Function Space

by Sven Nilsen

A program can be represented as a stack containing the state and a list of instructions. For simplicity we assume that all the variables are constant when it starts running and the program can not contain infinite loops. At the start the stack is empty and all the constants are stored in the list of instructions. We can generate all possible instructions and insert random variables to check whether two programs are equal or not. This is called the “function space”.

In the function space, there are operations on a function that takes us from one function to another. A point in function space represents all functions that gives the same output for the same input. The space is discrete for finite programs without loops. It is interesting to know how points in function space relate to each other. In this document I will try to describe many operations and which class they belong to.

Nil Operator

A program that contains no instructions is equal to the nil operator. A distance in function space is defined relative to a program without instructions, where the distance is the smallest amount of instructions for any function at that point. The distance any operator transforms a function depends on the smallest number of instructions required to describe the operator.

Any operator operates on the arguments to a function, that is the part of the program that can not be evaluated before running. A function without arguments got the distance equal to the number of values it returns. A function that takes no arguments and return no arguments is equal to the nil operator.

Identity Operator

A function that takes one argument and returns that argument is equal to the identity operator:

$$f(x)=x$$

If we have several arguments, the identity operator returns the arguments in the same order:

$$f(x,y)=x,y$$

The identity operator belongs to the “invariant operator class”.

Invariant Operator Class

An invariant operator class is one that does not take us to a different point in function space. Every identity operator is an invariant operator. Some functions have invariant operators that are not identity operator.

$$f(x)=x^2,[g(x)=-x]\in f$$

Swap Operator

A swap operator takes two arguments and return them in opposite order:

$$f(x,y)=y,x$$

Invariant Swap Operator Groups

All arguments to functions can be grouped by invariant swapping, that is if 'a, b' are swap invariant and 'b, c' are swap invariant then 'a, c' are also swap invariant. All associative and commutative operators preserve the group when they are included are associative or commutative respectively. That means it does not matter if we consider an associative operator part of the group or the members with respect to swap invariance. The same case is for commutative groups. If we introduce an associative operator into a commutative group, the arguments to the operator is no longer swap invariant.

Linear Operator Groups

When a function is linear in multiple arguments, the argument forms a linear operator group. It can only exist one such group per function. The function is linear to the product of the constants multiplied with the linear arguments. All points in function space given by this product are continuously connected. It means the functions touches each other. They are actually forming a line in function space, but the line has no visual representation as the number of dimensions and the relationships between them is not well defined.

Constraining Operator Class

By constraining the possible values for a given input, we can make two different functions go together in the function space. The constraining equals folding the function space. Equality in function space depends on the constraints, which acts as a context. One way of constraining is to specify an accuracy that if two functions are closer than the accuracy limit, they are considered equal. Another way of constraining is to select an interval. We can also select a repetitive pattern which we consider two functions equal if they intersect each other in that pattern.

Some constrains are exclusive, that is if both are specified, then no functions belong to same point in the function space. The function space is then empty. This leads to the idea that whenever we apply a constraint, we reduce the information in the function space.