Queremos demonstrar que `3*a = a + a + a` na álgebra de tipos. Como o produto entre dois tipos é uma tupla, essa igualdade pode ser reescrita como `(a, 3) = a + a + a`. Para isso, expressaremos os tipos dos dois lados da equação em Haskell e mostraremos que esses tipos são isomórficos.

Podemos definir o tipo `3` em Haskell como `data Three = One | Two | Three`. O tipo no lado esquerdo na equação, portanto, pode ser expresso como `data Left a = (a, Three)`. O tipo no lado direto na equação, por sua vez, pode ser expresso como `data Right a = First a | Second a | Third a`.

Para mostrarmos que `Left a` e `Right a` são isomórficos, mostraremos que existe uma bijeção `toRight :: Left a -> Right a` entre eles. Podemos expressar essa função em Haskell assim:

```
toRight :: Left a -> Right a
toRight (x, One) = First x
toRight (x, Two) = Second x
toRight (x, Three) = Three x
```

Para provarmos que `toRight` é uma bijeção, mostraremos que ela tem um inverso `toLeft :: Right a -> Left a`, já que toda função invertível é uma bijeção. (Aqui, nos referimos a funções matemáticas, que sempre são totais.) Podemos expressar `toLeft` em Haskell da seguinte forma:

```
toLeft :: Right a -> Left a
toLeft (First x) = (x, One)
toLeft (Second x) = (x, Two)
toLeft (Third x) = (x, Three)
```

Demonstraremos que `toLeft` é o inverso de `toRight` analizando os casos possíveis. Um valor de tipo `Left a` pode ser da forma `(x, One)`, `(x, Two)`, ou `(x, Three)`. Para cada um desses casos, mostraremos que `(toLeft . toRight) x = x` abaixo.

```
(toLeft . toRight) (x, One) = toLeft $ toRight (x, One) = toLeft $ First x = (x, One)
(toLeft . toRight) (x, Two) = toLeft $ toRight (x, Two) = toLeft $ Second x = (x, Two)
(toLeft . toRight) (x, Three) = toLeft $ toRight (x, Three) = toLeft $ Third x = (x, Three)
```

Logo, `toLeft` é o inverso de `toRight`; consequentemente, `toRight` é uma bijeção entre os tipos `Left a` e `Right a` e esses tipos são isomórficos.