We will use universal construction to build a *function object* (an **exponential**). We consider a category $\mathcal{C}$ that has products

1. we have an object b^a that represents a *function* $a \to b$, it is *evaluated* on an *argument* a and produces a *result* b, the pattern is

$$b^a \times a \xrightarrow{\epsilon} b$$

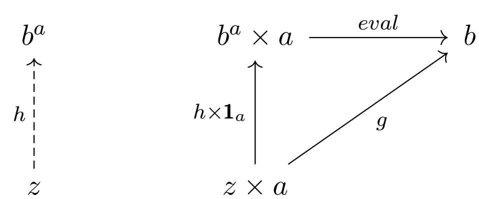
2. suppose we have different candidates for b^a, we want to rank them
3. we take the best one, the one that has a unique morphism from any other candidate

this leads to

Definition 0.1. Let $\mathcal{C}$ have products, the **exponential** of $a, b \in \mathcal{C}$ is

- an object $b^a \in \mathcal{C}$
- a morphism $\text{eval} : b^a \times a \to b$ (*evaluation*)

such that for any $z \in \mathcal{C}$ and $g : z \times a \to b$ there exists a unique morphism $h : z \to b^a$ such that



commutes: $g = \text{eval} \circ (h \times \mathbf{1}_a)$.

The exponential notation comes from considering functions between finite sets. The total number of function $A \to B$ is

$$|B^A| = |B|^{|A|}$$

In the exponential diagram we can see that g (a function of two arguments) is equivalent to b^a (a function of one argument) through h that takes an argument and returns a function.

This is called **Currying**

Example 0.1. *In Haskell these two are equivalent*[1]

```
h :: z -> (a -> b)
g :: (z,a) -> b
```

and we can curry/uncurry with

```
curry :: ((a,b) -> c) -> (a -> (b -> c))
curry f = λx -> (λy -> f (x,y))

uncurry :: (a -> (b -> c)) -> ((a,b) -> c)
uncurry f = λ(x,y) -> ((f x) y)
```

Definition 0.2. A category that has products and exponentials for every pair of objects and a terminal object is a **cartesian closed category** (CCC).

Definition 0.3. A category that has products, coproducts and exponentials for every pair of objects and initial and terminal objects is a **bicartesian closed category** (BCCC).

See [1] Ch.9 and also Category Theory 8.1: Function objects, exponentials - YouTube.

---

[1] B. Milewski, *Category Theory for Programmers* (2019).

---

[1] note that arrows in Haskell associate to the right so parentheses are really not required