Proof Optimization by Uniqueness

by Sven Nilsen, 2019

In this paper I show that there exists a general proof optimization algorithm using uniqueness.

Assume you have 3 boolean variables such that:

$$a \veebar b \veebar c \veebar (\neg a \wedge \neg b \wedge \neg c)$$

This constraint encodes the meaning of uniqueness. One could say that XOR (`⊻`) alone encodes meaning of uniqueness, but it is easier to understand why by starting here. Using De Morgan's law[1]:

$$a \veebar b \veebar c \veebar \neg(a \vee b \vee c)$$

One can also relax it to:

$$(a \veebar b \veebar c) \vee \neg(a \vee b \vee c)$$

This relaxation works because there is no intersection between the two parts:

$$(a \veebar b \veebar c) \wedge \neg(a \vee b \vee c) = \text{false}$$

In programming, the `¬(a ∨ b ∨ c)` part can be thought of as "null". This trick works because one can replace the proof that each variable is `false` with a new variable `null` representing this proof:

$$a \veebar b \veebar c \veebar \text{null} \qquad\qquad \text{null} = \neg(a \vee b \vee c)$$

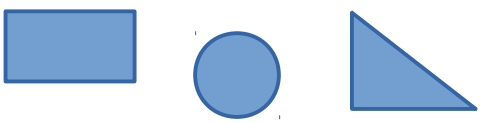
It is now easy to see that `null` can be introduced by adding any new variable:

$$a \veebar b \veebar c \veebar d$$

This is what I mean that XOR alone encodes the meaning of uniqueness.

The XOR operator with n-arguments can also be thought of as a set of sets where each member is the complementary set of all the other members. XOR with n-arguments is the simplest mathematical object that satisfy this set property. This set property is also uniform, meaning that each sub-set of the set has the same property.

Another example of more complex mathematical objects with this property is non-overlapping geometry, like this:



Each figure is a set of points that is complementary to all the other figures.

If I select one of the figures, you can see that no points in the other figures are selected:

What it means that XOR with n-arguments is the simplest mathematical object of this kind, is that it has all the properties we need when talking about uniqueness, without talking about the content inside. On the other hand, geometrical figures has much more information that is irrelevant for uniqueness.

Now I will use XOR with n-arguments to talk about a specific proof optimization algorithm. This algorithm assumes the following ingredients:

- A proof `f : bool$^N$ → bool` to be optimized
- A constraint `g : nat → bool` which returns `true` if the proof assumes XOR for the argument
- Higher Order Operator Overloading[2]

Higher Order Operator Overloading is used with self-referential lambdas[3] to perform partial evaluation of the proof[4]. In this context, we "pretend" that the self-referential lambda is constructed by returning an argument by an index:

s(i : nat) = \(xs : bool^N) = xs[i]

We can "pretend" that this is what the self-referential lambda does, because it simplifies how we think about the optimization algorithm. In an actual implementation of this optimization algorithm, the self-referential lambda refers to a truncated list of arguments for which `g` returns `false`. This is needed to reduce the proof to a more optimized version using same brute force checking. Anyway, although `s` does something different in the actual implementation, it refers correctly here to the argument by the index in the original proof. This is how it works to "pretend" that `s` has a different definition.

For example, if `N = 5` and `g` returns `true` for the first 3 arguments:

g := \(x : nat) = x < 3

The proof can now be optimized by using Higher Order Operator Overloading. For the first 3 arguments which `g` returns `true`, we insert constants. For the two remaining arguments we use `s`:

f(**true**, *false*, *false*, s(3), s(4)) ∧
f(*false*, **true**, *false*, s(3), s(4)) ∧
f(*false*, *false*, **true**, s(3), s(4))

Notice one term for every unique argument.

The magic of Higher Order Operator Overloading is that this becomes a new function which takes a reduced list of arguments for which `g` returns `false`. This new function, when checked for all inputs, preserves truth values from the original proof `f`. It returns `true` for all inputs if and only if the original proof returns `true` for all inputs. The run-time complexity[5] of checking has been reduced:

$O(2^N)$ => $O(2^{N-|g|})$

References:

[1] “De Morgan’s law”
Wikipedia
https://en.wikipedia.org/wiki/De_Morgan%27s_laws

[2] “Higher Order Operator Overloading”
Sven Nilsen, 2018
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading.pdf

[3] “Higher Order Operator Overloading and Self Reference”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading-and-self-reference.pdf

[4] “Higher Order Operator Overloading and Partial Evaluation”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading-and-partial-evaluation.pdf

[5] “Big O notation”
Wikipedia
https://en.wikipedia.org/wiki/Big_O_notation