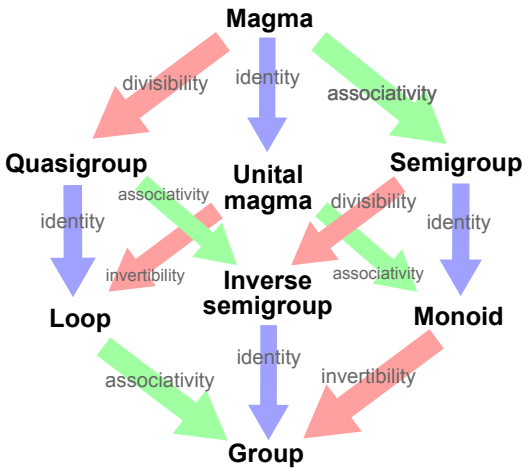
Sums, Products, Exponents, Monoids, Functors, Oh My!

STEVE DOWNEY


Magma
divisibility
identity
associativity
Quasigroup
Unital magma
Semigroup
associativity
divisibility
identity
identity
invertibility
associativity
Inverse semigroup
Loop
Monoid
identity
associativity
invertibility
Group

```cpp
class T {
    // ...
    friend T operator+(T const& lhs, T const& rhs) { /* ... */ }
    // ...
};
```

```
template class T<typename A>
```

```
auto circ(auto&& f, auto&& g) {
    return [=](auto&& x) { return f(g(x)); };
}
```

```haskell
fmap id = id
fmap (g . h) = (fmap g) . (fmap h)
```

std

std

```
std::transform(InputIt         first1,
               InputIt         last1,
               OutputIt        d_first,
               UnaryOperation unary_op);

std::ranges::transform(R&& r, O result, F op, Proj proj = {});

std::optional::transform(F&& f);
```

```cpp
template<class A> optional {
    // ...
    template <class F> constexpr auto and_then(F&& f);
    template <class F> constexpr auto transform(F&& f);
    // ...
};
```

```cpp
// Value categories and moves elided

template <typename Value>
Value evaluate(Lazy<Value> lazy) {
    return lazy.get();
}

template <typename F, typename... Args>
auto lazy(F f, Args... args) -> Lazy<std::invoke_result_t<F, Args...>> {
    co_return std::invoke(f, args...);
}

template <typename Value, typename F>
auto transform(Lazy<Value> l, F f) -> Lazy<std::invoke_result_t<F, Value>> {
    co_return f(evaluate(l));
}

template <typename Value>
auto join(Lazy<Lazy<Value>> l) -> Lazy<Value> {
    co_return evaluate(l);
}

template <typename Value, typename Func>
auto bind(Lazy<Value> l, Func f) -> decltype(f(evaluate(l))) {
    co_return f(evaluate(l));
```