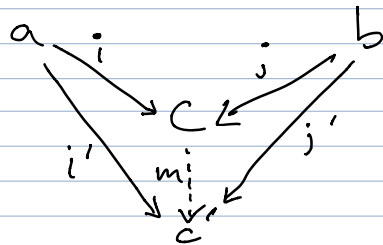


A **coproduct** is an object with two arrows coming to it. Any other object with two morphisms to it has a unique morphism from the coproduct to it. It is the inverse of a product.



$$i' = m \circ i$$

$$j' = m \circ j$$

In terms of set theory, a coproduct can be seen as a union of two sets. A union encompasses all elements of both sets.

Notice that because c can be mapped into c' with the same result as a direct mapping from a itself, c' must preserve a in its entirety.

Therefore the union is a discriminated union. That means if elements in sets a and b overlap, both are kept in c with a tag to discriminate them.

In Haskell, an example of a coproduct is the 'Either' sum type.

```
data Either a b = Left a | Right b
```

This line means an Either can be constructed with the defined 'Left' constructor that takes an 'a', or a 'Right' that takes a 'b'.

To handle an Either type, both types 'a' and types 'b' must be considered. This is called pattern matching.

Example of a pattern matching function

```
f :: Either Int Bool → Bool  
f (Left i) = i > 0  
f (Right b) = b
```

Algebraic Data Types

Does the "product" in category theory actually correspond to a multiplication of types?

Not directly. For instance tuples are not symmetric

$$(a, b) \neq (b, a)$$

Types are not associative

$$((a, b), c) \neq (a, (b, c))$$

But up to isomorphisms they can be