What is the meaning of `curry` / `uncurry`?

J.N.Oliveira

March 2012

"Good methods, properly explained, sell themselves."
David Parnas [2]

Curry

From the Haskell Prelude [1]:

$$curry :: ((a, b) \to c) \to (a \to b \to c)$$
$$curry\ f\ a\ b = f\ (a, b)$$
$$uncurry :: (a \to b \to c) \to (a, b) \to c$$
$$uncurry\ f\ (a, b) = f\ a\ b$$

Looking closer at $curry$ and using $\bar{f}$ as abbreviation of $curry\ f$:

$$\overbrace{(\underbrace{curry\ f\ a}_{\bar{f}})b}^{g} \;=\; f\ (a, b) \tag{1}$$

To better see what's going on, we want to turn the applications of functions f and g explicit through the binary operator ap available from `Cp.hs`, through the binary

$$ap :: (a \to b, a) \to b$$
$$ap\ (f, a) = f\ a$$

---

[1] Functions named after the mathematician Haskell Curry (1900-82).

which explicitly applies a function to its argument.

We calculate, taking (1) as starting point:

$$\overbrace{(\underbrace{curry\ f\ a}_{\bar{f}})}^{g}\,b = f\ (a, b)$$

$\equiv$ { definition of g }

$g\ b\ =\ f(a,b)$

$\equiv$ { since $g\ b = ap(g,b)$ }

$ap(g,b)\ =\ f(a,b)$

$\equiv$ { since $g = curry\ f\ a = \bar{f}\ a$ (abbreviation) ; natural-id }

$ap(\bar{f}\ a, id\ b)\ =\ f(a,b)$

$\equiv$ { product of functions: $(f \times g)(x,y) = (f\ x, g\ y)$ }

$ap((\bar{f} \times id)(a,b))\ =\ f(a,b)$

$\equiv$ { composition }

$(ap \cdot (\bar{f} \times id))(a,b)\ =\ f(a,b)$

$\equiv$ { extensional equality (=removing points a and b) }

$ap \cdot (\bar{f} \times id)\ =\ f$

In a diagram, denoting type $B \to C$ by C^B:

$$
\begin{array}{ccc}
C^B & & C^B \times B \xrightarrow{\ ap\ } C \\
\bigg\uparrow{\scriptstyle \bar{f}} & & \bigg\uparrow{\scriptstyle \bar{f}\times id}\quad \nearrow{\scriptstyle f} \\
A & & A \times B
\end{array}
$$

This means that $\bar{f}$ is a solution of the equation $ap \cdot (k \times id)\ =\ f$:

$$k = \bar{f}\quad \Rightarrow \quad ap \cdot (k \times id)\ =\ f$$

It turns out to be the **unique** such solution:

$$k = \bar{f}\quad \Leftrightarrow \quad ap \cdot (k \times id)\ =\ f \qquad\qquad (2)$$

Thus we have a universal property.

Uncurry

Next we introduce variables into $\underbrace{ap \cdot (k \times id)}_{h}$:

$$h(a, b)$$
$$= \qquad \{ \ h = ap \cdot (k \times id) \ \}$$
$$\underbrace{ap \cdot (k \times id)}_{h}(a, b)$$
$$= \qquad \{ \ \text{product } k \times id \ \}$$
$$ap \ (k \ a, b)$$
$$= \qquad \{ \ \text{unfold } ap \ \}$$
$$(k \ a) \ b$$
$$= \qquad \{ \ \text{recall } (k \ a) \ b = uncurry \ k \ (a, b) \ ; \text{abbreviate } uncurry \ k \text{ by } \hat{k} \ \}$$
$$\underbrace{uncurry \ k \ (a, b)}_{\hat{k}(a,b)}$$

Thus $h = \hat{k}$ and (2) can be re-written into:

$$k = \bar{f} \quad \Leftrightarrow \quad \hat{k} = f \tag{3}$$

This means that *curry* and *uncurry* are inverses of each other, leading to isomorphism

$$A \to C^B \quad \cong \quad A \times B \to C$$

which can also be written as

$$(C^B)^A \quad \cong \quad C^{A \times B} \tag{4}$$

The follow up of this can be found in chapter 3 of [1].

References

[1] J.N. Oliveira. Program Design by Calculation, 2008. Draft of textbook in preparation (since 1998). Informatics Department, University of Minho. The following chapters are available from the author's website: *An introduction to pointfree programming*, *Recursion in the pointfree style*, *Why monads matter* and *Quasi-inductive datatypes*.

[2] David Lorge Parnas. Really rethinking "formal methods". *IEEE Computer*, 43(1):28–34, 2010.