polynomial *functions* is injective if e.g. $\mathbb{F} = \mathbb{R}$, but it is never injective if $\mathbb{F}$ is a finite field, since in such fields there is always a power $p > 1$ such that $x^p = x$ for all x, and thus the polynomial $x^p - x$ maps to the same function as the zero polynomial.

However, in a field with a prime number p of elements (which is thus isomorphic to $\mathbb{Z}_p$ with addition and multiplication modulo p), there are p^p polynomials of degree less than p, *and* p^p unary functions, so one would wishfully think that perhaps each such polynomial corresponds bijectively to every possible unary function – and indeed, by Lagrange interpolation, any p points can be interpolated by a polynomial of degree at most $p - 1$.

The case $p = 3$ is particularly neat in how the polynomials correspond to the unary functions we constructed as helper functions throughout this paper. Let $(\mathtt{r}, \mathtt{b}, \mathtt{g})$ correspond to $(0, 1, 2)$. Then there are $3^3 = 27$ polynomials of degree less than 3, and also 27 unary functions:

- The three constant functions (image has cardinality 1) correspond to the three constant polynomials;
- There are six bijective functions (image has cardinality 3), which correspond to the six linear polynomials. Of these:
  - three are positively signed permutations, namely the identity and the two three-way switcheroos; which correspond to the linear polynomials with a leading 1 (e.g. $x + 1$, which corresponds to bgr);
  - the other three are negatively signed permutations, which are the three two-way switcheroos; these correspond to the linear polynomials with a leading 2.
- The remaining 18 functions are "coërcion functions" (image has cardinality 2), which correspond to the 18 quadratic polynomials. Of these:
  - 9 are "negatively oriented"; they correspond to the quadratic polynomials with a leading 1 (e.g. x^2, which corresponds to rbb);
  - the other 9 are "positively oriented"; they correspond to the quadratic polynomials with a leading 2.

References

Webb, Donald Loomis (1935). "Generation of any n-valued logic by one binary operation". In: *Proceedings of the National Academy of Sciences* 21.5, p. 252.