Monadic Subsets

by Sven Nilsen, 2024

In this paper I introduce monadic subsets, which is a predicate of two functions that return some optional value in a such way that the first argument is a subset of the second argument. I also generalize monadic subsets for topology and introduce algebraic operators for this calculus.

A monadic subset is a binary operator on functions of the following kind:

f : T → opt[U]

f(x) = none() | some(x') for all `x`

x : T x' : U

Monadic subset `f => g` is a predicate quantifying over the truth table for all `x, y : T`:

f(x)	**g(y)**	**f => g**	**f <=> g**
none()	none()	true	true
none()	some(y')	true	false
some(x')	none()	false	false
some(x')	some(y')	(x == y) => (x' == y')	(x == y) => (x' == y')

Where `f <=> g` is defined as `(f => g) & (g => f)` and is a kind of functional equality.

Monadic subsets are different from normal subsets because the individual values of `x'` and `y'` are dependent on the values of `x` and `y`. In order for `f => g` to be true, they have to match elements one to one where `f` returns some value.

For any `x` and `f => g`, `g(x) == none()` implies that `f(x) == none()`.

For any `x` and `f => g`, `f(x) == some(x')` implies that `g(x) == some(x')`.

`!f` is a shorthand for `f => \none()` which is the same as `f <=> \none()`, in both the sense of monadic subsets and functional equality. Notice that this is a proposition and is not a set inverse.

Monadic subsets are important for understanding analytic properties of geometry and topology. In geometry, the elements have to match one to one exactly. In topology, there has to exist a path from one element to the other in some space.

For example, under the space of translations, `f => g` means that there exists some `h` translating `f` such that `h(f) => g` under no translations. One can also abuse notation and write this as `translations(f) => g`. A path in this sense means some continuous translation of a starting point into a target point. The translation acts uniformly on all points such that there exists a path for every point in `f`. It follows that `translations(f) => g` has the same truth value as `f => translations(g)`.

For a concrete translation `h`, `h(f) => g` implies `f => $h^{-1}$(g)`.

In general, here are some algebraic properties of `=>`:

Reflexivity: f => f

Transitivity: if `f => g` and `g => h` then `f => h`.

Now, considering `=>` as fundamental, it becomes natural to create a operations `+` and `*`:

f(x)	**g(y)**	**f + g**	**f * g**
none()	none()	none()	none()
none()	some(y’)	if x == y { some(y’) } else { none() }	none()
some(x’)	none()	if x == y { some(x’) } else { none() }	none()
some(x’)	some(y’)	if (x == y) & (x’ == y’) { some(x’) } else { none() }	if (x == y) & (x’ == y’) { some(x’) } else { none() }

The following symmetries hold:

(f + g) => (g + f)   (f * g) => (g * f)

The following properties hold:

(f + f) <=> f   (f * f) <=> f

(f + \none()) <=> f   (f * \none()) <=> \none()

When `f` and `g` agrees for all inputs they overlap, the following maps exist:

f => (f + g)   g => (f + g)   (f * g) => f   (f * g) => g

One can also define subtraction:

f(x)	**g(y)**	**f – g**
none()	none()	none()
none()	some(y’)	none()
some(x’)	none()	if x == y { some(x’) } else { none() }
some(x’)	some(y’)	none()

The following property holds:

(f – f) <=> \none()

The following holds, but can not be reversed if `f` and `g` disagrees on overlapping inputs:

((f + g) – g) => f

If `f` and `g` agrees on overlapping inputs, then:

((f + g) – g) <=> f