Type Fusion

Ralf Hinze

Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, England
ralf.hinze@comlab.ox.ac.uk
http://www.comlab.ox.ac.uk/ralf.hinze/

June 2010

Section 1

Prologue

1.1 Memoisation

Say, you want to memoise the function

$$f : Nat \to V$$

so that it caches previously computed values.

1.1 Table: interface

Given the interface

$$\textbf{data}\,\mathsf{Table}\,v$$

$$\begin{aligned} \mathit{lookup} &: \forall\, v\,.\,\mathsf{Table}\,v \to (\mathit{Nat} \to v) \\ \mathit{tabulate} &: \forall\, v\,.\,(\mathit{Nat} \to v) \to \mathsf{Table}\,v, \end{aligned}$$

we can memoize f as follows

$$\begin{aligned} &\mathit{memo\text{-}f} : \mathit{Nat} \to V \\ &\mathit{memo\text{-}f} = \mathit{lookup}\,(\mathit{tabulate}\,f)\,. \end{aligned}$$

1.1 Table: implementation

$$\mathbf{data}\ Nat \quad = Zero \mid Succ\ Nat$$

$$\mathbf{data}\ \mathsf{Table}\ v = Node\ \{ zero : v, succ : \mathsf{Table}\ v \}$$

$$\begin{aligned} lookup\ (Node\ \{ zero = t \})\ Zero \quad &= t \\ lookup\ (Node\ \{ succ = t \})\ (Succ\ n) &= lookup\ t\ n \end{aligned}$$

$$\begin{aligned} tabulate\ f = Node\ \{ &zero = f\ Zero, \\ &succ = tabulate\ (\lambda n \to f\ (Succ\ n)) \} \end{aligned}$$

1.1 Correctness

$$\mathit{tabulate} : V^{\mathit{Nat}} \cong \mathsf{Table}\, V : \mathit{lookup}$$

1.2 Type firstification

The first-order datatype

$$\textbf{data}\ \mathit{Stack} = \mathit{Empty} \mid \mathit{Push}\,(\mathit{Nat}, \mathit{Stack})$$

is an instance of the second-order datatype

$$\textbf{data}\ \mathsf{List}\ a = \mathit{Nil} \mid \mathit{Cons}\,(a, \mathsf{List}\ a).$$

1.2 Correctness

$$\Lambda\text{-}drop : \mathsf{List}\, Nat \cong Stack : \Lambda\text{-}lift$$

1.3 Type specialisation

Lists of optional values, List (Maybe A) with

$$\textbf{data}\ \mathsf{Maybe}\ a = \mathit{Nothing} \mid \mathit{Just}\ a,$$

can be represented more compactly using the tailor-made

$$\textbf{data}\ \mathsf{Sequ}\ a = \mathit{Done} \mid \mathit{Skip}\ (\mathsf{Sequ}\ a) \mid \mathit{Yield}\ (a, \mathsf{Sequ}\ a).$$

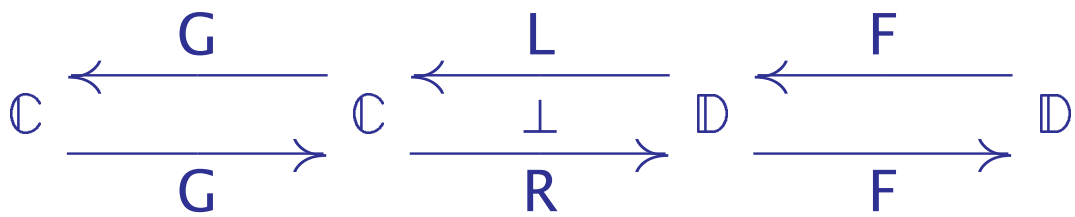
1.3 Correctness

$$\mathsf{List}\,(\mathsf{Maybe}\,A) \cong \mathsf{Sequ}\,A$$

Section 2

Type fusion

2.1 Type fusion



$$\begin{aligned} L\,(\mu F) &\cong \mu G &&\Longleftarrow & L \circ F &\cong G \circ L \\ \nu F &\cong R\,(\nu G) &&\Longleftarrow & F \circ R &\cong R \circ G \end{aligned}$$

2.2 Interlude: adjoint folds

An *adjoint initial fixed-point equation* in the unknown $x : \mathbb{C}(\mathsf{L}\,(\mu\mathsf{F}), A)$ has the syntactic form

$$x \cdot \mathsf{L}\,in \quad = \quad \Psi\, x \ ,$$

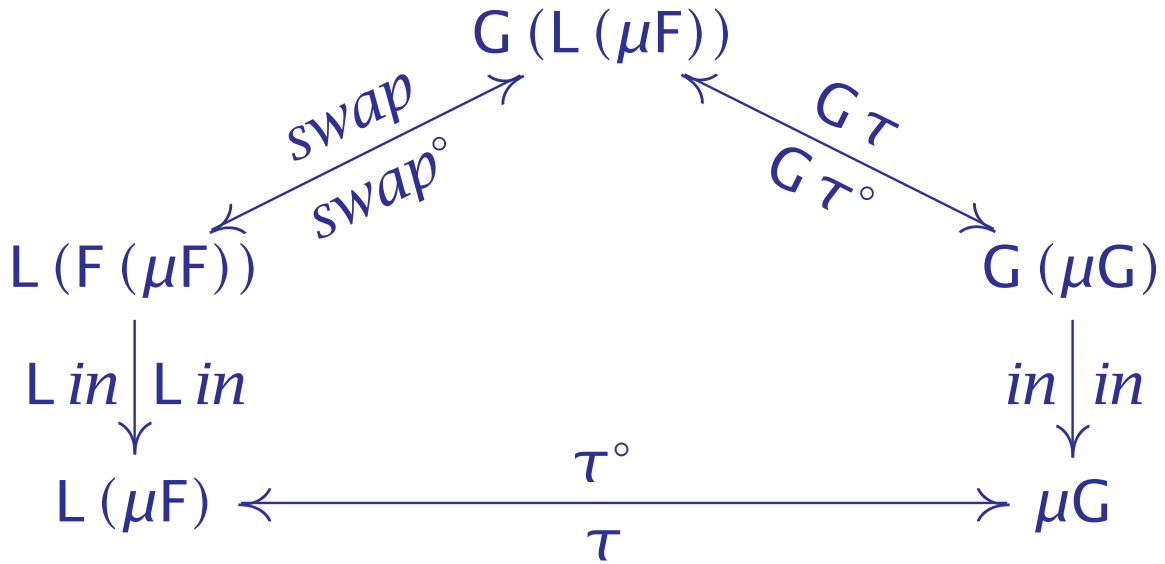
where the base function Ψ has type

$$\Psi : \forall\, X : \mathbb{D} \,.\, \mathbb{C}(\mathsf{L}\,X, A) \to \mathbb{C}(\mathsf{L}\,(\mathsf{F}\,X), A) \ .$$

The unique solution is called an *adjoint fold*.

$$\tau : \mathsf{L}\,(\mu\mathsf{F}) \cong \mu\mathsf{G} \quad \Longleftarrow \quad \mathit{swap} : \mathsf{L} \circ \mathsf{F} \cong \mathsf{G} \circ \mathsf{L}$$

2.3 Definition of τ and τ°



$$\tau \cdot \mathsf{L}\, in = in \cdot \mathsf{G}\, \tau \cdot swap \qquad \text{and} \qquad \tau^\circ \cdot in = \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, \tau^\circ$$

2.3 Proof of $\tau \cdot \tau^\circ = id_{\mu \mathsf{G}}$

$$
\begin{array}{ll}
 & (\tau \cdot \tau^\circ) \cdot in \\
= & \quad \{ \text{ definition of } \tau^\circ \ \} \\
 & \tau \cdot \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, \tau^\circ \\
= & \quad \{ \text{ definition of } \tau \ \} \\
 & in \cdot \mathsf{G}\, \tau \cdot swap \cdot swap^\circ \cdot \mathsf{G}\, \tau^\circ \\
= & \quad \{ \text{ inverses } \} \\
 & in \cdot \mathsf{G}\, \tau \cdot \mathsf{G}\, \tau^\circ \\
= & \quad \{ \mathsf{G} \text{ functor } \} \\
 & in \cdot \mathsf{G}\, (\tau \cdot \tau^\circ)
\end{array}
$$

The equation $x \cdot in = in \cdot \mathsf{G}\, x$ has a unique solution. Since id is also a solution, the result follows.

2.3 Proof of $\tau^\circ \cdot \tau = id_{\mathsf{L}\,(\mu \mathsf{F})}$

$$
\begin{array}{ll}
 & (\tau^\circ \cdot \tau) \cdot \mathsf{L}\, in \\
= & \quad \{ \text{ definition of } \tau \ \} \\
 & \tau^\circ \cdot in \cdot \mathsf{G}\, \tau \cdot swap \\
= & \quad \{ \text{ definition of } \tau^\circ \ \} \\
 & \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, \tau^\circ \cdot \mathsf{G}\, \tau \cdot swap \\
= & \quad \{ \ \mathsf{G} \text{ functor } \} \\
 & \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, (\tau^\circ \cdot \tau) \cdot swap
\end{array}
$$

Again, $x \cdot \mathsf{L}\, in = \mathsf{L}\, in \cdot swap^\circ \cdot \mathsf{G}\, x \cdot swap$ has a unique solution.
And again, id is also solution, which implies the result.

Section 3

Applications

3.1 Recall: type firstification

The first-order datatype

$$\textbf{data}\ \mathit{Stack} = \mathit{Empty} \mid \mathit{Push}\,(\mathit{Nat}, \mathit{Stack})$$

is an instance of the second-order datatype

$$\textbf{data}\ \mathsf{List}\ a = \mathit{Nil} \mid \mathit{Cons}\,(a, \mathsf{List}\ a).$$

Correctness:

$$\Lambda\text{-}\mathit{drop} : \mathsf{List}\ \mathit{Nat} \cong \mathit{Stack} : \Lambda\text{-}\mathit{lift}.$$

3.1 Speaking categorically

$$\mathsf{App}_{Nat}\,(\mu\mathsf{LIST}) \cong \mu\mathsf{Stack}$$

$$\Longleftarrow$$

$$\mathsf{App}_{Nat} \circ \mathsf{LIST} \cong \mathsf{Stack} \circ \mathsf{App}_{Nat}$$

where

$$\mathsf{App}_X : \mathbb{C}^{\mathbb{D}} \to \mathbb{C}$$
$$\mathsf{App}_X\,\mathsf{F} = \mathsf{F}\,X$$
$$\mathsf{App}_X\,\mathbb{\alpha} = \mathbb{\alpha}\,X$$

is ‘type application’.

3.2 Recall: type specialisation

Lists of optional values, List ∘ Maybe with

$$\textbf{data}\ \mathsf{Maybe}\ a = \mathit{Nothing} \mid \mathit{Just}\ a,$$

can be represented more compactly using the tailor-made

$$\textbf{data}\ \mathsf{Sequ}\ a = \mathit{Done} \mid \mathit{Skip}\ (\mathsf{Sequ}\ a) \mid \mathit{Yield}\ (a, \mathsf{Sequ}\ a).$$

Correctness:

$$\mathsf{List} \circ \mathsf{Maybe} \cong \mathsf{Sequ}.$$

3.2 Speaking categorically

$$\mathsf{Pre}_{\mathsf{Maybe}}\,(\mu\mathsf{LIST}) \cong \mu\mathsf{SEQU}$$

$$\Longleftarrow$$

$$\mathsf{Pre}_{\mathsf{Maybe}} \circ \mathsf{LIST} \cong \mathsf{SEQU} \circ \mathsf{Pre}_{\mathsf{Maybe}}$$

where

$$\mathsf{Pre}_{\mathsf{J}} : \mathbb{E}^{\mathbb{D}} \to \mathbb{E}^{\mathbb{C}}$$
$$\mathsf{Pre}_{\mathsf{J}}\,\mathsf{F} = \mathsf{F} \circ \mathsf{J}$$
$$\mathsf{Pre}_{\mathsf{J}}\,\mathbb{\alpha} = \mathbb{\alpha} \circ \mathsf{J}$$

is pre-composition, also written $\mathbb{E}^{\mathsf{J}}$.

3.3 Recall: memoisation

Functions from the natural numbers

$$\textbf{data}\ \mathit{Nat} = \mathit{Zero} \mid \mathit{Succ}\ \mathit{Nat}$$

can be memoised using streams

$$\textbf{data}\ \mathsf{Table}\ \mathit{val} = \mathit{Node}\ \{\mathit{zero} : \mathit{val}, \mathit{succ} : \mathsf{Table}\ \mathit{val}\}.$$

Correctness:

$$(-)^{\mathit{Nat}} \quad \cong \quad \mathsf{Table}$$

3.3 Speaking categorically

$$\nu\mathsf{TABLE} \cong \mathsf{Exp}\,(\mu\mathsf{Nat})$$

$$\Longleftarrow$$

$$\mathsf{TABLE} \circ \mathsf{Exp} \cong \mathsf{Exp} \circ \mathsf{Nat}$$

where

$$\begin{aligned} \mathsf{Exp} &: \mathbb{C} \to (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \\ \mathsf{Exp}\,K &= \Lambda\, V \,.\, V^K \\ \mathsf{Exp}\,f &= \Lambda\, V \,.\, V^f \end{aligned}$$

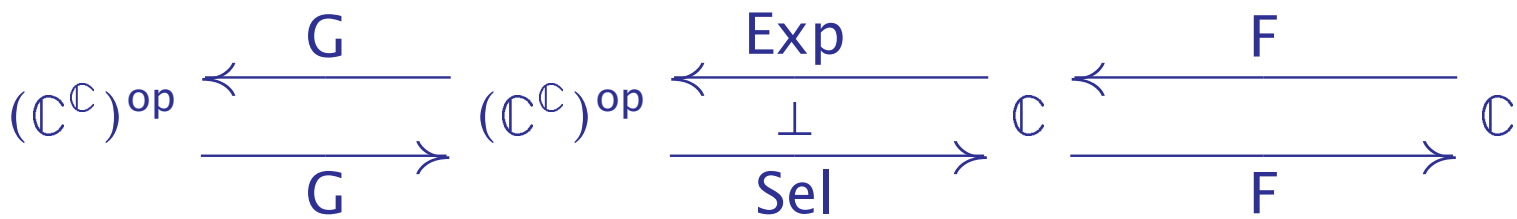
is curried (!) exponentiation.

3.3 Laws of exponentials

$$
\begin{aligned}
V^0 &\cong 1 \\
V^1 &\cong V \\
V^{A+B} &\cong V^A \times V^B \\
V^{A\times B} &\cong (V^B)^A
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{Exp}\,0 &\cong \mathsf{K}\,1 \\
\mathsf{Exp}\,1 &\cong \mathsf{Id} \\
\mathsf{Exp}\,(A+B) &\cong \mathsf{Exp}\,A \mathbin{\dot{\times}} \mathsf{Exp}\,B \\
\mathsf{Exp}\,(A\times B) &\cong \mathsf{Exp}\,A \circ \mathsf{Exp}\,B
\end{aligned}
$$

3.3 Exp is a left adjoint



3.3 Deriving the right adjoint

$$
\begin{aligned}
& (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}}(\mathsf{Exp}\,A, B) \\
\cong \quad & \{ \text{ definition of } -^{\mathsf{op}} \} \\
& \mathbb{C}^{\mathbb{C}}(B, \mathsf{Exp}\,A) \\
\cong \quad & \{ \text{ natural transformation as an end } \} \\
& \forall\, X : \mathbb{C} \,.\, \mathbb{C}(B\,X, \mathsf{Exp}\,A\,X) \\
\cong \quad & \{ \text{ definition of } \mathsf{Exp} \} \\
& \forall\, X : \mathbb{C} \,.\, \mathbb{C}(B\,X, X^A) \\
\cong \quad & \{ -\times Y \dashv (-)^Y \text{ and } Y \times Z \cong Z \times Y \} \\
& \forall\, X : \mathbb{C} \,.\, \mathbb{C}(A, X^{B\,X}) \\
\cong \quad & \{ \text{ the functor } \mathbb{C}(A, -) \text{ preserves ends } \} \\
& \mathbb{C}(A, \forall\, X : \mathbb{C} \,.\, X^{B\,X}) \\
\cong \quad & \{ \text{ define } \mathsf{Sel}\,B = \forall\, X : \mathbb{C} \,.\, X^{B\,X} \} \\
& \mathbb{C}(A, \mathsf{Sel}\,B)
\end{aligned}
$$

- Since Exp is a contravariant functor, τ and *swap* live in an opposite category.

$$(\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \underset{\mathsf{G}}{\overset{\mathsf{G}}{\leftrightarrows}} (\mathbb{C}^{\mathbb{C}})^{\mathsf{op}} \underset{\mathsf{Sel}}{\overset{\mathsf{Exp}}{\underset{\longrightarrow}{\overset{\longleftarrow}{\bot}}}} \mathbb{C} \underset{\mathsf{F}}{\overset{\mathsf{F}}{\leftrightarrows}} \mathbb{C}$$

- Type fusion in terms of arrows in $\mathbb{C}^{\mathbb{C}}$:

$$\tau : \nu\mathsf{G} \cong \mathsf{Exp}\,(\mu\mathsf{F}) \quad \Longleftarrow \quad \mathit{swap} : \mathsf{G} \circ \mathsf{Exp} \cong \mathsf{Exp} \circ \mathsf{F}.$$

- The isomorphism $\tau : \nu\mathsf{G} \mathrel{\dot{\to}} \mathsf{Exp}\,(\mu\mathsf{F})$ is a curried *look-up* function that maps a memo table to an exponential.
- The inverse $\tau^{\circ} : \mathsf{Exp}\,(\mu\mathsf{F}) \mathrel{\dot{\to}} \nu\mathsf{G}$ is a transformation that *tabulates* a given exponential.

Section 4

Epilogue

4.0 Summary and related work

Summary:

- Type fusion generalises
  - type firstification,
  - type specialisation,
  - memoisation or tabulation.
- Adjunctions play a central role.

Related work:

- Backhouse, Bijsterveld, van Geldrop, van der Woude, *Categorical fixed point calculus*. CTCS '95.
- Hinze, *Adjoint folds and unfolds*. MPC '10.