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

Lecture note for the
Algebra of Programming Course

CP/1112, 2nd year, LCC+LEI, Univ. of Minho

J.N.Oliveira

March 2012

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

Curry

From the Haskell Prelude [1]:

$$
\begin{array}{l}
\mathit{curry} :: ((a, b) \to c) \to (a \to b \to c) \\
\mathit{curry}\ f\ \ a\ \ b = f\ \ (a, b) \\
\mathit{uncurry} :: (a \to b \to c) \to (a, b) \to c) \\
\mathit{uncurry}\ f\ \ (a, b) = f\ \ a\ \ b
\end{array}
$$

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

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

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

$$
\begin{array}{l}
ap :: (a \to b, a) \to b \\
ap\ \ (f, a) = f\ \ a
\end{array}
$$

[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:

$$
\begin{array}{ll}
& \overbrace{(\underbrace{curry\ f}_{\bar{f}}\ \ a)}^{g}\, b = f\ (a, b) \\
\equiv & \qquad \{ \text{ definition of } g \text{ } \} \\
& g\ b \ = \ f(a, b) \\
\equiv & \qquad \{ \text{ since } g\ b = ap(g, b) \text{ } \} \\
& ap(g, b) \ = \ f(a, b) \\
\equiv & \qquad \{ \text{ since } g = curry\ f\ \ a = \bar{f}\ a \text{ (abbreviation) ; natural-}id \text{ } \} \\
& ap(\bar{f}\ a, id\ b) \ = \ f(a, b) \\
\equiv & \qquad \{ \text{ product of functions: } (f \times g)(x, y) = (f\ x, g\ y) \text{ } \} \\
& ap((\bar{f} \times id)(a, b)) \ = \ f(a, b) \\
\equiv & \qquad \{ \text{ composition } \} \\
& (ap \cdot (\bar{f} \times id))(a, b) \ = \ f(a, b) \\
\equiv & \qquad \{ \text{ extensional equality (=removing points } a \text{ and } b \text{) } \} \\
& ap \cdot (\bar{f} \times id) \ = \ f
\end{array}
$$

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

$$
\begin{tikzcd}
C^B & C^B \times B \arrow[r, "ap"] & C \\
A \arrow[u, "\bar{f}"] & A \times B \arrow[u, "\bar{f} \times id"] \arrow[ur, "f"'] &
\end{tikzcd}
$$

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:

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

Thus we have a universal property.

Uncurry

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

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

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.