Entangled Functions in Boolean Algebra

by Sven Nilsen, 2018

An entangled function depends on more than one of the same variable for multiple inputs. All entangled functions can be rewritten as a function of fewer variables. In boolean algebra the smallest function type that can be entangled is of type `bool × bool → bool`, because a function of type `bool → bool` takes a single argument and therefore can not be entangled. This means that when a boolean function of two arguments is entangled, it becomes a boolean function of a single argument. To describe which argument is used by the single-argument function, one uses `fst`, `snd`, `nfst` or `nsnd`.

00	false_1	false_2
01	id	{fst, snd}
10	not	{nfst, nsnd}
11	true_1	true_2

The `not` function can be replaced by `id` of the other argument, because if the arguments are equal, it does not matter which argument you pick and so can use `id` on the first argument, but if the arguments unequal, then the arguments always have opposite value. Therefore, the following holds for inequality:

nfst <=> snd
nsnd <=> fst

Since there are four possible arguments of type `bool × bool`, there are `$2^4$ = 16` possible functions. Instead of writing the full truth table for each function, one can use a 4-bit code that shows the output for each input `00`, `01`, `10` and `11`. One can then color the output that holds when inputs are equal in blue and unequal in red.

false_2	0000	false_2	false_2
and	0001	fst	false_2
exc	0010	false_2	fst
fst	0011	fst	fst
rexc	0100	false_2	snd
snd	0101	fst	snd
neq	0110	false_2	true_2
or	0111	fst	fst
nor	1000	fst	false_2
eq	1001	true_2	false_2
nsnd	1010	fst	fst
nrexc	1011	true_2	fst
nfst	1100	nfst	snd
imply	1101	true_2	snd
nand	1110	nfst	true_2
true_2	1111	true_2	true_2

Every boolean function `f` has an inverse `not · f`, so the same table used for equality and inequality can also be used on other functions of type `bool × bool → bool` which returns `true` one or two times. This is the same as the length of the existential path of `f` is 1 or 2, which together with its inverse consists of all non-trivial constraints (0 and 4 which contains no entangled functions).

(f, g)	$\|\exists f\| = \{1, 2\}$	$g = \text{not} \cdot f$

Here are all possible patterns that satisfies the criteria above:

0000	eq	neq	entangled
0000	fst	nfst	function currying
0000	nsnd	snd	function currying
0000	nor	or	constrained
0000	rexc	nrexc	constrained
0000	exc	imply	constrained
0000	and	nand	constrained

Here, "entangled" is used when it is guaranteed that the function can be rewritten with fewer arguments from the type of constraint used. Constrained functions in general might or might not be entangled, but all entangled functions are constrained functions.

Function currying requires some explanation. The `fst` function returns `true` only when the first argument is `true`, so it is the same as calling `f(1) : bool → bool`. Likewise, `nfst` returns `true` only when the first argument is `false`, so it is the same as calling `f(0) : bool → bool`. The same goes for `snd` and `nsnd`, but here the syntax in path semantics is `(f 1)` and `(f 0)` because it curries on the second argument. Function currying is not entangled functions, but can be reduced to a smaller type.

Here is the table for `nor/or`, using black color for non-reducible partial functions:

false_2	0000	false_2	false_2
and	0001	false_2	{and, eq}
exc	0010	false_2	nsnd
fst	0011	false_2	fst
rexc	0100	false_2	nfst
snd	0101	false_2	snd
neq	0110	false_2	{neq, nand}
or	0111	false_2	true_2
nor	1000	true_2	false_2
eq	1001	true_2	{and, eq}
nsnd	1010	true_2	nsnd
nrexc	1011	true_2	nfst
nfst	1100	true_2	nfst
imply	1101	true_2	snd
nand	1110	true_2	{neq, nand}
true_2	1111	true_2	true_2

Notice that 28 out of 32 of the constrained functions are entangled. Because of symmetry, it is expected that this is true for the other constrained functions as well. This implies that there there are `28·4 + 32 = 144` entangled functions of type `bool × bool → bool`, or 56.25% of all constraints (16·16).