The Arrow Calculus
(Technical Report)

Sam Lindley Philip Wadler Jeremy Yallop

Abstract

We introduce the arrow calculus, a metalanguage for manipulating Hughes's arrows with close relations both to Moggi's metalanguage for monads and to Paterson's arrow notation.

1 Introduction

Arrows [Hughes, 2000] generalise the *monads* of Moggi [1991] and the *idioms* of McBride and Paterson [2008]. They are closely related to *Freyd categories*, discovered independently from Hughes by Power, Robinson and Thielecke [Power and Robinson, 1997, Power and Thielecke, 1999]. Arrows enjoy a wide range of applications, including parsers and printers [Jansson and Jeuring, 1999], web interaction [Hughes, 2000], circuits [Paterson, 2001], graphic user interfaces [Courtney and Elliott, 2001], and robotics [Hudak et al., 2003].

Formally, arrows are defined by extending simply-typed lambda calculus with three constants satisfying nine laws. And here is where the problems start. While some of the laws are easy to remember, others are not. Further, arrow expressions written with these constants use a 'pointless' style of expression that can be hard to read and to write. (Not to mention that 'pointless' is the last thing arrows should be.)

Fortunately, Paterson [2001] introduced a notation for arrows that is easier to read and to write, and in which some arrow laws may be directly expressed. But for all its benefits, Paterson's notation is only a partial solution. It simply abbreviates terms built from the three constants, and there is no claim that its laws are adequate for all reasoning with arrows. Syntactic sugar is an apt metaphor: it sugars the pill, but the pill still must be swallowed.

Here we define the *arrow calculus*, which closely resembles both Paterson's notation for arrows and Moggi's metalanguage for monads. Instead of augmenting simply typed lambda calculus with three constants and nine laws, we augment it with four constructs satisfying five laws. Two of these constructs resemble function abstraction and application, and satisfy beta and eta laws. The remaining two constructs resemble the unit and bind of a monad, and satisfy left unit, right unit, and associativity laws. So instead of nine (somewhat idiosyncratic) laws, we have five laws that fit two well-known patterns.

The arrow calculus is equivalent to the classic formulation. We give a translation of the four constructs into the three constants, and show the five laws follow from the nine. Conversely, we also give a translation of the three constants into the four constructs, and show the nine laws follow from the five. Hence, the arrow calculus is no mere syntactic sugar. Instead of understanding it by translation into the three constants, we can understand the three constants by translating them to it!

Elsewhere, we have already applied the arrow calculus to elucidate the connections between idioms, arrows, and monads Lindley et al. [2008]. Arrow calculus was not the main focus of that paper, where it was a tool to an end, and that paper has perhaps too terse a description of the calculus. This paper was in fact written before the other, and we hope provides a readable introduction to the arrow calculus.

Our notation is a minor syntactic variation of Paterson's notation, and Paterson's paper contains essentially the same laws we give here. So what is new?

- Paterson translates his notation into classic arrows, and shows the five laws follow from the nine (Soundness). We are the first to give the inverse translation, and show that the nine laws follow from the five (Completeness).

Completeness isn't just a nicety: Paterson regards his notation as syntactic sugar for the classic arrows; completeness lets us claim our calculus can supplant classic arrows.

- We are also the first to publish concise formal type rules. The type rules are unusual in that they involve two contexts, one for variables bound by ordinary lambda abstraction and one for variables bound by arrow abstraction. Discovering the rules greatly improved our understanding of arrows.

  Or rather, we should say *rediscovering*. It turns out that the type rules were known to Paterson, and he used them to implement the arrow notation extension to the Glasgow Haskell Compiler. But Paterson never published the type rules; he explained to us that "Over the years I spent trying to get the arrow notation published, I replaced formal rules with informal descriptions because referees didn't like them." We are glad to help the formal rules finally into print.

- We show the two translations from classic arrows to arrow calculus and back are exactly inverse, providing an *equational correspondence* in the sense of Sabry and Felleisen [1993].

  The reader's reaction may be to say, 'Of course the translations are inverses, how could they be otherwise?' But in fact the more common situation is for forward and backward translations to compose to give an isomorphism (category theorists call this an *equivalence* of categories), rather than compose to give the identity on the nose (an *isomorphism* of categories). Lindley et al. [2008] gives forward and backward translations between variants of idioms, arrows, and monads, and only some turn out to be equational correspondences; we had to invent a more general notion of *equational equivalence* to characterize the others.

- The first fruit of our new calculus is to reveal a redundancy: the nine classic arrow laws can be reduced to eight. Notation alone was not adequate to lead to this discovery; it flowed from our attempts to show the translations between classic arrows and arrow calculus preserve the laws.

- The arrow calculus has already proven useful in practice. It enabled us to clarify the relationship between idioms, arrows and monads [Lindley et al., 2008]. Further, it provided the inspiration for the categorical semantics of arrows [Atkey, 2008].

The rest of this report is organized as follows. Section 2 reviews the classic formulation of arrows. Section 3 introduces the arrow calculus. Section 4 translates the arrow calculus into classic arrows, and vice versa, showing that the laws of each can be derived from the other. Section 5 outlines two specialisations of arrow calculus: *static arrows* and *higher-order arrows*. Section 6 outlines a strongly normalising and confluent rewriting theory for arrow calculus. Section 7 highlights some of the fruits of our work on the arrow calculus.

2 Classic arrows

We refer to the classic presentation of arrows as classic arrows, and to our new metalanguage as the arrow calculus.

The core of both is an entirely pedestrian simply-typed lambda calculus with products and functions, as shown in Figure 1. Let A, B, C range over types, L, M, N range over terms, and Γ, Δ range over environments. A type judgment $\Gamma \vdash M : A$ indicates that in environment Γ term M has type A. We use a Curry formulation, eliding types from terms. Products and functions satisfy beta and eta laws. (The laws define an equational judgement between well-typed terms. Each law $M = N$ is shorthand for $\Gamma \vdash M = N : A$ for all Γ, A such that $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$. The equational judgement is defined as the contextual and equivalence closure of the laws.)

Classic arrows extends the core lambda calculus with one type and three constants satisfying nine laws, as shown in Figure 2. The type $A \rightsquigarrow B$ denotes a computation that accepts a value of type A and returns a value of type B, possibly performing some side effects. The three constants are: *arr*, which promotes a

Syntax

$$\begin{array}{lll} \text{Types} & A, B, C & ::= \quad X \mid A{\times}B \mid A \to B \\ \text{Terms} & L, M, N & ::= \quad x \mid \langle M, N\rangle \mid \mathsf{fst}\ L \mid \mathsf{snd}\ L \mid \lambda x.\, N \mid L\ M \\ \text{Environments} & \Gamma, \Delta & ::= \quad x_1 : A_1, \ldots, x_n : A_n \end{array}$$

Types

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}$$

$$\frac{\begin{array}{l}\Gamma \vdash M : A \\ \Gamma \vdash N : B\end{array}}{\Gamma \vdash \langle M, N\rangle : A{\times}B} \qquad \frac{\Gamma \vdash L : A{\times}B}{\Gamma \vdash \mathsf{fst}\ L : A} \qquad \frac{\Gamma \vdash L : A{\times}B}{\Gamma \vdash \mathsf{snd}\ L : B}$$

$$\frac{\Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x.\, N : A \to B} \qquad \frac{\begin{array}{l}\Gamma \vdash L : A \to B \\ \Gamma \vdash M : A\end{array}}{\Gamma \vdash L\ M : B}$$

Laws

$$\begin{array}{lrcl} (\beta_1^{\times}) & \mathsf{fst}\ \langle M, N\rangle & = & M \\ (\beta_2^{\times}) & \mathsf{snd}\ \langle M, N\rangle & = & N \\ (\eta^{\times}) & \langle \mathsf{fst}\ L, \mathsf{snd}\ L\rangle & = & L \\ (\beta^{\to}) & (\lambda x.\, N)\ M & = & N[x := M] \\ (\eta^{\to}) & \lambda x.\, (L\ x) & = & L \end{array}$$

Figure 1: Lambda calculus

function to a pure arrow with no side effects; $\ggg$, which composes two arrows; and *first*, which extends an arrow to act on the first component of a pair leaving the second component unchanged. We allow infix notation as usual, writing $M \ggg N$ in place of $(\ggg)\ M\ N$.

The figure defines ten auxiliary functions, all of which are standard. The identity function *id*, selector *fst*, associativity *assoc*, function composition $f \cdot g$, and product bifunctor $f \times g$ are required for the nine laws. Functions *dup* and *swap* are used to define *second*, which is like *first* but acts on the second component of a pair, and $f \mathbin{\&\&\&} g$, which applies arrows f and g to the same argument and pairs the results. We also define the selector *snd*.

The nine laws state that arrow composition has a left and right unit ($\rightsquigarrow_1$,$\rightsquigarrow_2$), arrow composition is associative ($\rightsquigarrow_3$), composition of functions promotes to composition of arrows ($\rightsquigarrow_4$), *first* on pure functions rewrites to a pure function ($\rightsquigarrow_5$), *first* is a homomorphism for composition ($\rightsquigarrow_6$), *first* commutes with a pure function that is the identity on the first component of a pair ($\rightsquigarrow_7$), and *first* pushes through promotions of *fst* and *assoc* ($\rightsquigarrow_8$,$\rightsquigarrow_9$).

Every arrow of interest comes with additional operators, which perform side effects or combine arrows in other ways (such as choice or parallel composition). The story for these additional operators is essentially the same for classic arrows and the arrow calculus, so we say little about them.

3 The arrow calculus

Arrow calculus extends the core lambda calculus with four constructs satisfying five laws, as shown in Figure 3. As before, the type $A \rightsquigarrow B$ denotes a computation that accepts a value of type A and returns a value of type B, possibly performing some side effects.

We now have two syntactic categories. Terms, as before, are ranged over by L, M, N, and commands are ranged over by P, Q, R. In addition to the terms of the core lambda calculus, there is one new term form:

Syntax

$$\begin{array}{llll}
\text{Types} & A, B, C & ::= & \cdots \mid A \rightsquigarrow B \\
\text{Terms} & L, M, N & ::= & \cdots \mid \mathit{arr} \mid (\ggg) \mid \mathit{first}
\end{array}$$

Types

$$\begin{array}{rcl}
\mathit{arr} & : & (A \to B) \to (A \rightsquigarrow B) \\
(\ggg) & : & (A \rightsquigarrow B) \to (B \rightsquigarrow C) \to (A \rightsquigarrow C) \\
\mathit{first} & : & (A \rightsquigarrow B) \to (A{\times}C \rightsquigarrow B{\times}C)
\end{array}$$

Definitions

$$\begin{array}{rcl}
\mathit{id} & : & A \to A \\
\mathit{id} & = & \lambda x.\, x \\[4pt]
\mathit{fst} & : & A{\times}B \to A \\
\mathit{fst} & = & \lambda z.\, \mathsf{fst}\ z \\[4pt]
\mathit{snd} & : & A{\times}B \to B \\
\mathit{snd} & = & \lambda z.\, \mathsf{snd}\ z \\[4pt]
\mathit{assoc} & : & (A{\times}B){\times}C \to A{\times}(B{\times}C) \\
\mathit{assoc} & = & \lambda z.\, \langle \mathsf{fst}\ (\mathsf{fst}\ z), \langle \mathsf{snd}\ (\mathsf{fst}\ z), \mathsf{snd}\ z \rangle\rangle \\[4pt]
(\cdot) & : & (B \to C) \to (A \to B) \to (A \to C) \\
(\cdot) & = & \lambda f.\, \lambda g.\, \lambda x.\, f\ (g\ x)
\end{array}$$

$$\begin{array}{rcl}
(\times) & : & (A \to C) \to (B \to D) \to (A{\times}B \to C{\times}D) \\
(\times) & = & \lambda f.\, \lambda g.\, \lambda z.\, (f\ (\mathsf{fst}\ z), g\ (\mathsf{snd}\ z)) \\[4pt]
\mathit{dup} & : & A \to A{\times}A \\
\mathit{dup} & = & \lambda x.\, (x, x) \\[4pt]
\mathit{swap} & : & A{\times}B \to B{\times}A \\
\mathit{swap} & = & \lambda z.\, \langle \mathsf{snd}\ z, \mathsf{fst}\ z \rangle \\[4pt]
\mathit{second} & : & (A \rightsquigarrow B) \to (C{\times}A \rightsquigarrow C{\times}B) \\
\mathit{second} & = & \lambda f.\, \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ f \ggg \mathit{arr}\ \mathit{swap} \\[4pt]
(\&\&\&) & : & (C \rightsquigarrow A) \to (C \rightsquigarrow B) \to (C \rightsquigarrow A{\times}B) \\
(\&\&\&) & = & \lambda f.\, \lambda g.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ f \ggg \mathit{second}\ g
\end{array}$$

Laws

$$\begin{array}{lrcl}
(\rightsquigarrow_1) & \mathit{arr}\ \mathit{id} \ggg f & = & f \\
(\rightsquigarrow_2) & f \ggg \mathit{arr}\ \mathit{id} & = & f \\
(\rightsquigarrow_3) & (f \ggg g) \ggg h & = & f \ggg (g \ggg h) \\
(\rightsquigarrow_4) & \mathit{arr}\ (g \cdot f) & = & \mathit{arr}\ f \ggg \mathit{arr}\ g \\
(\rightsquigarrow_5) & \mathit{first}\ (\mathit{arr}\ f) & = & \mathit{arr}\ (f \times \mathit{id}) \\
(\rightsquigarrow_6) & \mathit{first}\ (f \ggg g) & = & \mathit{first}\ f \ggg \mathit{first}\ g \\
(\rightsquigarrow_7) & \mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \times g) & = & \mathit{arr}\ (\mathit{id} \times g) \ggg \mathit{first}\ f \\
(\rightsquigarrow_8) & \mathit{first}\ f \ggg \mathit{arr}\ \mathit{fst} & = & \mathit{arr}\ \mathit{fst} \ggg f \\
(\rightsquigarrow_9) & \mathit{first}\ (\mathit{first}\ f) \ggg \mathit{arr}\ \mathit{assoc} & = & \mathit{arr}\ \mathit{assoc} \ggg \mathit{first}\ f
\end{array}$$

Figure 2: Classic arrows

Syntax

$$\begin{array}{llll} \text{Types} & A, B, C & ::= & \cdots \mid A \rightsquigarrow B \\ \text{Terms} & L, M, N & ::= & \cdots \mid \lambda^{\bullet} x.\, Q \\ \text{Commands} & P, Q, R & ::= & L \bullet M \mid [M] \mid \mathsf{let}\ x = P\ \mathsf{in}\ Q \end{array}$$

Types

$$\frac{\Gamma;\ x : A \vdash Q \,!\, B}{\Gamma \vdash \lambda^{\bullet} x.\, Q : A \rightsquigarrow B} \qquad \frac{\begin{array}{c}\Gamma \vdash L : A \rightsquigarrow B \\ \Gamma, \Delta \vdash M : A\end{array}}{\Gamma;\ \Delta \vdash L \bullet M \,!\, B}$$

$$\frac{\Gamma,\ \Delta \vdash M : A}{\Gamma;\ \Delta \vdash [M] \,!\, A} \qquad \frac{\begin{array}{c}\Gamma;\ \Delta \vdash P \,!\, A \\ \Gamma;\ \Delta,\ x : A \vdash Q \,!\, B\end{array}}{\Gamma;\ \Delta \vdash \mathsf{let}\ x = P\ \mathsf{in}\ Q \,!\, B}$$

Laws

$$\begin{array}{lrcl} (\beta^{\rightsquigarrow}) & (\lambda^{\bullet} x.\, Q) \bullet M & = & Q[x := M] \\ (\eta^{\rightsquigarrow}) & \lambda^{\bullet} x.\, (L \bullet x) & = & L \\ (\text{left}) & \mathsf{let}\ x = [M]\ \mathsf{in}\ Q & = & Q[x := M] \\ (\text{right}) & \mathsf{let}\ x = P\ \mathsf{in}\ [x] & = & P \\ (\text{assoc}) & \mathsf{let}\ y = (\mathsf{let}\ x = P\ \mathsf{in}\ Q)\ \mathsf{in}\ R & = & \mathsf{let}\ x = P\ \mathsf{in}\ (\mathsf{let}\ y = Q\ \mathsf{in}\ R) \end{array}$$

Figure 3: Arrow calculus

arrow abstraction $\lambda^{\bullet} x.\, Q$. There are three command forms: arrow application $L \bullet M$, arrow unit $[M]$ (which resembles unit in a monad), and arrow bind $\mathsf{let}\ x = P\ \mathsf{in}\ Q$ (which resembles bind in a monad).

In addition to the term typing judgment

$$\Gamma \vdash M : A.$$

we now also have a command typing judgment

$$\Gamma;\ \Delta \vdash P \,!\, A.$$

An important feature of the arrow calculus is that the command type judgment has two environments, Γ and Δ, where variables in Γ come from ordinary lambda abstractions $\lambda x.\, N$, while variables in Δ come from arrow abstractions $\lambda^{\bullet} x.\, Q$.

We will give a translation of commands to classic arrows, such that a command P satisfying the judgment

$$\Gamma;\ \Delta \vdash P \,!\, A$$

translates to a term $[\![P]\!]_{\Delta}$ satisfying the judgment

$$\Gamma \vdash [\![P]\!]_{\Delta} : \Delta \rightsquigarrow A.$$

That is, the command P denotes an arrow, taking argument of type Δ and returning a result of type A. We explain this translation further in Section 4.

Here are the type rules for the four constructs. Arrow abstraction converts a command into a term.

$$\frac{\Gamma;\ x : A \vdash Q \,!\, B}{\Gamma \vdash \lambda^{\bullet} x.\, Q : A \rightsquigarrow B}$$

Arrow abstraction closely resembles function abstractions, save that the body Q is a command (rather than a term) and the bound variable x goes into the second environment (separated from the first by a semicolon).

Conversely, arrow application embeds a term into a command.

$$\frac{\Gamma \vdash L : A \rightsquigarrow B \qquad \Gamma, \Delta \vdash M : A}{\Gamma; \Delta \vdash L \bullet M \; ! \; B}$$

Arrow application closely resembles function application. The arrow to be applied is denoted by a term, not a command; this is because there is no way to apply an arrow that is itself yielded by another arrow. This is why we distinguish two environments, Γ and Δ: variables in Γ may be used to compute arrows that are applied to arguments, but variables in Δ may not. (As we shall see in Section 5, an arrow with an apply operator—which is equivalent to a monad—relinquishes precisely this restriction.)

Arrow unit promotes a term to a command.

$$\frac{\Gamma, \Delta \vdash M : A}{\Gamma; \Delta \vdash [M] \; ! \; A}$$

Note that in the hypothesis we have a term judgment with one environment (there is a comma between Γ and Δ), while in the conclusion we have a command judgment with two environments (there is a semicolon between Γ and Δ). This is the analogue of promotion of a function to an arrow in the classic formulation. It also resembles the unit of a monad.

Lastly, the value returned by a command may be bound.

$$\frac{\Gamma; \Delta \vdash P \; ! \; A \qquad \Gamma; \Delta, x : A \vdash Q \; ! \; B}{\Gamma; \Delta \vdash \mathsf{let}\ x = P\ \mathsf{in}\ Q \; ! \; B}$$

This resembles a traditional let term, save that the bound variable goes into the second environment, not the first. This is the analogue of arrow composition in the classic formulation, but (though this may not be immediately obvious) it also embodies the operation *first*. It also resembles the bind of a monad.

Arrow abstraction and application satisfy beta and eta laws, $(\beta^{\rightsquigarrow})$ and $(\eta^{\rightsquigarrow})$, while arrow unit and bind satisfy left unit, right unit, and associativity laws, (left), (right), and (assoc). Similar laws appear in the computational metalanguage of Moggi [1991]. The beta law equates the application of an abstraction to a bind; substitution is not part of beta, but instead appears in the left unit law.

(The laws on commands define an equational judgement between well-typed commands. Each law $P = Q$ is shorthand for $\Gamma; \Delta \vdash P = Q \; ! \; A$ for all Γ, Δ, A such that $\Gamma; \Delta \vdash M \; ! \; A$ and $\Gamma; \Delta \vdash N \; ! \; A$. The equational judgement is defined as the contextual and equivalence closure of the laws.)

Paterson's notation is closely related to ours. Here is a translation table, with our notation on the left and his on the right.

$\lambda^{\bullet} x.\, Q$	$\mathsf{proc}\ x \rightarrow Q$
$L \bullet M$	$L \ \prec\ M$
$[M]$	$\mathit{arrowA} \ \prec\ M$
$\mathsf{let}\ x = P\ \mathsf{in}\ Q$	$\mathsf{do}\ \{x \leftarrow P; Q\}$

In essence, each is a minor syntactic variant of the other. The only difference of note is that we introduce arrow unit as an explicit construct, $[M]$, while Paterson uses the equivalent form $\mathit{arrowA} \ \prec\ M$ where *arrowA* is *arr id*. Our introduction of a separate construct for arrow unit is slightly neater, and avoids the need to introduce *arrowA* as a constant in the arrow calculus.

4 Translations

We now consider translations between our two formulations, and show they are equivalent.

Translation

$$\begin{array}{rcl}
[\![x]\!] & = & x \\
[\![(M,N)]\!] & = & ([\![M]\!],[\![N]\!]) \\
[\![\mathsf{fst}\ L]\!] & = & \mathsf{fst}\ [\![L]\!] \\
[\![\mathsf{snd}\ L]\!] & = & \mathsf{snd}\ [\![L]\!] \\
[\![\lambda x.\, N]\!] & = & \lambda x.\, [\![N]\!] \\
[\![L\ M]\!] & = & [\![L]\!]\ [\![M]\!] \\
[\![\lambda^\bullet x.\, Q]\!] & = & [\![Q]\!]_x \\
[\![L \bullet M]\!]_\Delta & = & \mathit{arr}\ (\lambda\Delta.\, [\![M]\!]) \ggg [\![L]\!] \\
[\![[M]]\!]_\Delta & = & \mathit{arr}\ (\lambda\Delta.\, [\![M]\!]) \\
[\![\mathsf{let}\ x = P\ \mathsf{in}\ Q]\!]_\Delta & = & (\mathit{arr}\ \mathit{id}\ \&\&\&\ [\![P]\!]_\Delta) \ggg [\![Q]\!]_{\Delta,x}
\end{array}$$

Translation preserves types

$$\left[\!\!\left[\frac{\Gamma;\, x : A \vdash Q\ !\ B}{\Gamma \vdash \lambda^\bullet x.\, Q : A \rightsquigarrow B}\right]\!\!\right] \quad = \quad \frac{\Gamma \vdash [\![Q]\!]_x : A \rightsquigarrow B}{\Gamma \vdash [\![Q]\!]_x : A \rightsquigarrow B}$$

$$\left[\!\!\left[\frac{\Gamma \vdash L : A \rightsquigarrow B \qquad \Gamma, \Delta \vdash M : A}{\Gamma;\, \Delta \vdash L \bullet M\ !\ B}\right]\!\!\right] \quad = \quad \frac{\Gamma \vdash [\![L]\!] : A \rightsquigarrow B \qquad \Gamma, \Delta \vdash [\![M]\!] : A}{\Gamma \vdash \mathit{arr}\ (\lambda\Delta.\, [\![M]\!]) \ggg [\![L]\!] : \Delta \rightsquigarrow B}$$

$$\left[\!\!\left[\frac{\Gamma,\, \Delta \vdash M : A}{\Gamma;\, \Delta \vdash [M]\ !\ A}\right]\!\!\right] \quad = \quad \frac{\Gamma,\, \Delta \vdash [\![M]\!] : A}{\Gamma \vdash \mathit{arr}\ (\lambda\Delta.\, [\![M]\!]) : \Delta \rightsquigarrow A}$$

$$\left[\!\!\left[\frac{\Gamma;\, \Delta \vdash P\ !\ A \qquad \Gamma;\, \Delta,\, x : A \vdash Q\ !\ B}{\Gamma;\, \Delta \vdash \mathsf{let}\ x = P\ \mathsf{in}\ Q\ !\ B}\right]\!\!\right] \quad = \quad \frac{\Gamma \vdash [\![P]\!]_\Delta : \Delta \rightsquigarrow A \qquad \Gamma \vdash [\![Q]\!]_{\Delta,x} : \Delta \times A \rightsquigarrow B}{\Gamma \vdash (\mathit{arr}\ \mathit{id}\ \&\&\&\ [\![P]\!]_\Delta) \ggg [\![Q]\!]_{\Delta,x} : \Delta \rightsquigarrow B}$$

Figure 4: Translating Arrow Calculus into Classic Arrows

Translation

$$\begin{array}{rcl}
[\![x]\!]^{-1} & = & x \\
[\![(M,N)]\!]^{-1} & = & ([\![M]\!]^{-1},[\![N]\!]^{-1}) \\
[\![\mathsf{fst}\ L]\!]^{-1} & = & \mathsf{fst}\ [\![L]\!]^{-1} \\
[\![\mathsf{snd}\ L]\!]^{-1} & = & \mathsf{snd}\ [\![L]\!]^{-1} \\
[\![\lambda x.\, N]\!]^{-1} & = & \lambda x.\, [\![N]\!]^{-1} \\
[\![L\ M]\!]^{-1} & = & [\![L]\!]^{-1}\ [\![M]\!]^{-1} \\
[\![\mathit{arr}]\!]^{-1} & = & \lambda f.\, \lambda^\bullet x.\, [f\ x] \\
[\![(\ggg)]\!]^{-1} & = & \lambda f.\, \lambda g.\, \lambda^\bullet x.\, \mathsf{let}\ y = (f \bullet x)\ \mathsf{in}\ g \bullet y \\
[\![\mathit{first}]\!]^{-1} & = & \lambda f.\, \lambda^\bullet z.\, \mathsf{let}\ x = f \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle x, \mathsf{snd}\ z\rangle]
\end{array}$$

Figure 5: Translating Classic Arrows into Arrow Calculus

The translation from the arrow calculus into classic arrows is shown in Figure 4. An arrow calculus term judgment

$$\Gamma \vdash M : A$$

maps into a classic arrow judgment

$$\Gamma \vdash [\![M]\!] : A$$

while an arrow calculus command judgment

$$\Gamma; \Delta \vdash P \,!\, A$$

maps into a classic arrow judgment

$$\Gamma \vdash [\![P]\!]_\Delta : \Delta \rightsquigarrow A.$$

In $[\![P]\!]_\Delta$, we take Δ to stand for the sequence of variables in the environment, and in $\Delta \rightsquigarrow A$ we take Δ to stand for the product of the types in the environment. Hence, the denotation of a command is an arrow, with arguments corresponding to the environment Δ and result of type A.

The translation of the constructs of the core lambda calculus are straightforward homomorphisms. The translations of the remaining four constructs are shown twice, in the top half of the figure as equations on syntax, and in the bottom half in the context of type derivations; the latter are longer, but may be clearer to read. We comment briefly on each of the four:

- $\lambda^\bullet x.\, N$ translates straightforwardly; it is a no-op.
- $L \bullet M$ translates to $\ggg$.
- $[M]$ translates to *arr*.
- $\mathsf{let}\ x = P\ \mathsf{in}\ Q$ translates to pairing &&& (to extend the environment with P) and composition $\ggg$ (to then apply Q). The pairing operator &&& is defined using *first* (and *second*), as shown in Figure 2.

The translation uses the notation $\lambda\Delta.\, N$, which is given the obvious meaning: $\lambda x.\, N$ stands for itself, and $\lambda x_1, x_2.\, N$ stands for $\lambda z.\, N[x_1 := \mathsf{fst}\ z, x_2 := \mathsf{snd}\ z]$, and $\lambda x_1, x_2, x_3.\, N$ stands for $\lambda z.\, N[x_1 := \mathsf{fst}\ (\mathsf{fst}\ z), x_2 := \mathsf{snd}\ (\mathsf{fst}\ z), x_3 := \mathsf{snd}\ z]$, and so on.

The inverse translation, from classic arrows to the arrow calculus, is given in Figure 5. Again, the translation of the constructs of the core lambda calculus are straightforward homomorphisms. Each of the three constants translates to an appropriate term in the arrow calculus.

We can now show the following four properties.

- The five laws of the arrow calculus follow from the nine laws of classic arrows. That is,

$$M = N \text{ implies } [\![M]\!] = [\![N]\!]$$
$$\text{and}$$
$$P = Q \text{ implies } [\![P]\!]_\Delta = [\![Q]\!]_\Delta$$

  for all arrow calculus terms M, N and commands P, Q. The proof requires five calculations, one for each law of the arrow calculus.

- The nine laws of classic arrows follow from the five laws of the arrow calculus. That is,

$$M = N \text{ implies } [\![M]\!]^{-1} = [\![N]\!]^{-1}$$

  for all classic arrow terms M, N. The proof requires nine calculations, one for each classic arrow law.

- Translating a term from the arrow calculus into classic arrows and back again is the identity (up to equivalence). That is,
$$[\![\, [\![M]\!] \,]\!]^{-1} = M$$
for all arrow calculus terms M. Translating a command of the arrow calculus into classic arrows and back again is the identity (up to equivalence). That is,
$$[\![\, [\![P]\!]_\Delta \,]\!]^{-1} \bullet \Delta = P$$
for all arrow calculus commands P. The proof requires four calculations, one for each construct of the arrow calculus.

- Translating from classic arrows into the arrow calculus and back again is the identity (up to equivalence). That is,
$$[\![\, [\![M]\!]^{-1} \,]\!] = M$$
for all classic arrow terms M. The proof requires three calculations, one for each classic arrow constant.

These four properties together constitute an *equational correspondence* between classic arrows and the arrow calculus Sabry and Felleisen [1993]. The proofs that these properties hold are given in the Appendix.

5 Specialising arrows

Arrows generalise a whole range of different notions of computation. We can obtain specific kinds of arrows by adding extra syntax and extra laws to the arrow calculus. Here we describe how to capture *monads* and *idioms* in the arrow calculus. One might also capture other variants of arrows such as arrows with choice and arrows with fixed points.

5.1 Higher-order arrows

A *higher-order arrow* permits us to *apply* an arrow that is itself yielded by another arrow. As explained by Hughes [2000], an arrow with apply is equivalent to a monad. It is equipped with an additional constant

$$\mathit{app} \quad : \quad (A \rightsquigarrow B) \times A \rightsquigarrow B$$

which is an arrow analogue of function application. For the arrow calculus, equivalent structure is provided by a second version of arrow application, where the arrow to apply may itself be computed by an arrow.

$$\frac{\begin{array}{l}\Gamma, \Delta \vdash L : A \rightsquigarrow B \\ \Gamma, \Delta \vdash M : A\end{array}}{\Gamma, \Delta \vdash L \star M \; ! \; B}$$

This lifts the central restriction on arrow application. Now the arrow to apply may be the result of a command, and the command denoting the arrow may contain free variables in both Γ and Δ.

For classic arrows the additional laws for arrows with apply are:

$$\begin{array}{lrcl}
(\rightsquigarrow_{H1}) & \mathit{first}\ (\mathit{arr}\ (\lambda x.\, \mathit{arr}\ (\lambda y.\, \langle x, y \rangle))) \ggg \mathit{app} & = & \mathit{arr}\ \mathit{id} \\
(\rightsquigarrow_{H2}) & \mathit{first}\ (\mathit{arr}\ (g \ggg)) \ggg \mathit{app} & = & \mathit{second}\ g \ggg \mathit{app} \\
(\rightsquigarrow_{H3}) & \mathit{first}\ (\mathit{arr}\ (\ggg h)) \ggg \mathit{app} & = & \mathit{app} \ggg h
\end{array}$$

For the arrow calculus the additional laws are simply the beta and eta laws for $\star$:

$$\begin{array}{lrcl}
(\beta^{app}) & (\lambda^\bullet x.\, Q) \star M & = & Q[x := M] \\
(\eta^{app}) & \lambda^\bullet x.\, (L \star x) & = & L
\end{array}$$

There is a slight subtlety here. The β^{app} law does not necessarily preserve well-typing. Consider:

$$\lambda^\bullet f.\,(\lambda^\bullet x.\,f \bullet x) \star M = \lambda^\bullet f.\,f \bullet M$$

The left-hand-side can be assigned a type, but the right-hand-side cannot (f is a Δ-variable, but appears on the left of an arrow application). However, this is not a problem because the equational judgement on commands is only defined when both sides can be assigned types.

Remark The η^{app} laws is in fact redundant.

$$\begin{array}{ll} & \lambda^\bullet x.\,L \star x \\ = & \qquad (\eta^{\leadsto}) \\ & \lambda^\bullet x.\,(\lambda^\bullet y.\,L \bullet y) \star x \\ = & \qquad (\beta^{app}) \\ & \lambda^\bullet x.\,L \bullet x \\ = & \qquad (\eta^{\leadsto}) \\ & L \end{array}$$

These extensions to classic arrows and the arrow calculus maintain the equational correspondence. The translations are each extended with an extra clause.
Higher-order arrows to classic arrows with apply:

$$[\![L \star M]\!]_\Delta = \mathit{arr}\ (\lambda\Delta.\,[\![L]\!]\ \&\&\&\ [\![M]\!]) \ggg \mathit{app}$$

Classic arrows with apply to higher-order arrows:

$$[\![\mathit{app}]\!]^{-1} = \lambda^\bullet p.\,(\mathsf{fst}\ p) \star (\mathsf{snd}\ p)$$

5.2 Static arrows

A *static arrow* is a kind of arrow that is equivalent to an *idiom* (also known as *applicative functor*) [McBride and Paterson, 2008]. Static arrow computations are *oblivious* in the sense that they do not allow subsequent computations to depend on the result of prior computations.

In order to model static arrows in the framework of classic arrows we extend our base lambda calculus to include a unit type 1 along with the unit term () and add an additional constant

$$\mathit{delay} : (A \leadsto B) \to (1 \leadsto (A \to B))$$

which allows the input to a computation be delayed until after the side-effects have taken place.

For the arrow calculus, equivalent structure is provided by a restriction of arrow application which runs an arrow computation without supplying it with an input, producing a value of function type.

$$\frac{\Gamma \vdash L : A \leadsto B}{\Gamma;\,\Delta \vdash \mathsf{run}\ L\ !\ A \to B}$$

Note that the variables from the Δ environment cannot occur in L.

For classic arrows the additional laws for are:

$$\begin{array}{llcl} (\leadsto_{S1}) & \mathit{force}\ (\mathit{delay}\ a) & = & a \\ (\leadsto_{S2}) & \mathit{delay}\ (\mathit{force}\ a) & = & a \end{array}$$

where

$$\begin{array}{l} \mathit{force} : (1 \leadsto (A \to B)) \to (A \leadsto B) \\ \mathit{force} = \lambda f.\,\mathit{arr}\ (\lambda x.\,((), x)) \ggg \mathit{first}\ f \ggg \mathit{arr}\ (\lambda z.\,\mathsf{fst}\ z\ (\mathsf{snd}\ z)) \end{array}$$

For the arrow calculus the additional laws are:

$$\begin{array}{lrcl}
(ob_1) & L \bullet M & = & \mathsf{let}\ f = \mathsf{run}\ L\ \mathsf{in}\ [f\ M] \\
(ob_2) & \mathsf{run}\ (\lambda^{\bullet} x.\, [M]) & = & [\lambda x.\, M] \\
(ob_3) & \mathsf{run}\ (\lambda^{\bullet} x.\, \mathsf{let}\ y = P\ \mathsf{in}\ Q) & = & \mathsf{let}\ y = P\ \mathsf{in}\ \mathsf{let}\ f = \mathsf{run}\ (\lambda^{\bullet} \langle x, y \rangle.\, Q)\ \mathsf{in}\ [\lambda x.\, f\ \langle x, y \rangle]
\end{array}$$

The first law eliminates arrow applications by decomposing them into two parts. The first part runs the side-effects without looking at the input and returns a function value. The second part passes the input into the function obtained from the first part. The second law eliminates run commands. Running a pure arrow computation gives a pure function. The third law allows computation that is independent of the input to be lifted out of a run command. By repeated application of (ob_1) and (ob_2) in conjunction with the other laws we can reduce all run commands in a term to the canonical form $\mathsf{run}\ (f\ x_1\ \ldots\ x_n)$.

These extensions to classic arrows and the arrow calculus maintain the equational correspondence. The translations are each extended with an extra clause.

Static arrows to classic arrows with delay:

$$[\![\mathsf{run}\ L]\!]_{\Delta} = \mathit{arr}\ (\lambda \Delta.\, ()) \ggg \mathit{delay}\ [\![L]\!]$$

Classic arrows with delay to static arrows:

$$[\![\mathit{delay}]\!]^{-1} = \lambda x.\, \lambda^{\bullet} u.\, \mathsf{run}\ x$$

Robert Atkey suggested to us an alternative formulation of static arrows. His suggestion is to replace $\mathsf{run}\ L$ with a command level lambda abstraction.

$$\frac{\Gamma;\ \Delta, x : A \vdash P\ !\ B}{\Gamma;\ \Delta \vdash \lambda^{\star} x.\, P\ !\ A \to B}$$

with beta and eta laws

$$\begin{array}{lrcl}
(\beta^{ob}) & \mathsf{let}\ f = \lambda^{\star} x.\, P\ \mathsf{in}\ [f\ M] & = & P[x := M] \\
(\eta^{ob}) & \lambda^{\star} x.\, \mathsf{let}\ f = P\ \mathsf{in}\ [f\ x] & = & P
\end{array}$$

Run and command-level lambda abstraction are inter-definable.

$$\lambda^{\star} x.\, P = \mathsf{run}\ \ (\lambda^{\bullet} x.\, P)$$
$$\mathsf{run}\ L = \lambda^{\star} x.\, L \bullet x$$

An advantage of this version of static arrows is that it gives a nice duality between static arrows and higher-order arrows. For higher-order arrows we add a new kind of application, whereas for static arrows we add a new kind of abstraction. Further, his oblivious laws fit the standard pattern of beta and eta laws. A disadvantage is that (β^{ob}) and (η^{ob}) are not so convenient to use, and it is not clear how to obtain rewrite rules from them.

Remark Though we have defined languages for higher-order arrows and static arrows, it is not clear how useful they are for practical programming.

For programming arrows the syntactic sugar introduced by Ross Paterson, which amounts to the same thing as the arrow calculus, is often much more convenient than classic arrows as it does not force the programmer to write programs in a point-free style. However, if all you want is a monad then using something like Haskell `do` notation is considerably simpler than using higher-order arrows. Similarly, if all you want is an idiom, then using static arrows is overkill. For point-free programming the *pure* and *apply* combinators defined by McBride and Paterson [2008] work well. Our syntactic sugar for *formlets* [Cooper et al., 2008] illustrates a simpler approach than static arrows to programming static arrows without forcing a point-free style on the programmer (throwing away the XML from that syntax gives a general metalanguage that can be used for any idiom).

The reason we wanted to define static arrows and higher-order arrows was to bring idioms, arrows and monads into the same framework in order to make it easier to compare their relative expressive powers. One situation in which it may be desirable to use static arrows or higher-order arrows in practical programs is if you want to apply a generic arrow meta program to an idiom or a monad.

Beta rules

$$\begin{array}{lrcl}
(\beta^{\rightarrow}) & (\lambda x.\, N) M & \longrightarrow & N[x := M] \\
(\beta^{\leadsto}) & (\lambda^{\bullet} x.\, Q) \bullet M & \longrightarrow & Q[x := M] \\
(\beta^{!}) & \mathsf{let}\ x = [M]\ \mathsf{in}\ Q & \longrightarrow & Q[x := M]
\end{array}$$

Commuting conversions

$$(\text{assoc}) \quad \mathsf{let}\ y = (\mathsf{let}\ x = P\ \mathsf{in}\ Q)\ \mathsf{in}\ R \quad \longrightarrow \quad \mathsf{let}\ x = P\ \mathsf{in}\ (\mathsf{let}\ y = Q\ \mathsf{in}\ R)$$

Eta rules

$$\begin{array}{lrcl}
(\eta^{\rightarrow}) & E[S^{A \rightarrow B}] & \longrightarrow & E[\lambda x.\, S\ x] \\
(\eta^{\leadsto}) & E[S^{A \leadsto B}] & \longrightarrow & E[\lambda^{\bullet} x.\, S \bullet x] \\
(\eta^{!}) & F[T] & \longrightarrow & F[\mathsf{let}\ x = T\ \mathsf{in}\ [x]]
\end{array}$$

Neutral forms

$$\begin{array}{lcll}
\text{Terms} & S & ::= & x \mid L\ M \\
\text{Commands} & T & ::= & L \bullet M
\end{array}$$

Eta contexts

$$\begin{array}{lcll}
\text{Term contexts} & E[\,] & ::= & [\,] \mid \lambda x.\, [\,] \mid L\ [\,] \mid L \bullet [\,] \mid [[\,]] \\
\text{Command contexts} & F[\,] & ::= & [\,] \mid \lambda^{\bullet} x.\, [\,] \mid \mathsf{let}\ x = P\ \mathsf{in}\ [\,]
\end{array}$$

Normal forms

$$\begin{array}{lcll}
\text{Terms} & L, M, N & ::= & S^{X} \mid \lambda x.\, M \mid \lambda^{\bullet} x.\, P \\
\text{Commands} & P, Q, R & ::= & [M] \mid \mathsf{let}\ x = T\ \mathsf{in}\ P \\
 & & & \\
\text{Neutral terms} & S^{A} & ::= & x \mid S^{B \rightarrow A} M \\
\text{Neutral commands} & T^{A} & ::= & S^{B \leadsto A} \bullet M
\end{array}$$

Figure 6: Arrow calculus rewriting rules and normal forms

6 Rewriting and normal forms

It is relatively straightforward to obtain confluent and strongly normalising rewrite rules for the arrow calculus. Essentially we orient the β-laws and commuting conversion from left to right, and as advocated by Jay and Ghani [1995] orient the η-laws from right to left (as expansions). The eta rules are restricted in the usual way to operate only on neutral forms in non-elimination contexts. In order to handle the type-directed η-expansions and the subtleties of Δ contexts, we assume that writing takes place with respect to a typing environment (or two environments for commands), and terms are well-typed with respect to the typing environment. In other words when we write $M \longrightarrow N$ this really means: $\Gamma \vdash M \longrightarrow N : A$ whenever $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$, and similarly for $P \longrightarrow Q$.

Theorem 1. *The reduction rules are strongly normalising.*

Proof. The proof is by translation into the computational metalanguage, which is known to be strongly normalising [Lindley and Stark, 2005]. We define the translation $\mathcal{T}(\cdot)$ on types, terms and commands (the missing cases are homorphisms):

$$\mathcal{T}(A \leadsto B) = \mathcal{T}(A) \rightarrow \mathrm{T}\,(\mathcal{T}(B))$$

$$\mathcal{T}(\lambda^{\bullet} x.\, P) = \lambda x.\, \mathcal{T}(P)$$
$$\mathcal{T}(L \bullet M) = \mathcal{T}(L)\ \mathcal{T}(M)$$

(Note that both terms and commands map to terms in the computational metalanguage.) We now need to show that every reduction on an arrow calculus term M corresponds to one or more computational metalanguage reductions on $\mathcal{T}(M)$ (similarly for arrow calculus commands P). The only interesting cases are those involving arrow abstraction and application.

($\beta^{\rightsquigarrow}$) $(\lambda x.\, Q) \bullet M \;\longrightarrow\; Q[x := M]$:

$$\begin{array}{ll} & \mathcal{T}((\lambda^{\bullet} x.\, Q) \bullet M) \\ = & \\ & (\lambda x.\, \mathcal{T}(Q))\; \mathcal{T}(M) \\ \longrightarrow & \\ & \mathcal{T}(Q)[x := \mathcal{T}(M)] \\ = & \\ & \mathcal{T}(Q[x := M]) \end{array}$$

($\eta^{\rightsquigarrow}$) $E[S^{A \rightsquigarrow B}] \;\longrightarrow\; E[\lambda^{\bullet} x.\, (S \bullet x)]$:

$$\begin{array}{ll} & \mathcal{T}(E[S^{A \rightsquigarrow B}]) \\ = & \\ & \mathcal{T}(E)[\mathcal{T}(S^{A \rightsquigarrow B})] \\ = & \\ & \mathcal{T}(E)[\mathcal{T}(S)^{A \to \mathrm{T}\, B}] \\ \longrightarrow & \\ & \mathcal{T}(E)[\lambda x.\, \mathcal{T}(S)^{A \to \mathrm{T}\, B}\; x] \\ = & \\ & \mathcal{T}(E)[\lambda x.\, \mathcal{T}(S^{A \rightsquigarrow B} \bullet x)] \\ = & \\ & \mathcal{T}(E[\lambda^{\bullet} x.\, S^{A \rightsquigarrow B} \bullet x]) \end{array}$$

Thus every arrow calculus reduction maps to exactly one computational metalanguage reduction. $\square$

(Note that the proof of strong normalisation in [Lindley and Stark, 2005] does not include η-rules, but they are easy to add as illustrated in the more complicated setting of sums [Lindley, 2007].)

Theorem 2. *The reduction rules are confluent.*

Proof. It is straightforward to check weak confluence: that all the "critical pairs" of overlapping reductions are joinable. Confluence then follows from strong normalisation and Newman's lemma [Newman, 1942]. $\square$

6.1 Rewriting with higher-order arrows

In order to extend the rewriting theory to accomodate higher-order arrows, we orient (β^{app}) left-to-right and (η^{app}) right-to-left as reduction rules (Figure 7). We also need an extra rule for converting standard arrow applications to higher-order arrow applications.

$$(\textit{relax}) \quad L \bullet M \;=\; L \star M$$

The (*relax*) law is easily derived from the existing laws.

$$\begin{array}{ll} & L \bullet M \\ = & \qquad (\eta^{app}) \\ & (\lambda^{\bullet} x.\, L \star x) \bullet M \\ = & \qquad (\beta^{\rightsquigarrow}) \\ & L \star M \end{array}$$

Beta rules

$$\begin{array}{llll} \ldots & & & \\ (\beta^{app}) & (\lambda^{\bullet} x.\, Q) \star M & \longrightarrow & Q[x := M] \end{array}$$

Eta rules

$$\begin{array}{llll} \ldots & & & \\ (\eta^{app}) & E[S^{A \rightsquigarrow B}] & \longrightarrow & E[\lambda^{\bullet} x.\, S \star x] \end{array}$$

Neutral forms

$$\text{Commands} \quad T \quad ::= \quad \cdots \mid L \star M$$

Eta contexts

$$\text{Command contexts} \quad F[\;] \quad ::= \quad \cdots \mid L \star [\;]$$

Relaxation rule

$$(\mathit{relax}) \quad L \bullet M \quad \longrightarrow \quad L \star M$$

Normal forms

$$\begin{array}{llll} \text{Terms} & L, M, N & ::= & S^X \mid \lambda x.\, M \mid \lambda^{\bullet} x.\, P \\ \text{Commands} & P, Q, R & ::= & [M] \mid \mathsf{let}\ x = T\ \mathsf{in}\ P \\ & & & \\ \text{Neutral terms} & S^A & ::= & x \mid S^{B \to A} M \\ \text{Neutral commands} & T^A & ::= & S^{B \rightsquigarrow A} \star M \end{array}$$

Figure 7: Higher-order arrows rewriting rules and normal forms

Neutral forms

$$\text{Commands} \quad T \quad ::= \quad \cdots \mid \mathsf{run}\ L$$

The oblivious laws

$$\begin{array}{lrcl}
(ob_1) & L \bullet M & \longrightarrow & \mathsf{let}\ f = \mathsf{run}\ L\ \mathsf{in}\ [f\ M] \\
(ob_2) & \mathsf{run}\ (\lambda^\bullet x.\,[M]) & \longrightarrow & [\lambda x.\, M] \\
(ob_3) & \mathsf{run}\ (\lambda^\bullet x.\,\mathsf{let}\ y = P\ \mathsf{in}\ Q) & \longrightarrow & \mathsf{let}\ y = P\ \mathsf{in}\ \mathsf{let}\ f = \mathsf{run}\ (\lambda^\bullet \langle x, y\rangle.\, Q)\ \mathsf{in}\ [\lambda x.\, f\ \langle x, y\rangle]
\end{array}$$

Normal forms

$$\begin{array}{lll}
\text{Terms} & L, M, N & ::= \quad S^X \mid \lambda x.\, M \mid \lambda^\bullet x.\, P \\
\text{Commands} & P, Q, R & ::= \quad [M] \mid \mathsf{let}\ x = T\ \mathsf{in}\ P \\
 & & \\
\text{Neutral terms} & S^A & ::= \quad x \mid S^{B \to A} M \\
\text{Neutral commands} & T^A & ::= \quad \mathsf{run}\ S^A
\end{array}$$

Figure 8: Static arrows rewriting rules and normal forms

Strong normalisation and confluence follow from essentially the same argument as before. The only minor technicality is that we need to account for the relaxation rule. One way of doing so is to amend the translation to introduce a dummy β-redex on arguments. Then the relaxation rule can be simulated by contracting this redex.

$$\mathcal{T}(A \rightsquigarrow B) = \mathcal{T}(A) \to \mathrm{T}\,(\mathcal{T}(B))$$

$$\begin{aligned}
\mathcal{T}(\lambda^\bullet x.\, P) &= \lambda x.\, \mathcal{T}(P) \\
\mathcal{T}(L \bullet M) &= \mathcal{T}(L)\ (\lambda x.\, \mathcal{T}(M)\ x) \\
\mathcal{T}(L \star M) &= \mathcal{T}(L)\ \mathcal{T}(M)
\end{aligned}$$

6.2 Rewriting with static arrows

In order to extend the rewriting theory to accomodate static arrows, we orient the additional laws left-to-right as reduction rules (Figure 8).

Proving strong normalisation and confluence for static arrows is somewhat complicated by the oblivious laws. We believe that the rewriting theory is strongly normalising and confluent but do not have a proof.

6.3 Normalisation for equality checking

Based on the rewriting rules described above we have implemented an arrow calculus normaliser in OCaml along with a translation from classic arrows to arrow calculus. This provides a simple way of checking whether two classic arrow terms are equivalent: translate to arrow calculus, reduce to normal form and compare for syntactic equality.

7 Fruits

7.1 First surprise

A look at the proof of (right) reveals a mild surprise: ($\rightsquigarrow_2$), the right unit law of classic arrows, is not required to prove (right), the right unit law of the arrow calculus. Further, it turns out that ($\rightsquigarrow_2$) is also not required to prove the other four laws. But this is a big surprise! From the classic laws—excluding $(\rightsquigarrow_2)$—we can prove the laws of the arrow calculus, and from these we can in turn prove the classic laws—including $(\rightsquigarrow_2)$. It follows that $(\rightsquigarrow_2)$ must be redundant.

Once the arrow calculus provided the insight, it was not hard to find a direct proof of redundancy.

$$
\begin{array}{ll}
 & f \ggg \mathit{arr}\ \mathit{id} \\
= & \quad (\rightsquigarrow_1) \\
 & \mathit{arr}\ \mathit{id} \ggg f \ggg \mathit{arr}\ \mathit{id} \\
= & \quad (\mathit{fst} \cdot \mathit{dup} = \mathit{id}) \\
 & \mathit{arr}\ (\mathit{fst} \cdot \mathit{dup}) \ggg f \ggg \mathit{arr}\ \mathit{id} \\
= & \quad (\rightsquigarrow_4) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ \mathit{fst} \ggg f \ggg \mathit{arr}\ \mathit{id} \\
= & \quad (\rightsquigarrow_8) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ f \ggg \mathit{arr}\ \mathit{fst} \ggg \mathit{arr}\ \mathit{id} \\
= & \quad (\rightsquigarrow_4) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \cdot \mathit{fst}) \\
= & \quad (\mathit{id} \cdot \mathit{fst} = \mathit{fst}) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ f \ggg \mathit{arr}\ \mathit{fst} \\
= & \quad (\rightsquigarrow_8) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ \mathit{fst} \ggg f \\
= & \quad (\rightsquigarrow_4) \\
 & \mathit{arr}\ (\mathit{fst} \cdot \mathit{dup}) \ggg f \\
= & \quad (\mathit{fst} \cdot \mathit{dup} = \mathit{id}) \\
 & \mathit{arr}\ \mathit{id} \ggg f \\
= & \quad (\rightsquigarrow_1) \\
 & f
\end{array}
$$

We believe we are the first to observe that the nine classic arrow laws can be reduced to eight.

7.2 Second surprise

A look at the proof of (β^{app}) reveals another surprise: $(\rightsquigarrow_{H2})$, is not required to prove (β^{app}). From the classic laws—with apply but excluding $(\rightsquigarrow_{H2})$—we can prove the laws of higher-order arrows, and from these we can in turn prove the classic laws—including $(\rightsquigarrow_{H2})$. It follows that $(\rightsquigarrow_{H2})$ must be redundant.

We believe we are the first to observe that the three classic arrow laws for apply can be reduced to two.

7.3 Applications

The arrow calculus has already proven useful in practice. It enabled us to clarify the relationship between idioms, arrows and monads [Lindley et al., 2008]. Further, it provided the inspiration for the categorical semantics of arrows [Atkey, 2008].

Acknowledgements

We thank Robert Atkey and John Hughes for their comments.

References

Robert Atkey. What is a categorical model of arrows? In *Mathematical Structures in Functional Programming*, ENTCS, 2008.

Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop. An idiom's guide to formlets. Technical Report EDI-INF-RR-1263, School of Informatics, University of Edinburgh, 2008.

Antony Courtney and Conal Elliott. Genuinely functional user interfaces. In *Haskell Workshop*, pages 41–69, September 2001.

Paul Hudak, Antony Courtney, Henrik Nilsson, and John Peterson. Arrows, robots, and functional reactive programming. In Johan Jeuring and Simon Peyton Jones, editors, *Advanced Functional Programming, 4th International School*, volume 2638 of *LNCS*. Springer-Verlag, 2003.

John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37:67–111, May 2000.

Patrik Jansson and Johan Jeuring. Polytypic compact printing and parsing. In *European Symposium on Programming*, volume 1576 of *LNCS*, pages 273–287. Springer-Verlag, 1999.

C. Barry Jay and Neil Ghani. The virtues of eta-expansion. *J. Funct. Program.*, 5(2):135–154, 1995.

Sam Lindley. Extensional rewriting with sums. In *TLCA*, pages 255–271, 2007.

Sam Lindley and Ian Stark. Reducibility and tt-lifting for computation types. In *TLCA*, pages 262–277, 2005.

Sam Lindley, Philip Wadler, and Jeremy Yallop. Idioms are oblivious, arrows are meticulous, monads are promiscuous. In *Mathematical Structures in Functional Programming*, ENTCS, 2008.

Conor McBride and Ross Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1):1–13, 2008.

Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

Ross Paterson. A new notation for arrows. In *International Conference on Functional Programming*, pages 229–240. ACM Press, September 2001.

John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, October 1997.

John Power and Hayo Thielecke. Closed Freyd- and kappa-categories. In *ICALP*, volume 1644 of *LNCS*. Springer, 1999.

Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, 1993.

A Lemmas

Lemma 3. *[Weakening] Suppose that Δ_1, Δ_2 are well-formed environments and P is an arrow calculus command. Suppose further that (viewing environments as sequences) Δ_2 is a subsequence of Δ_1 and (viewing environments as sets) $fv(P) \subseteq \Delta_2$. Then*

$$[\![P]\!]_{\Delta_1} = arr\ (\lambda\Delta_1.\,\Delta_2) \ggg [\![P]\!]_{\Delta_2}$$

Proof. By induction on the structure of P. There is one case for each command form.

1. Case $P = L \bullet M$.

$$\begin{array}{ll}
 & [\![L \bullet M]\!]_{\Delta_1} \\
= & \qquad (\text{def } [\![-]\!]_{\Delta_1}) \\
 & \mathit{arr}\ (\lambda\Delta_1.\, [\![M]\!]) \ggg [\![L]\!] \\
= & \qquad ((\lambda\Delta_2.\, [\![M]\!]) \cdot (\lambda\Delta_1.\, \Delta_2) = \lambda\Delta_1.\, M) \\
 & \mathit{arr}\ ((\lambda\Delta_2.\, [\![M]\!]) \cdot (\lambda\Delta_1.\, \Delta_2)) \ggg [\![L]\!] \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ (\lambda\Delta_1.\, \Delta_2) \ggg \mathit{arr}\ (\lambda\Delta_2.\, [\![M]\!]) \ggg [\![L]\!] \\
= & \qquad (\text{def } [\![-]\!]_{\Delta_2}) \\
 & \mathit{arr}\ (\lambda\Delta_1.\, \Delta_2) \ggg [\![L \bullet M]\!]_{\Delta_2}
\end{array}$$

2. Case $P = [M]$.

$$\begin{array}{ll}
 & [\![[M]]\!]_{\Delta_1} \\
= & \qquad (\text{def } [\![-]\!]_{\Delta_1}) \\
 & \mathit{arr}\ (\lambda\Delta_1.\, [\![M]\!]) \\
= & \qquad ((\lambda\Delta_2.\, [\![M]\!]) \cdot (\lambda\Delta_1.\, \Delta_2) = \lambda\Delta_1.\, M) \\
 & \mathit{arr}\ ((\lambda\Delta_2.\, [\![M]\!]) \cdot (\lambda\Delta_1.\, \Delta_2)) \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ (\lambda\Delta_1.\, \Delta_2) \ggg \mathit{arr}\ (\lambda\Delta_2.\, [\![M]\!]) \\
= & \qquad (\text{def } [\![-]\!]_{\Delta_2}) \\
 & \mathit{arr}\ (\lambda\Delta_1.\, \Delta_2) \ggg [\![[M]]\!]_{\Delta_2}
\end{array}$$

3. Case $P = \mathsf{let}\ x = P\ \mathsf{in}\ Q$.

$$
\begin{array}{ll}
 & \llbracket \mathsf{let}\ x = P\ \mathsf{in}\ Q \rrbracket_{\Delta_1} \\
= & \qquad (\mathrm{def}\ \llbracket - \rrbracket_{\Delta_1}) \\
 & (arr\ id \mathbin{\&\&\&} \llbracket P \rrbracket_{\Delta_1}) \ggg \llbracket Q \rrbracket_{\Delta_1,x} \\
= & \qquad (\text{induction hypothesis}) \\
 & (arr\ id \mathbin{\&\&\&} (arr\ (\lambda\Delta_1.\,\Delta_2) \ggg \llbracket P \rrbracket_{\Delta_2})) \ggg arr\ ((\lambda\Delta_1.\,\Delta_2) \times id) \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\mathrm{def}\ \&\&\&) \\
 & arr\ dup \ggg \mathit{first}\ (arr\ id) \ggg arr\ swap \ggg \mathit{first}\ (arr\ (\lambda\Delta_1.\,\Delta_2) \ggg \llbracket P \rrbracket_{\Delta_2}) \ggg \\
 & \quad arr\ swap \ggg arr\ ((\lambda\Delta_1.\,\Delta_2) \times id) \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_6) \\
 & arr\ dup \ggg \mathit{first}\ (arr\ id) \ggg arr\ swap \ggg \mathit{first}\ (arr\ (\lambda\Delta_1.\,\Delta_2)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg \\
 & \quad arr\ swap \ggg arr\ ((\lambda\Delta_1.\,\Delta_2) \times id) \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_5) \\
 & arr\ dup \ggg arr\ (id \times id) \ggg arr\ swap \ggg arr\ ((\lambda\Delta_1.\,\Delta_2) \times id) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg \\
 & \quad arr\ swap \ggg arr\ ((\lambda\Delta_1.\,\Delta_2) \times id) \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_4) \\
 & arr\ (((\lambda\Delta_1.\,\Delta_2) \times id) \cdot swap \cdot (id \times id) \cdot dup) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg \\
 & \quad arr\ (((\lambda\Delta_1.\,\Delta_2) \times id) \cdot swap) \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (swap \cdot (id \times id) \cdot dup = dup) \\
 & arr\ (((\lambda\Delta_1.\,\Delta_2) \times id) \cdot dup) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ (((\lambda\Delta_1.\,\Delta_2) \times id) \cdot swap) \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (((\lambda\Delta_1.\,\Delta_2) \times id) \cdot dup = \lambda\Delta_1.\,(\Delta_2, \Delta_1)) \\
 & arr\ (\lambda\Delta_1.\,(\Delta_2, \Delta_1)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ (((\lambda\Delta_1.\,\Delta_2) \times id) \cdot swap) \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad ((f \times id) \cdot swap = swap \cdot (id \times f)) \\
 & arr\ (\lambda\Delta_1.\,(\Delta_2, \Delta_1)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ (swap.(id \times (\lambda\Delta_1.\,\Delta_2))) \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_4) \\
 & arr\ (\lambda\Delta_1.\,(\Delta_2, \Delta_1)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ (id \times (\lambda\Delta_1.\,\Delta_2)) \ggg arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_7) \\
 & arr\ (\lambda\Delta_1.\,(\Delta_2, \Delta_1)) \ggg arr\ (id \times (\lambda\Delta_1.\,\Delta_2)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_4) \\
 & arr\ ((id \times (\lambda\Delta_1.\,\Delta_2)) \cdot (\lambda\Delta_1.\,(\Delta_2, \Delta_1))) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad ((id \times (\lambda\Delta_1.\,\Delta_2)) \cdot (\lambda\Delta_1.\,(\Delta_2, \Delta_1)) = \lambda\Delta_1.\,(\Delta_2, \Delta_2)) \\
 & arr\ (\lambda\Delta_1.\,(\Delta_2, \Delta_2)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (swap \cdot (id \times id) \cdot dup \cdot (\lambda\Delta_1.\,\Delta_2) = \lambda\Delta_1.\,(\Delta_2, \Delta_2)) \\
 & arr\ (swap \cdot (id \times id) \cdot dup \cdot (\lambda\Delta_1.\,\Delta_2)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_4) \\
 & arr\ (\lambda\Delta_1.\,\Delta_2) \ggg arr\ dup \ggg arr\ (id \times id) \ggg arr\ swap \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg \\
 & \quad arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\leadsto_5) \\
 & arr\ (\lambda\Delta_1.\,\Delta_2) \ggg arr\ dup \ggg \mathit{first}\ (arr\ id) \ggg arr\ swap \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta_2} \ggg \\
 & \quad arr\ swap \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\mathrm{def}\ \&\&\&) \\
 & arr\ (\lambda\Delta_1.\,\Delta_2) \ggg (arr\ id \mathbin{\&\&\&} \llbracket P \rrbracket_{\Delta_2}) \ggg \llbracket Q \rrbracket_{\Delta_2,x} \\
= & \qquad (\mathrm{def}\ \llbracket - \rrbracket_{\Delta_2}) \\
 & arr\ (\lambda\Delta_1.\,\Delta_2) \ggg \llbracket \mathsf{let}\ x = P\ \mathsf{in}\ Q \rrbracket_{\Delta_2}
\end{array}
$$

□

Lemma 4. *[Interchange] If P is an arrow calculus command and Δ_1 and Δ_2 are environments such that $dom(\Delta_1) = dom(\Delta_2)$ and $\Delta_1(x) = \Delta_2(x)$ for all $x \in dom(\Delta_1)$ then*

$$\llbracket P \rrbracket_{\Delta_1} = arr\ (\lambda\Delta_1.\,\Delta_2) \ggg \llbracket P \rrbracket_{\Delta_2}$$

Proof. By induction on the structure of P. □

Lemma 5. *The translations of substitution on terms and commands is as follows.*

$$\begin{aligned} [\![P[x := N]]\!]_\Delta &= \mathit{arr}\ (\lambda\Delta.\,(\Delta, [\![N]\!])) \ggg [\![P]\!]_{\Delta,x} \\ [\![M[x := N]]\!] &= [\![M]\!][x := [\![N]\!]] \end{aligned}$$

Proof. By mutual induction on the derivations of P and M. There is one case for each command form and each term form. We give only the cases for command forms here.

1. Case $P = L \bullet M$.

$$\begin{array}{ll} & [\![(L \bullet M)[x := N]]\!]_\Delta \\ = & \qquad \text{(def substitution)} \\ & [\![L \bullet (M[x := N])]\!]_\Delta \\ = & \qquad \text{(def } [\![-]\!]_\Delta) \\ & \mathit{arr}\ (\lambda\Delta.\,[\![M[x := N]]\!]) \ggg [\![L]\!] \\ = & \qquad \text{(induction hypothesis)} \\ & \mathit{arr}\ (\lambda\Delta.\,[\![M]\!][x := [\![N]\!]]) \ggg [\![L]\!] \\ = & \qquad (\leadsto_4) \\ & \mathit{arr}\ (\lambda\Delta.\,(\Delta, [\![N]\!])) \ggg \mathit{arr}\ (\lambda\langle\Delta, x\rangle.\,[\![M]\!]) \ggg [\![L]\!] \\ = & \qquad \text{(def } [\![-]\!]_{\Delta,x}) \\ & \mathit{arr}\ (\lambda\Delta.\,(\Delta, [\![N]\!])) \ggg [\![L \bullet M]\!]_{\Delta,x} \end{array}$$

2. Case $P = [M]$.

$$\begin{array}{ll} & [\![[M][x := N]]\!]_\Delta \\ = & \qquad \text{(def substitution)} \\ & [\![[M[x := N]]]\!]_\Delta \\ = & \qquad \text{(def } [\![-]\!]_\Delta) \\ & \mathit{arr}\ (\lambda\Delta.\,[\![M[x := N]]\!]) \\ = & \qquad \text{(induction hypothesis)} \\ & \mathit{arr}\ (\lambda\Delta.\,[\![M]\!][x := [\![N]\!]]) \\ = & \qquad (\leadsto_4) \\ & \mathit{arr}\ (\lambda\Delta.\,\langle\Delta, [\![N]\!]\rangle) \ggg \mathit{arr}\ (\lambda\langle\Delta, x\rangle.\,[\![M]\!]) \\ = & \qquad \text{(def } [\![-]\!]_{\Delta,x}) \\ & \mathit{arr}\ (\lambda\Delta.\,\langle\Delta, [\![N]\!]\rangle) \ggg [\![[M]]\!]_{\Delta,x} \end{array}$$

3. Case $P = \mathsf{let}\ y = P\ \mathsf{in}\ Q$.

$$
\begin{aligned}
& \llbracket (\mathsf{let}\ y = P\ \mathsf{in}\ Q)[x := N] \rrbracket_{\Delta} \\
= & \qquad \text{(def substitution)} \\
& \llbracket \mathsf{let}\ y = P[x := N]\ \mathsf{in}\ Q[x := N] \rrbracket_{\Delta} \\
= & \qquad \text{(def } \llbracket - \rrbracket_{\Delta}\text{)} \\
& (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \llbracket P[x := N] \rrbracket_{\Delta}) \ggg \llbracket Q[x := N] \rrbracket_{\Delta,y} \\
= & \qquad \text{(induction hypothesis)} \\
& (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} (\mathit{arr}\ (\lambda\Delta.\, \langle \Delta, \llbracket N \rrbracket \rangle) \ggg \llbracket P \rrbracket_{\Delta,x})) \ggg \mathit{arr}\ (\lambda\langle \Delta, y\rangle.\, \langle \Delta, y, \llbracket N \rrbracket \rangle) \ggg \llbracket Q \rrbracket_{\Delta,y,x} \\
= & \qquad \text{(def } \mathbin{\&\&\&}\text{)} \\
& \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{id}) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ (\mathit{arr}\ (\lambda\Delta.\, \langle \Delta, \llbracket N \rrbracket \rangle) \ggg \llbracket P \rrbracket_{\Delta,x}) \ggg \\
& \quad \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ (\lambda\langle \Delta, y\rangle.\, \langle \Delta, y, \llbracket N \rrbracket \rangle) \ggg \llbracket Q \rrbracket_{\Delta,y,x} \\
= & \qquad (\leadsto_6) \\
& \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{id}) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ (\mathit{arr}\ (\lambda\Delta.\, \langle \Delta, \llbracket N \rrbracket \rangle)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ (\lambda\langle \Delta, y\rangle.\, \langle \Delta, y, \llbracket N \rrbracket \rangle) \ggg \llbracket Q \rrbracket_{\Delta,y,x} \\
= & \qquad (\leadsto_5) \\
& \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{id} \times \mathit{id}) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ ((\lambda\Delta.\, \langle \Delta, \llbracket N \rrbracket \rangle) \times \mathit{id}) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ (\lambda\langle \Delta, y\rangle.\, \langle \Delta, y, \llbracket N \rrbracket \rangle) \ggg \llbracket Q \rrbracket_{\Delta,y,x} \\
= & \qquad (\leadsto_4) \\
& \mathit{arr}\ (\lambda\Delta.\, (\langle \Delta, \llbracket N \rrbracket \rangle, \Delta)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ (\lambda\langle y, \Delta\rangle.\, \langle \Delta, y, \llbracket N \rrbracket \rangle) \ggg \llbracket Q \rrbracket_{\Delta,y,x} \\
= & \qquad \text{(lemma 4)} \\
& \mathit{arr}\ (\lambda\Delta.\, (\langle \Delta, \llbracket N \rrbracket \rangle, \Delta)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ (\lambda\langle y, \Delta\rangle.\, \langle \Delta, y, \llbracket N \rrbracket \rangle) \ggg \mathit{arr}\ (\lambda\langle \Delta, y, x\rangle.\, \langle \Delta, x, y \rangle) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad (\leadsto_4) \\
& \mathit{arr}\ (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle) \ggg \mathit{arr}\ (\mathit{id} \times \mathit{fst}) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ (\lambda\langle y, \Delta\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad (\leadsto_7) \\
& \mathit{arr}\ (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ (\mathit{id} \times \mathit{fst}) \ggg \mathit{arr}\ (\lambda\langle y, \Delta\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad (\leadsto_4) \\
& \mathit{arr}\ (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ (\mathit{swap} \cdot (\mathit{id} \times (\mathit{id} \times (\lambda x.\, \llbracket N \rrbracket)))) \ggg \mathit{arr}\ ((\lambda\langle y, \langle \Delta, x\rangle\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle)) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad (\leadsto_4) \\
& \mathit{arr}\ (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ (\mathit{id} \times (\mathit{id} \times (\lambda x.\, \llbracket N \rrbracket))) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ ((\lambda\langle y, \langle \Delta, x\rangle\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle)) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad (\leadsto_7) \\
& \mathit{arr}\ (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle) \ggg \mathit{arr}\ (\mathit{id} \times (\mathit{id} \times (\lambda x.\, \llbracket N \rrbracket))) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ ((\lambda\langle y, \langle \Delta, x\rangle\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle)) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad (\leadsto_4) \\
& \mathit{arr}\ ((\mathit{id} \times (\mathit{id} \times (\lambda x.\, \llbracket N \rrbracket))) \cdot (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle)) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ ((\lambda\langle y, \langle \Delta, x\rangle\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle)) \ggg \llbracket Q \rrbracket_{\Delta,x,y} \\
= & \qquad ((\mathit{id} \times (\mathit{id} \times (\lambda x.\, \llbracket N \rrbracket))) \cdot (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle)) = (\lambda\Delta.\, (\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle))) \\
& \mathit{arr}\ (\lambda\Delta.\, \langle\langle \Delta, \llbracket N \rrbracket \rangle, \langle \Delta, \llbracket N \rrbracket \rangle\rangle) \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta,x} \ggg \\
& \quad \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ ((\lambda\langle y, \langle \Delta, x\rangle\rangle.\, \langle \Delta, \llbracket N \rrbracket, y \rangle)) \ggg \llbracket Q \rrbracket_{\Delta,x,y}
\end{aligned}
$$

$=$ $\quad (\leadsto_4)$
$\mathit{arr}\;(\lambda\Delta.\,\langle\Delta, \llbracket N\rrbracket\rangle) \ggg \mathit{arr}\;\mathit{dup} \ggg \mathit{arr}\;(\mathit{id} \times \mathit{id}) \ggg \mathit{arr}\;\mathit{swap} \ggg \mathit{first}\;\llbracket P\rrbracket_{\Delta,x} \ggg$
$\quad \mathit{arr}\;\mathit{swap} \ggg \llbracket Q\rrbracket_{\Delta,x,y}$
$=$ $\quad (\leadsto_5)$
$\mathit{arr}\;(\lambda\Delta.\,\langle\Delta, \llbracket N\rrbracket\rangle) \ggg \mathit{arr}\;\mathit{dup} \ggg \mathit{first}\;(\mathit{arr}\;\mathit{id}) \ggg \mathit{arr}\;\mathit{swap} \ggg \mathit{first}\;\llbracket P\rrbracket_{\Delta,x} \ggg$
$\quad \mathit{arr}\;\mathit{swap} \ggg \llbracket Q\rrbracket_{\Delta,x,y}$
$=$ $\quad (\text{def } \&\&\&)$
$\mathit{arr}\;(\lambda\Delta.\,\langle\Delta, \llbracket N\rrbracket\rangle) \ggg (\mathit{arr}\;\mathit{id} \mathbin{\&\&\&} \llbracket P\rrbracket_{\Delta,x}) \ggg \llbracket Q\rrbracket_{\Delta,x,y}$
$=$ $\quad (\text{def } \llbracket - \rrbracket_{\Delta,x})$
$\mathit{arr}\;(\lambda\Delta.\,\langle\Delta, \llbracket N\rrbracket\rangle) \ggg \llbracket \mathsf{let}\; y = P \;\mathsf{in}\; Q\rrbracket_{\Delta,x}$

$\square$

B Arrow calculus proofs

B.1 The arrow laws follow from the laws of the metalanguage

For each law $M = N$ we must show $\llbracket M\rrbracket^{-1} = \llbracket N\rrbracket^{-1}$.

1. $\llbracket \mathit{arr}\;\mathit{id} \ggg f\rrbracket^{-1} = \llbracket f\rrbracket^{-1}$

$\llbracket \mathit{arr}\;\mathit{id} \ggg f\rrbracket^{-1}$
$=$ $\quad (\text{def } \llbracket - \rrbracket^{-1})$
$\lambda^\bullet x.\,(\mathsf{let}\; y = (\lambda^\bullet z.\,[\mathit{id}\;z]) \bullet x \;\mathsf{in}\; \llbracket f\rrbracket^{-1} \bullet y)$
$=$ $\quad (\beta^{\to})$
$\lambda^\bullet x.\,(\mathsf{let}\; y = (\lambda^\bullet z.\,[z]) \bullet x \;\mathsf{in}\; \llbracket f\rrbracket^{-1} \bullet y)$
$=$ $\quad (\beta^{\leadsto})$
$\lambda^\bullet x.\,(\mathsf{let}\; y = [x] \;\mathsf{in}\; \llbracket f\rrbracket^{-1} \bullet y)$
$=$ $\quad (\text{left})$
$\lambda^\bullet x.\,(\llbracket f\rrbracket^{-1} \bullet x)$
$=$ $\quad (\eta^{\leadsto})$
$\llbracket f\rrbracket^{-1}$

2. $\llbracket f \ggg \mathit{arr}\;\mathit{id}\rrbracket^{-1} = \llbracket f\rrbracket^{-1}$

$\llbracket f \ggg \mathit{arr}\;\mathit{id}\rrbracket^{-1}$
$=$ $\quad (\text{def } \llbracket - \rrbracket^{-1})$
$\lambda^\bullet x.\,(\mathsf{let}\; y = \llbracket f\rrbracket^{-1} \bullet x \;\mathsf{in}\; (\lambda^\bullet z.\,[\mathit{id}\;z]) \bullet y)$
$=$ $\quad (\beta^{\to})$
$\lambda^\bullet x.\,(\mathsf{let}\; y = \llbracket f\rrbracket^{-1} \bullet x \;\mathsf{in}\; (\lambda^\bullet z.\,[z]) \bullet y)$
$=$ $\quad (\beta^{\leadsto})$
$\lambda^\bullet x.\,(\mathsf{let}\; y = \llbracket f\rrbracket^{-1} \bullet x \;\mathsf{in}\; [y])$
$=$ $\quad ((\text{right}))$
$\lambda^\bullet x.\,(\llbracket f\rrbracket^{-1} \bullet x)$
$=$ $\quad (\eta\leadsto)$
$\llbracket f\rrbracket^{-1}$

3. $\llbracket (f \ggg g) \ggg h \rrbracket^{-1} = \llbracket f \ggg (g \ggg h) \rrbracket^{-1}$

$$\begin{array}{ll}
 & \llbracket (f \ggg g) \ggg h \rrbracket^{-1} \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ w = (\lambda^{\bullet} x.\, (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet x\ \mathsf{in}\ \llbracket g \rrbracket^{-1} \bullet y)) \bullet z\ \mathsf{in}\ \llbracket h \rrbracket^{-1} \bullet w) \\
= & \qquad (\beta^{\leadsto}) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ w = (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet z\ \mathsf{in}\ \llbracket g \rrbracket^{-1} \bullet y)\ \mathsf{in}\ \llbracket h \rrbracket^{-1} \bullet w) \\
= & \qquad (\text{assoc}) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet z\ \mathsf{in}\ (\mathsf{let}\ y = \llbracket g \rrbracket^{-1} \bullet y)\ \mathsf{in}\ \llbracket h \rrbracket^{-1} \bullet w) \\
= & \qquad (\beta^{\leadsto}) \\
 & \lambda^{\bullet} x.\, (\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet x\ \mathsf{in}\ (\lambda^{\bullet} z.\, (\mathsf{let}\ y = \llbracket g \rrbracket^{-1} \bullet z\ \mathsf{in}\ \llbracket h \rrbracket^{-1} \bullet y)) \bullet w) \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \llbracket f \ggg (g \ggg h) \rrbracket^{-1}
\end{array}$$

4. $\llbracket arr\ (g \cdot f) \rrbracket^{-1} = \llbracket arr\ f \ggg arr\ g \rrbracket^{-1}$

$$\begin{array}{ll}
 & \llbracket arr\ (g \cdot f) \rrbracket^{-1} \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \lambda^{\bullet} x.\, [(\llbracket g \rrbracket^{-1} \cdot \llbracket f \rrbracket^{-1})\ x] \\
= & \qquad (\text{def } \cdot) \\
 & \lambda^{\bullet} x.\, ([\llbracket g \rrbracket^{-1}\ (\llbracket f \rrbracket^{-1}\ x)]) \\
= & \qquad (\text{left}) \\
 & \lambda^{\bullet} x.\, (\mathsf{let}\ y = [\llbracket f \rrbracket^{-1}\ x]\ \mathsf{in}\ [\llbracket g \rrbracket^{-1}\ y]) \\
= & \qquad (\beta^{\leadsto}(\times 2)) \\
 & \lambda^{\bullet} x.\, (\mathsf{let}\ y = (\lambda^{\bullet} x.\, [\llbracket f \rrbracket^{-1}\ x]) \bullet x\ \mathsf{in}\ (\lambda^{\bullet} x.\, [\llbracket g \rrbracket^{-1}\ x]) \bullet y) \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \llbracket arr\ f \ggg arr\ g \rrbracket^{-1}
\end{array}$$

5. $\llbracket \mathit{first}\ (f \ggg g) \rrbracket^{-1} = \llbracket \mathit{first}\ f \ggg \mathit{first}\ g \rrbracket^{-1}$

$$\begin{array}{ll}
 & \llbracket \mathit{first}\ (f \ggg g) \rrbracket^{-1} \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \lambda^{\bullet} z.\, \mathsf{let}\ w = (\lambda^{\bullet} x.\, (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet x\ \mathsf{in}\ \llbracket g \rrbracket^{-1} \bullet y)) \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle w, \mathsf{snd}\ z \rangle] \\
= & \qquad (\beta^{\leadsto}) \\
 & \lambda^{\bullet} z.\, \mathsf{let}\ w = (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ \llbracket g \rrbracket^{-1} \bullet y)\ \mathsf{in}\ [\langle w, \mathsf{snd}\ z \rangle] \\
= & \qquad (\text{assoc}) \\
 & \lambda^{\bullet} z.\, \mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ \mathsf{let}\ w = \llbracket g \rrbracket^{-1} \bullet y\ \mathsf{in}\ [\langle w, \mathsf{snd}\ z \rangle] \\
= & \qquad (\beta_1^{\times}, \beta_2^{\times}) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ \mathsf{let}\ w = \llbracket g \rrbracket^{-1} \bullet \mathsf{fst}\ \langle y, \mathsf{snd}\ z \rangle\ \mathsf{in}\ [\langle w, \mathsf{snd}\ \langle y, \mathsf{snd}\ z \rangle \rangle]) \\
= & \qquad (\text{left}) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ \mathsf{let}\ v = [\langle y, \mathsf{snd}\ z \rangle]\ \mathsf{in}\ \mathsf{let}\ w = \llbracket g \rrbracket^{-1} \bullet \mathsf{fst}\ v\ \mathsf{in}\ [\langle w, \mathsf{snd}\ v \rangle]) \\
= & \qquad (\text{assoc}) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ v = (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle y, \mathsf{snd}\ z \rangle])\ \mathsf{in}\ \mathsf{let}\ w = \llbracket g \rrbracket^{-1} \bullet \mathsf{fst}\ v\ \mathsf{in}\ [\langle w, \mathsf{snd}\ v \rangle]) \\
= & \qquad (\beta^{\leadsto}(\times 2)) \\
 & \lambda^{\bullet} z.\, (\mathsf{let}\ v = (\lambda^{\bullet} x.\, (\mathsf{let}\ y = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\langle y, \mathsf{snd}\ x \rangle])) \bullet z\ \mathsf{in} \\
 & \qquad (\lambda^{\bullet} x.\, (\mathsf{let}\ w = \llbracket g \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\langle w, \mathsf{snd}\ x \rangle])) \bullet v) \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \llbracket \mathit{first}\ f \ggg \mathit{first}\ g \rrbracket^{-1}
\end{array}$$

6. $[\![\mathit{first}\ (\mathit{arr}\ f)]\!]^{-1} = [\![\mathit{arr}\ (f \times \mathit{id})]\!]^{-1}$

$$
\begin{array}{ll}
 & [\![\mathit{first}\ (\mathit{arr}\ f)]\!]^{-1} \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^{\bullet} z.\,(\mathsf{let}\ x = (\lambda^{\bullet} w.\,([[\![f]\!]^{-1}\ w])) \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle x, \mathsf{snd}\ z\rangle]) \\
= & \qquad (\beta^{\leadsto}) \\
 & \lambda^{\bullet} z.\,(\mathsf{let}\ x = [[\![f]\!]^{-1}\ \mathsf{fst}\ z]\ \mathsf{in}\ [\langle x, \mathsf{snd}\ z\rangle]) \\
= & \qquad (\text{left}) \\
 & \lambda^{\bullet} z.\,[\langle [\![f]\!]^{-1}\ \mathsf{fst}\ z, \mathsf{snd}\ z\rangle] \\
= & \qquad (\text{def } \times) \\
 & \lambda^{\bullet} z.\,[([\![f]\!]^{-1} \times \mathit{id})\ z] \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & [\![\mathit{arr}\ (f \times \mathit{id})]\!]^{-1}
\end{array}
$$

7. $[\![\mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \times g)]\!]^{-1} = [\![\mathit{arr}\ (\mathit{id} \times g) \ggg \mathit{first}\ f]\!]^{-1}$

$$
\begin{array}{ll}
 & [\![\mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \times g)]\!]^{-1} \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ y = (\lambda^{\bullet} z.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle v, \mathsf{snd}\ z\rangle])) \bullet x\ \mathsf{in}\ (\lambda^{\bullet} w.\,[(\mathit{id} \times [\![g]\!]^{-1})\ w]) \bullet y) \\
= & \qquad (\beta^{\leadsto}(\times 2)) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ y = (\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\langle v, \mathsf{snd}\ x\rangle])\ \mathsf{in}\ [(\mathit{id} \times [\![g]\!]^{-1})\ y]) \\
= & \qquad (\text{assoc}) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ \mathsf{let}\ y = [\langle v, \mathsf{snd}\ x\rangle]\ \mathsf{in}\ [(\mathit{id} \times [\![g]\!]^{-1})\ y]) \\
= & \qquad (\text{left}) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [(\mathit{id} \times [\![g]\!]^{-1})\ \langle v, \mathsf{snd}\ x\rangle]) \\
= & \qquad (\text{def } \times) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\langle v, [\![g]\!]^{-1} \mathsf{snd}\ x\rangle]) \\
= & \qquad (\beta_1^{\times}, \beta_2^{\times}) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ \langle \mathsf{fst}\ x, [\![g]\!]^{-1}\ x\rangle\ \mathsf{in}\ [\langle v, \mathsf{snd}\ \langle \mathsf{fst}\ x, [\![g]\!]^{-1}\ \mathsf{snd}\ x\rangle\rangle]) \\
= & \qquad (\text{def } \times) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ (\mathit{id} \times [\![g]\!]^{-1})\ x\ \mathsf{in}\ [\langle v, \mathsf{snd}\ ((\mathit{id} \times [\![g]\!]^{-1})\ x)\rangle]) \\
= & \qquad (\text{left}) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ y = [(\mathit{id} \times [\![g]\!]^{-1})\ x]\ \mathsf{in}\ \mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ y\ \mathsf{in}\ [\langle v, \mathsf{snd}\ y\rangle]) \\
= & \qquad (\beta^{\leadsto}(\times 2)) \\
 & \lambda^{\bullet} x.\,(\mathsf{let}\ y = (\lambda^{\bullet} w.\,[(\mathit{id} \times [\![g]\!]^{-1})\ w]) \bullet x\ \mathsf{in}\ (\lambda^{\bullet} z.\,(\mathsf{let}\ v = [\![f]\!]^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle v, \mathsf{snd}\ z\rangle])) \bullet y) \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & [\![\mathit{arr}\ (\mathit{id} \times g) \ggg \mathit{first}\ f]\!]^{-1}
\end{array}
$$

8. $[\![\mathit{first}\ (\mathit{first}\ f) \ggg \mathit{arr}\ \mathit{assoc}]\!]^{-1} = [\![\mathit{arr}\ \mathit{assoc} \ggg \mathit{first}\ f]\!]^{-1}$

$\llbracket \mathit{first}\ (\mathit{first}\ f) \ggg \mathit{arr}\ \mathit{assoc} \rrbracket^{-1}$

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\lambda^\bullet a.\,(\mathsf{let}\ b = (\lambda^\bullet z.\,(\mathsf{let}\ x = (\lambda^\bullet q.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ q\ \mathsf{in}\ [\langle r, \mathsf{snd}\ q\rangle])) \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle x, \mathsf{snd}\ z\rangle])) \bullet a\ \mathsf{in}$
$\quad (\lambda^\bullet w.\,[\mathit{assoc}\ w]) \bullet b)$

$=$ ($\beta^{\leadsto}(\times 3)$)

$\lambda^\bullet a.\,(\mathsf{let}\ b = (\mathsf{let}\ x = (\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ (\mathsf{fst}\ a)\ \mathsf{in}\ [\langle r, \mathsf{snd}\ (\mathsf{fst}\ a)\rangle])\ \mathsf{in}\ [\langle x, \mathsf{snd}\ a\rangle])\ \mathsf{in}\ [\mathit{assoc}\ b])$

$=$ (assoc($\times 3$))

$\lambda^\bullet a.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ (\mathsf{fst}\ a)\ \mathsf{in}\ \mathsf{let}\ x = [\langle r, \mathsf{snd}\ (\mathsf{fst}\ a)\rangle]\ \mathsf{in}\ \mathsf{let}\ b = [\langle x, \mathsf{snd}\ a\rangle]\ \mathsf{in}\ [\mathit{assoc}\ b])$

$=$ (left($\times 2$))

$\lambda^\bullet a.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ (\mathsf{fst}\ a)\ \mathsf{in}\ [\mathit{assoc}\ \langle\langle r, \mathsf{snd}\ (\mathsf{fst}\ a)\rangle, \mathsf{snd}\ a\rangle])$

$=$ (def assoc)

$\lambda^\bullet a.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ (\mathsf{fst}\ a)\ \mathsf{in}$
$\quad [\langle \mathsf{fst}\ \mathsf{fst}\ (\langle r, \mathsf{snd}\ (\mathsf{fst}\ a)\rangle, \mathsf{snd}\ a\rangle, \langle \mathsf{snd}\ (\mathsf{fst}\ \langle\langle r, \mathsf{snd}\ (\mathsf{fst}\ a)\rangle, \mathsf{snd}\ a\rangle), \mathsf{snd}\ \langle\langle r, \mathsf{snd}\ (\mathsf{fst}\ a)\rangle, \mathsf{snd}\ a\rangle\rangle\rangle])$

$=$ ($\beta_1^\times(\times 3), \beta_2^\times(\times 2)$)

$\lambda^\bullet a.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ (\mathsf{fst}\ a)\ \mathsf{in}\ [\langle r, \langle \mathsf{snd}\ (\mathsf{fst}\ a), \mathsf{snd}\ a\rangle\rangle])$

$=$ ($\beta_1^\times, \beta_2^\times$)

$\lambda^\bullet a.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ \langle \mathsf{fst}\ (\mathsf{fst}\ a), \langle \mathsf{snd}\ (\mathsf{fst}\ a), \mathsf{snd}\ a\rangle\rangle\ \mathsf{in}\ [\langle r, \mathsf{snd}\ \langle \mathsf{fst}\ (\mathsf{fst}\ a), \langle \mathsf{snd}\ (\mathsf{fst}\ a), \mathsf{snd}\ a\rangle\rangle\rangle])$

$=$ (def assoc)

$\lambda^\bullet a.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ \mathit{assoc}\ a\ \mathsf{in}\ [\langle r, \mathsf{snd}\ \mathit{assoc}\ a\rangle])$

$=$ (left)

$\lambda^\bullet a.\,(\mathsf{let}\ b = [\mathit{assoc}\ a]\ \mathsf{in}\ \mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ b\ \mathsf{in}\ [\langle r, \mathsf{snd}\ b\rangle])$

$=$ ($\beta^{\leadsto}(\times 2)$)

$\lambda^\bullet a.\,(\mathsf{let}\ b = (\lambda^\bullet r.\,[\mathit{assoc}\ r]) \bullet a\ \mathsf{in}\ (\lambda^\bullet z.\,(\mathsf{let}\ r = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle r, \mathsf{snd}\ z\rangle])) \bullet b)$

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\llbracket \mathit{arr}\ \mathit{assoc} \ggg \mathit{first}\ f \rrbracket^{-1}$

9. $\llbracket \mathit{first}\ f \ggg \mathit{arr}\ \mathit{fst} \rrbracket^{-1} = \llbracket \mathit{arr}\ \mathit{fst} \ggg f \rrbracket^{-1}$

$\llbracket \mathit{first}\ f \ggg \mathit{arr}\ \mathit{fst} \rrbracket^{-1}$

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\lambda^\bullet x.\,(\mathsf{let}\ y = (\lambda^\bullet z.\,(\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle w, \mathsf{snd}\ z\rangle])) \bullet x\ \mathsf{in}\ (\lambda^\bullet z.\,[\mathsf{fst}\ z]) \bullet y)$

$=$ ($\beta^{\leadsto}(\times 2)$)

$\lambda^\bullet x.\,(\mathsf{let}\ y = (\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\langle w, \mathsf{snd}\ x\rangle])\ \mathsf{in}\ [\mathsf{fst}\ y])$

$=$ (assoc)

$\lambda^\bullet x.\,(\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ \mathsf{let}\ y = [\langle w, \mathsf{snd}\ x\rangle]\ \mathsf{in}\ [\mathsf{fst}\ y])$

$=$ (left)

$\lambda^\bullet x.\,(\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\mathsf{fst}\ \langle w, \mathsf{snd}\ x\rangle])$

$=$ ($\beta_1^\times$)

$\lambda^\bullet x.\,(\mathsf{let}\ w = \llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [w])$

$=$ (right)

$\lambda^\bullet x.\,(\llbracket f \rrbracket^{-1} \bullet \mathsf{fst}\ x)$

$=$ (left)

$\lambda^\bullet x.\,(\mathsf{let}\ y = [\mathsf{fst}\ x]\ \mathsf{in}\ \llbracket f \rrbracket^{-1} \bullet y)$

$=$ ($\beta^{\leadsto}$)

$\lambda^\bullet x.\,(\mathsf{let}\ y = (\lambda^\bullet z.\,[\mathsf{fst}\ z]) \bullet x\ \mathsf{in}\ \llbracket f \rrbracket^{-1} \bullet y)$

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\llbracket \mathit{arr}\ \mathit{fst} \ggg f \rrbracket^{-1}$

B.2 The laws of the metalanguage follow from the arrow laws

For each law $M = N$ we must show $[\![M]\!] = [\![N]\!]$.

We make implicit use of $\rightsquigarrow_3$, writing $L \ggg M \ggg N$ for both $(L \ggg M) \ggg N$ and $L \ggg (M \ggg N)$.

1. $[\![\mathsf{let}\ x = [M]\ \mathsf{in}\ Q]\!]_\Delta = [\![Q[x := M]]\!]_\Delta$

$$\begin{array}{ll}
 & [\![\mathsf{let}\ x = [M]\ \mathsf{in}\ Q]\!]_\Delta \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & \mathit{arr\ dup} \ggg \mathit{first}\ (\mathit{arr}\ (\lambda\Delta.\, [\![M]\!])) \ggg \mathit{arr\ swap} \ggg [\![Q]\!]_{\Delta,x} \\
= & \qquad (\rightsquigarrow_5) \\
 & \mathit{arr\ dup} \ggg \mathit{arr}\ ((\lambda\Delta.\, [\![M]\!]) \times \mathit{id}) \ggg \mathit{arr\ swap} \ggg [\![Q]\!]_{\Delta,x} \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ (\mathit{swap} \cdot ((\lambda\Delta.\, [\![M]\!]) \times \mathit{id}) \cdot \mathit{dup}) \ggg [\![Q]\!]_{\Delta,x} \\
= & \qquad (\mathit{swap} \cdot ((\lambda\Delta.\, [\![M]\!]) \times \mathit{id}) \cdot \mathit{dup} = \lambda\Delta.\, \langle \Delta, [\![M]\!] \rangle) \\
 & \mathit{arr}\ (\lambda\Delta.\, \langle \Delta, [\![M]\!] \rangle) \ggg [\![Q]\!]_{\Delta,x} \\
= & \qquad (\text{lemma } 5) \\
 & [\![Q[x := M]]\!]_\Delta
\end{array}$$

2. $[\![\mathsf{let}\ x = P\ \mathsf{in}\ [x]]\!]_\Delta = [\![P]\!]_\Delta$

$$\begin{array}{ll}
 & [\![\mathsf{let}\ x = P\ \mathsf{in}\ [x]]\!]_\Delta \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & (\mathit{arr\ id} \mathbin{\&\&\&} [\![P]\!]_\Delta) \ggg \mathit{arr\ snd} \\
= & \qquad (\text{def } \&\&\&) \\
 & \mathit{arr\ dup} \ggg \mathit{first}\ (\mathit{arr\ id}) \ggg \mathit{second}\ [\![P]\!]_\Delta \ggg \mathit{arr\ snd} \\
= & \qquad (\rightsquigarrow_5) \\
 & \mathit{arr\ dup} \ggg \mathit{arr}\ (\mathit{id} \times \mathit{id}) \ggg \mathit{second}\ [\![P]\!]_\Delta \ggg \mathit{arr\ snd} \\
= & \qquad (\mathit{id} \times \mathit{id} = \mathit{id}) \\
 & \mathit{arr\ dup} \ggg \mathit{arr\ id} \ggg \mathit{second}\ [\![P]\!]_\Delta \ggg \mathit{arr\ snd} \\
= & \qquad (\rightsquigarrow_1) \\
 & \mathit{arr\ dup} \ggg \mathit{second}\ [\![P]\!]_\Delta \ggg \mathit{arr\ snd} \\
= & \qquad (\text{def } \mathit{second}) \\
 & \mathit{arr\ dup} \ggg \mathit{arr\ swap} \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr\ swap} \ggg \mathit{arr\ snd} \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ (\mathit{swap} \cdot \mathit{dup}) \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr}\ (\mathit{snd} \cdot \mathit{swap}) \\
= & \qquad (\mathit{swap} \cdot \mathit{dup} = \mathit{dup},\ \ \mathit{snd} \cdot \mathit{swap} = \mathit{fst}) \\
 & \mathit{arr\ dup} \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr\ fst} \\
= & \qquad (\rightsquigarrow_8) \\
 & \mathit{arr\ dup} \ggg \mathit{arr\ fst} \ggg [\![P]\!]_\Delta \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ (\mathit{fst} \cdot \mathit{dup}) \ggg [\![P]\!]_\Delta \\
= & \qquad (\mathit{fst} \cdot \mathit{dup} = \mathit{id}) \\
 & \mathit{arr\ id} \ggg [\![P]\!]_\Delta \\
= & \qquad (\rightsquigarrow_1) \\
 & [\![P]\!]_\Delta
\end{array}$$

3. $[\![\lambda^\bullet x.\, (L \bullet x)]\!] = [\![L]\!]$

$$
\begin{array}{ll}
 & [\![\lambda^{\bullet} x.\,(L \bullet x)]\!] \\
= & \qquad (\text{def } [\![-]\!]) \\
 & \mathit{arr}\ \mathit{id} \ggg [\![L]\!] \\
= & \qquad (\rightsquigarrow_1) \\
 & [\![L]\!]
\end{array}
$$

4. $[\![\mathsf{let}\ y = (\mathsf{let}\ x = P\ \mathsf{in}\ Q)\ \mathsf{in}\ R]\!]_\Delta = [\![\mathsf{let}\ x = P\ \mathsf{in}\ \mathsf{let}\ y = Q\ \mathsf{in}\ R]\!]_\Delta$

$$
\begin{array}{ll}
 & [\![\mathsf{let}\ y = (\mathsf{let}\ x = P\ \mathsf{in}\ Q)\ \mathsf{in}\ R]\!]_\Delta \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr}\ \mathit{swap} \ggg [\![Q]\!]_{\Delta,x}) \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_6) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{dup}) \ggg \mathit{first}\ (\mathit{first}\ [\![P]\!]_\Delta) \ggg \mathit{first}\ (\mathit{arr}\ \mathit{swap}) \ggg \mathit{first}\ ([\![Q]\!]_{\Delta,x}) \ggg \\
 & \quad \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_5) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{dup} \times \mathit{id}) \ggg \mathit{first}\ (\mathit{first}\ [\![P]\!]_\Delta) \ggg \mathit{arr}\ (\mathit{swap} \times \mathit{id}) \ggg \mathit{first}\ ([\![Q]\!]_{\Delta,x}) \ggg \\
 & \quad \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ ((\mathit{dup} \times \mathit{id}) \cdot \mathit{dup}) \ggg \mathit{first}\ (\mathit{first}\ [\![P]\!]_\Delta) \ggg \mathit{arr}\ (\mathit{first}\ \mathit{swap}) \ggg \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \\
 & \quad \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\mathit{first}\ \mathit{swap} = (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle) \cdot \mathit{assoc}) \\
 & \mathit{arr}\ ((\mathit{dup} \times \mathit{id}) \cdot \mathit{dup}) \ggg \mathit{first}\ (\mathit{first}\ [\![P]\!]_\Delta) \ggg \mathit{arr}\ (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle \cdot \mathit{assoc}) \ggg \\
 & \quad \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ ((\mathit{dup} \times \mathit{id}) \cdot \mathit{dup}) \ggg \mathit{first}\ (\mathit{first}\ [\![P]\!]_\Delta) \ggg \mathit{arr}\ \mathit{assoc} \ggg \mathit{arr}\ (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle) \ggg \\
 & \quad \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_9) \\
 & \mathit{arr}\ ((\mathit{dup} \times \mathit{id}) \cdot \mathit{dup}) \ggg \mathit{arr}\ \mathit{assoc} \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr}\ (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle) \ggg \\
 & \quad \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_r, \mathit{assoc} \cdot (\mathit{dup} \times \mathit{id}) \cdot \mathit{dup} = (\mathit{id} \times \mathit{dup}) \cdot \mathit{dup}) \\
 & \mathit{arr}\ ((\mathit{id} \times \mathit{dup}) \cdot \mathit{dup}) \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr}\ (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle) \ggg \\
 & \quad \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_4) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{id} \times \mathit{dup}) \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr}\ (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle) \ggg \\
 & \quad \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y} \\
= & \qquad (\rightsquigarrow_7) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ [\![P]\!]_\Delta \ggg \mathit{arr}\ (\mathit{id} \times \mathit{dup}) \ggg \mathit{arr}\ (\lambda\langle x, \langle y, z\rangle\rangle.\,\langle\langle y, x\rangle, z\rangle) \ggg \\
 & \quad \mathit{first}\ [\![Q]\!]_{\Delta,x} \ggg \mathit{arr}\ \mathit{swap} \ggg [\![R]\!]_{\Delta,y}
\end{array}
$$

$$
\begin{array}{ll}
= & \quad (\rightsquigarrow_4) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ ((\lambda\langle x,\langle y,z\rangle\rangle.\,\langle\langle y,x\rangle,z\rangle)\cdot(id\times dup)) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ swap \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (\lambda\langle x,\langle y,z\rangle\rangle.\,\langle\langle y,x\rangle,z\rangle)\cdot(id\times dup) = ((id\times fst)\cdot(dup\cdot swap)) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ ((id\times fst)\cdot(dup\cdot swap)) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ swap \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (\rightsquigarrow_{4},) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ (dup\cdot swap) \ggg \\
& \quad arr\ (id\times fst) \ggg first\ [\![Q]\!]_{\Delta,x} \ggg arr\ swap \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (\rightsquigarrow_7) \\
& arr\ dup \ggg (first\ [\![P]\!]_\Delta \ggg arr\ (dup\cdot swap) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ (id\times fst) \ggg arr\ swap \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (\rightsquigarrow_4) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ (dup\cdot swap) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ (swap\cdot(id\times fst)) \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (swap\cdot(id\times fst) = (\lambda\langle y,\langle\Delta,x\rangle\rangle.\,\langle\Delta,y\rangle)) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ (dup\cdot swap) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ ((\lambda\langle y,\langle\Delta,x\rangle\rangle.\,\langle\Delta,y\rangle)) \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (\rightsquigarrow_4) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ (dup\cdot swap) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ swap \ggg arr\ (\lambda\langle\Delta,x,y\rangle.\,\langle\Delta,y\rangle) \ggg [\![R]\!]_{\Delta,y} \\
= & \quad (\text{lemma } 3) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ (dup\cdot swap) \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ swap \ggg [\![R]\!]_{\Delta,x,y} \\
= & \quad (\rightsquigarrow_4) \\
& arr\ dup \ggg first\ [\![P]\!]_\Delta \ggg arr\ swap \ggg arr\ dup \ggg \\
& \quad first\ [\![Q]\!]_{\Delta,x} \ggg arr\ swap \ggg [\![R]\!]_{\Delta,x,y} \\
= & \quad (\text{def } [\![-]\!]_\Delta) \\
& [\![\mathsf{let}\ x = P\ \mathsf{in}\ (\mathsf{let}\ y = Q\ \mathsf{in}\ R)]\!]_\Delta
\end{array}
$$

5. $[\![(\lambda^\bullet x.\,Q)\bullet M]\!]_\Delta = [\![Q[x:=M]]\!]_\Delta$

$$
\begin{array}{ll}
& [\![(\lambda^\bullet x.\,Q)\bullet M]\!]_\Delta \\
= & \quad (\text{def } [\![-]\!]_\Delta) \\
& arr\ (\lambda\Delta.\,[\![M]\!]) \ggg [\![Q]\!]_x \\
= & \quad (\rightsquigarrow_4) \\
& arr\ (\lambda\Delta.\,\langle\Delta,[\![M]\!]\rangle) \ggg arr\ (\lambda\langle\Delta,x\rangle.\,x) \ggg [\![Q]\!]_x \\
= & \quad (\text{lemma } 3) \\
& arr\ (\lambda\Delta.\,\langle\Delta,[\![M]\!]\rangle) \ggg [\![Q]\!]_{\Delta,x} \\
= & \quad (\text{lemma } 5) \\
& [\![Q[x:=M]]\!]_\Delta
\end{array}
$$

B.3 Translating classic arrows to arrow calculus and back

For each classic arrows term M, $[\![\,[\![M]\!]^{-1}\,]\!] = M$. The proof is by induction on the structure of terms. The only interesting cases are the constants.

1. $[\![\,[\![arr]\!]^{-1}\,]\!] = arr.$

$$
\begin{array}{ll}
& [\![\, [\![arr]\!]^{-1} \,]\!] \\
= & \qquad (\text{def } [\![-]\!]^{-1}, \beta^{\rightarrow}) \\
& [\![\lambda f.\ \lambda^{\bullet} x.\ [f\ x] \,]\!] \\
= & \qquad (\text{def } [\![-]\!]) \\
& \lambda f.\ arr\ (\lambda x.\ (f\ x)) \\
= & \qquad (\eta^{\rightarrow}, \eta^{\rightarrow}) \\
& arr
\end{array}
$$

2. $[\![\, (\ggg) \,]\!] = (\ggg)$.

$$
\begin{array}{ll}
& [\![\, [\![(\ggg)]\!]^{-1} \,]\!] \\
= & \qquad (\text{def } [\![-]\!]^{-1}, \beta^{\rightarrow}) \\
& [\![\lambda f.\ \lambda g.\ \lambda^{\bullet} x.\ (\mathsf{let}\ y = f \bullet x\ \mathsf{in}\ g \bullet y) \,]\!] \\
= & \qquad (\text{def } [\![-]\!], (\rightsquigarrow_1)) \\
& \lambda f.\ \lambda g.\ (arr\ id \mathbin{\&\&\&} f) \ggg (arr\ snd \ggg g) \\
= & \qquad (\text{def } \&\&\&) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg first\ (arr\ id) \ggg arr\ swap \ggg first\ f \ggg arr\ swap \ggg (arr\ snd \ggg g) \\
= & \qquad (\rightsquigarrow_5) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg arr\ (id \times id) \ggg arr\ swap \ggg first\ f \ggg arr\ swap \ggg (arr\ snd \ggg g) \\
= & \qquad (id \times id = id) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg arr\ id \ggg arr\ swap \ggg first\ f \ggg arr\ swap \ggg (arr\ snd \ggg g) \\
= & \qquad (\rightsquigarrow_2) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg arr\ swap \ggg first\ f \ggg arr\ swap \ggg (arr\ snd \ggg g) \\
= & \qquad (\rightsquigarrow_4, swap \cdot dup = dup) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg first\ f \ggg arr\ swap \ggg (arr\ snd \ggg g) \\
= & \qquad (\rightsquigarrow_3) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg first\ f \ggg arr\ swap \ggg arr\ snd \ggg g \\
= & \qquad (\rightsquigarrow_4, snd \cdot swap = fst) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg first\ f \ggg arr\ fst \ggg g \\
= & \qquad (\rightsquigarrow_8) \\
& \lambda f.\ \lambda g.\ arr\ dup \ggg arr\ fst \ggg f \ggg g \\
= & \qquad (\rightsquigarrow_4, fst \cdot dup = id) \\
& \lambda f.\ \lambda g.\ arr\ id \ggg f \ggg g \\
= & \qquad (\rightsquigarrow_1, \eta^{\rightarrow}(\times 2)) \\
& (\ggg)
\end{array}
$$

3. $[\![\, [\![first]\!]^{-1} \,]\!] = first$.

$$
\begin{array}{ll}
& [\![\, [\![\mathit{first}\ f]\!]^{-1}\,]\!] \\
= & \qquad (\text{def } [\![-]\!]^{-1}, \beta^{\rightarrow}) \\
& [\![\lambda f.\ \lambda^{\bullet} z.\, \mathsf{let}\ x = f \bullet [\mathsf{fst}\ z]\ \mathsf{in}\ [\langle x, \mathsf{snd}\ z\rangle]\,]\!] \\
= & \qquad (\text{def } [\![-]\!]) \\
& \lambda f.\, (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} (\mathit{arr}\ (\lambda u.\, \mathsf{fst}\ u) \ggg f)) \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\text{def } \mathit{fst}) \\
& \lambda f.\, (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} (\mathit{arr}\ \mathit{fst} \ggg f)) \ggg \mathit{arr}\ (\lambda v.\, (\mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v))) \\
= & \qquad (\text{def } \&\&\&) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{id}) \ggg \mathit{second}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\rightsquigarrow_5) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{id} \times \mathit{id}) \ggg \mathit{second}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\mathit{id} \times \mathit{id} = \mathit{id}) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ \mathit{id} \ggg \mathit{second}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\rightsquigarrow_1) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{second}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\text{def } \mathit{second}) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\rightsquigarrow_4) \\
& \lambda f.\, \mathit{arr}\ (\mathit{swap} \cdot \mathit{dup}) \ggg \mathit{first}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\mathit{swap} \cdot \mathit{dup} = \mathit{dup}) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{arr}\ (\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \\
= & \qquad (\rightsquigarrow_4) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ ((\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \cdot \mathit{swap}) \\
= & \qquad ((\lambda v.\, \langle \mathsf{snd}\ v, \mathsf{snd}\ (\mathsf{fst}\ v)\rangle) \cdot \mathit{swap} = \mathit{id} \times \mathit{snd}) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{fst} \ggg f) \ggg \mathit{arr}\ (\mathit{id} \times \mathit{snd}) \\
= & \qquad (\rightsquigarrow_6) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{fst}) \ggg \mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \times \mathit{snd}) \\
= & \qquad (\rightsquigarrow_5) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{fst} \times \mathit{id}) \ggg \mathit{first}\ f \ggg \mathit{arr}\ (\mathit{id} \times \mathit{snd}) \\
= & \qquad (\rightsquigarrow_7) \\
& \lambda f.\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{fst} \times \mathit{id}) \ggg \mathit{arr}\ (\mathit{id} \times \mathit{snd}) \ggg \mathit{first}\ f \\
= & \qquad (\rightsquigarrow_4) \\
& \lambda f.\, \mathit{arr}\ ((\mathit{id} \times \mathit{snd}) \cdot (\mathit{fst} \times \mathit{id}) \cdot \mathit{dup}) \ggg \mathit{first}\ f \\
= & \qquad ((\mathit{id} \times \mathit{snd}) \cdot (\mathit{fst} \times \mathit{id}) \cdot \mathit{dup} = \mathit{id}) \\
& \lambda f.\, \mathit{arr}\ \mathit{id} \ggg \mathit{first}\ f \\
= & \qquad (\rightsquigarrow_1, \eta^{\rightarrow}) \\
& \mathit{first}
\end{array}
$$

B.4 Translating arrow calculus to classic arrows and back

For each arrow calculus term M, $[\![\, [\![M]\!]\,]\!]^{-1} = M$.

1. $[\![\, [\![\lambda^{\bullet} x.\, Q]\!]\,]\!]^{-1} = \lambda^{\bullet} x.\, Q$.

$$
\begin{array}{ll}
& [\![\, [\![\lambda^{\bullet} x.\, Q]\!]\,]\!]^{-1} \\
= & \qquad (\text{def } [\![-]\!]) \\
& [\![\, [\![Q]\!]_x\,]\!]^{-1} \\
= & \qquad (\text{induction hypothesis}) \\
& \lambda^{\bullet} x.\, Q
\end{array}
$$

For each arrow calculus command P, $[\![\, [\![P]\!]_\Delta \,]\!]^{-1} \bullet \Delta = P$.

1. $[\![\, [\![L \bullet M]\!]_\Delta \,]\!]^{-1} \bullet \Delta = L \bullet M$.

$$\begin{array}{ll}
 & [\![\, [\![L \bullet M]\!]_\Delta \,]\!]^{-1} \bullet \Delta \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & [\![\, arr\ (\lambda\Delta.\, [\![M]\!]) \ggg [\![L]\!] \,]\!]^{-1} \bullet \Delta \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^\bullet\Delta.\, (\mathsf{let}\ y = (\lambda^\bullet z.\, [(\lambda\Delta.\, [\![[\![M]\!]]\!]^{-1})\ z]) \bullet \Delta\ \mathsf{in}\ [\![[\![L]\!]]\!]^{-1} \bullet y) \bullet \Delta \\
= & \qquad (\text{induction hypothesis}) \\
 & \lambda^\bullet\Delta.\, (\mathsf{let}\ y = (\lambda^\bullet z.\, [(\lambda\Delta.\, M)\ z]) \bullet \Delta\ \mathsf{in}\ L \bullet y) \bullet \Delta \\
= & \qquad (\beta^{\leadsto}) \\
 & \mathsf{let}\ y = [(\lambda\Delta.\, M)\ \Delta]\ \mathsf{in}\ L \bullet y \\
= & \qquad (\beta^{\to}) \\
 & \mathsf{let}\ y = [M]\ \mathsf{in}\ L \bullet y \\
= & \qquad (\text{left}) \\
 & L \bullet M
\end{array}$$

2. $[\![\, [\![[M]]\!]_\Delta \,]\!]^{-1} \bullet \Delta = [M]$.

$$\begin{array}{ll}
 & [\![\, [\![[M]]\!]_\Delta \,]\!]^{-1} \bullet \Delta \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & [\![\, arr\ (\lambda\Delta.\, [\![M]\!]) \,]\!]^{-1} \bullet \Delta \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^\bullet\Delta.\, [(\lambda\Delta.\, [\![[\![M]\!]]\!]^{-1})\ \Delta] \bullet \Delta \\
= & \qquad (\text{induction hypothesis}) \\
 & \lambda^\bullet\Delta.\, [(\lambda\Delta.\, M)\ \Delta] \bullet \Delta \\
= & \qquad (\beta^{\leadsto}) \\
 & [(\lambda\Delta.\, M)\ \Delta] \\
= & \qquad (\beta^{\to}) \\
 & [M]
\end{array}$$

3. $[\![\, [\![\mathsf{let}\ x = P\ \mathsf{in}\ Q]\!]_\Delta \,]\!]^{-1} \bullet \Delta = \mathsf{let}\ x = P\ \mathsf{in}\ Q$.

$$
\begin{aligned}
& \llbracket\, \llbracket \mathsf{let}\ x = P\ \mathsf{in}\ Q \rrbracket_{\Delta}\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\text{def } \llbracket - \rrbracket_{\Delta}) \\
& \llbracket\, (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \llbracket P \rrbracket_{\Delta}) \ggg \llbracket Q \rrbracket_{\Delta,x},\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\text{def } \&\&\&) \\
& \llbracket\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ \mathit{id}) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta} \ggg \mathit{arr}\ \mathit{swap} \ggg \llbracket Q \rrbracket_{\Delta,x}\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\leadsto_5) \\
& \llbracket\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ (\mathit{id} \times \mathit{id}) \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta} \ggg \mathit{arr}\ \mathit{swap} \ggg \llbracket Q \rrbracket_{\Delta,x}\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\mathit{id} \times \mathit{id} = \mathit{id}) \\
& \llbracket\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ \mathit{id} \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta} \ggg \mathit{arr}\ \mathit{swap} \ggg \llbracket Q \rrbracket_{\Delta,x}\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\leadsto_2) \\
& \llbracket\, \mathit{arr}\ \mathit{dup} \ggg \mathit{arr}\ \mathit{swap} \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta} \ggg \mathit{arr}\ \mathit{swap} \ggg \llbracket Q \rrbracket_{\Delta,x}\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\leadsto_4, \mathit{swap} \cdot \mathit{dup} = \mathit{dup}) \\
& \llbracket\, \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ \llbracket P \rrbracket_{\Delta} \ggg \mathit{arr}\ \mathit{swap} \ggg \llbracket Q \rrbracket_{\Delta,x}\, \rrbracket^{-1} \bullet \Delta \\
= \quad & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
& \lambda^{\bullet}\Delta.\, (\mathsf{let}\ x = (\lambda^{\bullet} z.\, [\mathit{dup}\ z]) \bullet \Delta\ \mathsf{in} \\
& \qquad\qquad (\lambda^{\bullet}\Delta'.\, (\mathsf{let}\ w = (\lambda^{\bullet} z.\, (\mathsf{let}\ v = \llbracket \llbracket P \rrbracket_{\Delta} \rrbracket^{-1} \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle v, \mathsf{snd}\ z \rangle])) \bullet \Delta'\ \mathsf{in} \\
& \qquad\qquad\qquad\qquad (\lambda^{\bullet}\Delta''.\, (\mathsf{let}\ y = (\lambda^{\bullet} z.\, [\mathit{swap}\ z]) \bullet \Delta''\ \mathsf{in}\ \llbracket \llbracket Q \rrbracket_{\Delta,x} \rrbracket^{-1} \bullet y)) \bullet w)) \bullet x) \bullet \Delta \\
= \quad & \qquad (\beta^{\leadsto}(\times 6)) \\
& \mathsf{let}\ x = [\mathit{dup}\ \Delta]\ \mathsf{in} \\
& \quad (\mathsf{let}\ w = (\mathsf{let}\ v = \llbracket \llbracket P \rrbracket_{\Delta} \rrbracket^{-1} \bullet \mathsf{fst}\ x\ \mathsf{in}\ [\langle v, \mathsf{snd}\ x \rangle])\ \mathsf{in}\ \mathsf{let}\ y = [\mathit{swap}\ w]\ \mathsf{in}\ \llbracket \llbracket Q \rrbracket_{\Delta,x} \rrbracket^{-1} \bullet y) \\
= \quad & \qquad (\text{left}(\times 2)) \\
& \mathsf{let}\ w = (\mathsf{let}\ x = \llbracket \llbracket P \rrbracket_{\Delta} \rrbracket^{-1} \bullet \mathsf{fst}\ \mathit{dup}\ \Delta\ \mathsf{in}\ [\langle x, \mathsf{snd}\ \mathit{dup}\ \Delta \rangle])\ \mathsf{in}\ \llbracket \llbracket Q \rrbracket_{\Delta,x} \rrbracket^{-1} \bullet \mathit{swap}\ w \\
= \quad & \qquad (\text{assoc}) \\
& \mathsf{let}\ x = \llbracket \llbracket P \rrbracket_{\Delta} \rrbracket^{-1} \bullet \mathsf{fst}\ \mathit{dup}\ \Delta\ \mathsf{in}\ \mathsf{let}\ w = [\langle x, \mathsf{snd}\ \mathit{dup}\ \Delta \rangle]\ \mathsf{in}\ \llbracket \llbracket Q \rrbracket_{\Delta,x} \rrbracket^{-1} \bullet \mathit{swap}\ w \\
= \quad & \qquad (\text{left}) \\
& \mathsf{let}\ x = \llbracket \llbracket P \rrbracket_{\Delta} \rrbracket^{-1} \bullet \mathsf{fst}\ \mathit{dup}\ \Delta\ \mathsf{in}\ \llbracket \llbracket Q \rrbracket_{\Delta,x} \rrbracket^{-1} \bullet \mathit{swap}\ \langle x, \mathsf{snd}\ \mathit{dup}\ \Delta \rangle \\
= \quad & \qquad (\text{def } \mathit{swap}, \mathit{dup}, \beta^{\to}, \beta^{\times}_1, \beta^{\times}_2) \\
& \mathsf{let}\ x = \llbracket \llbracket P \rrbracket_{\Delta} \rrbracket^{-1} \bullet \Delta\ \mathsf{in}\ \llbracket \llbracket Q \rrbracket_{\Delta,x} \rrbracket^{-1} \bullet \langle \Delta, x \rangle \\
= \quad & \qquad (\text{induction hypothesis}) \\
& \mathsf{let}\ x = P\ \mathsf{in}\ Q
\end{aligned}
$$

C Higher-order arrows proofs

C.1 The arrow laws follow from the laws of the metalanguage

$M = N$ implies $\llbracket M \rrbracket^{-1} = \llbracket N \rrbracket^{-1}$.

$\leadsto_{H1}$:

$$
\begin{array}{ll}
 & \llbracket \mathit{first}\ (\mathit{arr}\ (\lambda x.\ \mathit{arr}\ (\lambda y.\ \langle x,y\rangle))) \ggg \mathit{app}\rrbracket^{-1} \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ b = (\lambda f.\ \lambda^\bullet z.\ \mathsf{let}\ c = f \bullet (\mathsf{fst}\ z)\ \mathsf{in}\ [\langle c, \mathsf{snd}\ z\rangle])(\lambda^\bullet x.\ [\lambda^\bullet y.\ [\langle x,y\rangle]]) \bullet a\ \mathsf{in} \\
 & \qquad (\lambda^\bullet z.\ \mathsf{fst}\ z \star \mathsf{snd}\ z) \bullet b \\
= & \qquad (\beta^{\to}, \beta^{\rightsquigarrow}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ b = (\mathsf{let}\ c = (\lambda^\bullet x.\ [\lambda^\bullet y.\ [\langle x,y\rangle]]) \bullet (\mathsf{fst}\ a)\ \mathsf{in}\ [\langle c, \mathsf{snd}\ a\rangle])\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\beta^{\rightsquigarrow}, \text{assoc}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ c = [\lambda^\bullet y.\ [\langle \mathsf{fst}\ a, y\rangle]]\ \mathsf{in}\ \mathsf{let}\ b = [\langle c, \mathsf{snd}\ a\rangle]\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\text{left}, \text{left}, \beta^{\times}, \beta^{\times}) \\
 & \lambda^\bullet a.\ (\lambda^\bullet y.\ [\langle \mathsf{fst}\ a, y\rangle]) \star \mathsf{snd}\ a \\
= & \qquad (\beta^{app}) \\
 & \lambda^\bullet a.\ [\langle \mathsf{fst}\ a, \mathsf{snd}\ a\rangle] \\
= & \qquad (\eta^{\times}) \\
 & \lambda^\bullet a.\ [a] \\
= & \\
 & \llbracket \mathit{arr}\ \mathit{id} \rrbracket^{-1}
\end{array}
$$

$\rightsquigarrow_{H2}$:

$$
\begin{array}{ll}
 & \llbracket \mathit{first}\ (\mathit{arr}\ (g \ggg)) \ggg \mathit{app}\rrbracket^{-1} \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ b = (\lambda^\bullet z.\ \mathsf{let}\ x = (\lambda^\bullet f.\ [\llbracket g \ggg f\rrbracket^{-1}]) \bullet \mathsf{fst}\ z\ \mathsf{in}\ \langle x, \mathsf{snd}\ z\rangle) \bullet a\ \mathsf{in}\ (\lambda^\bullet w.\ \mathsf{fst}\ w \star \mathsf{snd}\ w) \bullet b \\
= & \qquad (\beta^{\rightsquigarrow}, \beta^{\rightsquigarrow}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ b = (\mathsf{let}\ x = (\lambda^\bullet f.\ [\llbracket g \ggg f\rrbracket^{-1}]) \bullet \mathsf{fst}\ a\ \mathsf{in}\ [\langle x, \mathsf{snd}\ a\rangle])\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\text{assoc}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ x = (\lambda^\bullet f.\ [\llbracket g \ggg f\rrbracket^{-1}]) \bullet \mathsf{fst}\ a\ \mathsf{in}\ \mathsf{let}\ b = [\langle x, \mathsf{snd}\ a\rangle]\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\beta^{\rightsquigarrow}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ x = [\llbracket g \ggg \mathsf{fst}\ a\rrbracket^{-1}]\ \mathsf{in}\ \mathsf{let}\ b = [\langle x, \mathsf{snd}\ a\rangle]\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\text{left}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ x = [\llbracket g \ggg \mathsf{fst}\ a\rrbracket^{-1}]\ \mathsf{in}\ x \star \mathsf{snd}\ a \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ x = [\lambda^\bullet d.\ \mathsf{let}\ y = g \bullet d\ \mathsf{in}\ \mathsf{fst}\ a \bullet y]\ \mathsf{in}\ x \star \mathsf{snd}\ a \\
= & \qquad (\eta^{app}, \beta^{\rightsquigarrow}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ x = [\lambda^\bullet d.\ \mathsf{let}\ y = g \bullet d\ \mathsf{in}\ \mathsf{fst}\ a \star y]\ \mathsf{in}\ x \star \mathsf{snd}\ a \\
= & \qquad (\text{left}, \beta^{app}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ y = g \bullet \mathsf{snd}\ a\ \mathsf{in}\ \mathsf{fst}\ a \star y \\
= & \qquad (\text{left}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ y = g \bullet \mathsf{snd}\ a\ \mathsf{in}\ \mathsf{let}\ b = [\langle \mathsf{fst}\ a, y\rangle]\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\text{assoc}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ b = (\mathsf{let}\ y = g \bullet \mathsf{snd}\ a\ \mathsf{in}\ [\langle \mathsf{fst}\ a, y\rangle])\ \mathsf{in}\ \mathsf{fst}\ b \star \mathsf{snd}\ b \\
= & \qquad (\beta^{\rightsquigarrow}, \beta^{\rightsquigarrow}) \\
 & \lambda^\bullet a.\ \mathsf{let}\ b = (\lambda^\bullet z.\ \mathsf{let}\ y = g \bullet \mathsf{snd}\ z\ \mathsf{in}\ [\langle \mathsf{fst}\ z, y\rangle]) \bullet a\ \mathsf{in}\ (\lambda^\bullet w.\ \mathsf{fst}\ w \star \mathsf{snd}\ w) \bullet b \\
= & \qquad (\text{def } \llbracket - \rrbracket^{-1}) \\
 & \llbracket \mathit{second}\ g \ggg \mathit{app}\rrbracket^{-1}
\end{array}
$$

$\rightsquigarrow_{H3}$:

$$\begin{array}{ll}
 & [\![\mathit{first}\ (\mathit{arr}\ ({\ggg}\ h)) \ggg \mathit{app}]\!]^{-1} \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^\bullet a.\, \mathsf{let}\ b = (\lambda^\bullet z.\, \mathsf{let}\ x = (\lambda^\bullet f.\, [[\![f \ggg h]\!]^{-1}]) \bullet \mathsf{fst}\ z\ \mathsf{in}\ [\langle x, \mathsf{snd}\ z\rangle]) \bullet a\ \mathsf{in}\ (\lambda^\bullet w.\, \mathsf{fst}\ w \star \mathsf{snd}\ w) \bullet b \\
= & \qquad (\text{as before}) \\
 & \lambda^\bullet a.\, \mathsf{let}\ x = [[\![\mathsf{fst}\ a \ggg h]\!]^{-1}]\ \mathsf{in}\ x \star \mathsf{snd}\ a \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^\bullet a.\, \mathsf{let}\ x = [\lambda^\bullet d.\, \mathsf{let}\ e = \mathsf{fst}\ a \bullet d\ \mathsf{in}\ h \bullet e]\ \mathsf{in}\ x \star \mathsf{snd}\ a \\
= & \qquad (\eta^{app}, \beta^{\leadsto}) \\
 & \lambda^\bullet a.\, \mathsf{let}\ x = [\lambda^\bullet d.\, \mathsf{let}\ e = \mathsf{fst}\ a \star d\ \mathsf{in}\ h \bullet e]\ \mathsf{in}\ x \star \mathsf{snd}\ a \\
= & \qquad (\text{left}, \beta^{\leadsto}) \\
 & \lambda^\bullet a.\, \mathsf{let}\ e = \mathsf{fst}\ a \star \mathsf{snd}\ a\ \mathsf{in}\ h \bullet e \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & [\![\mathit{app} \ggg h]\!]^{-1}
\end{array}$$

C.2 The laws of the metalanguage follow from the arrow laws

$M = N$ implies $[\![M]\!] = [\![N]\!]$ and $P = Q$ implies $[\![P]\!]_\Delta = [\![Q]\!]_\Delta$.

β^{app}:

$$\begin{array}{ll}
 & [\![(\lambda^\bullet x.\, Q) \star M]\!]_\Delta \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & \mathit{arr}\ (\lambda\Delta.\, ([\![Q]\!]_x, [\![M]\!])) \ggg \mathit{app} \\
= & \qquad (\text{classic arrow laws}) \\
 & (\mathit{arr}\ (\lambda\Delta.\, [\![Q]\!]_x) \mathbin{\&\&\&} \mathit{arr}\ (\lambda\Delta.\, [\![M]\!])) \ggg \mathit{app} \\
= & \qquad (\beta^{\leadsto}) \\
 & (\mathit{arr}\ (\lambda\Delta.\, [\![(\lambda^\bullet\langle\Delta, x\rangle.\, Q) \bullet \langle\Delta, x\rangle]\!]_x) \mathbin{\&\&\&} \mathit{arr}\ (\lambda\Delta.\, [\![M]\!])) \ggg \mathit{app} \\
= & \qquad (\text{def } [\![-]\!]_x) \\
 & (\mathit{arr}\ (\lambda\Delta.\, \mathit{arr}\ (\lambda x.\, \langle\Delta, x\rangle) \ggg [\![Q]\!]_{\langle\Delta,x\rangle}) \mathbin{\&\&\&} \mathit{arr}\ (\lambda\Delta.\, [\![M]\!])) \ggg \mathit{app} \\
= & \qquad (\text{let } j = \lambda x.\, \mathit{arr}\ (\lambda y.\, (x,y)), p = \mathit{arr}\ (\lambda\Delta.\, [\![M]\!]), q = [\![Q]\!]_{\langle\Delta,x\rangle}) \\
 & (\mathit{arr}\ (({\ggg}\ q) \cdot j) \mathbin{\&\&\&} p) \ggg \mathit{app} \\
= & \qquad (\text{def } \&\&\&) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{first}\ (\mathit{arr}\ (({\ggg}\ q) \cdot j))) \ggg \mathit{second}\ p \ggg \mathit{app} \\
= & \qquad (\text{classic arrow laws}) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{second}\ p \ggg \mathit{first}\ (\mathit{arr}\ (({\ggg}\ q) \cdot j))) \ggg \mathit{app} \\
= & \qquad (\text{classic arrow laws}) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{second}\ p \ggg \mathit{first}\ (\mathit{arr}\ j) \ggg \mathit{first}\ (\mathit{arr}\ ({\ggg}\ q)) \ggg \mathit{app} \\
= & \qquad (\leadsto_{H3}) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{second}\ p \ggg \mathit{first}\ (\mathit{arr}\ j) \ggg \mathit{app} \ggg q \\
= & \qquad (\leadsto_{H1}) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{second}\ p \ggg \mathit{arr}\ \mathit{id} \ggg q \\
= & \qquad (\text{classic arrow laws}) \\
 & \mathit{arr}\ \mathit{dup} \ggg \mathit{second}\ p \ggg q \\
= & \qquad (\text{classic arrow laws}) \\
 & (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} p) \ggg q \\
= & \qquad (p = \mathit{arr}\ (\lambda\Delta.\, [\![M]\!]), q = [\![Q]\!]_{\langle\Delta,x\rangle}) \\
 & (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \mathit{arr}\ (\lambda\Delta.\, [\![M]\!])) \ggg [\![Q]\!]_{\langle\Delta,x\rangle} \\
= & \qquad (\text{Lemma 5}) \\
 & [\![Q[x := M]]\!]_\Delta
\end{array}$$

C.3 Translating classic arrows to arrow calculus and back

$[\![\, [\![M]\!]^{-1} \,]\!] = M$.

Case *app*:

$$
\begin{array}{ll}
 & [\![[\![app]\!]^{-1}]\!] \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & [\![\lambda^\bullet z.\, \mathsf{fst}\ z \star \mathsf{snd}\ z]\!] \\
= & \qquad (\text{def } [\![-]\!]) \\
 & arr\ (\lambda z.\, \langle \mathsf{fst}\ z, \mathsf{snd}\ z\rangle) \ggg app \\
= & \qquad (\eta^\times) \\
 & arr\ id \ggg app \\
= & \qquad (\rightsquigarrow_1) \\
 & app
\end{array}
$$

C.4 Translating arrow calculus to classic arrows and back

$[\![\, [\![M]\!] \,]\!]^{-1} = M$ and $[\![\, [\![P]\!]_\Delta \,]\!]^{-1} = \lambda^\bullet \Delta.\, P$.

Case $L \star M$:

$$
\begin{array}{ll}
 & [\![[\![L \star M]\!]_\Delta]\!]^{-1} \\
= & \qquad (\text{def } [\![-]\!]_\Delta) \\
 & [\![arr\ (\lambda \Delta.\, \langle [\![L]\!], [\![M]\!] \rangle) \ggg app]\!]^{-1} \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & \lambda^\bullet \Delta.\, \mathsf{let}\ y = (\lambda^\bullet \Delta.\, [\langle [\![[\![L]\!]]\!]^{-1}, [\![[\![M]\!]]\!]^{-1} \rangle]) \bullet \Delta\ \mathsf{in}\ \mathsf{fst}\ y \star \mathsf{snd}\ y \\
= & \qquad (\text{induction hypothesis}) \\
 & \lambda^\bullet \Delta.\, \mathsf{let}\ y = (\lambda^\bullet \Delta.\, [\langle L, M \rangle]) \bullet \Delta\ \mathsf{in}\ \mathsf{fst}\ y \star \mathsf{snd}\ y \\
= & \qquad (\beta^\rightsquigarrow) \\
 & \lambda^\bullet \Delta.\, \mathsf{let}\ y = [\langle L, M \rangle]\ \mathsf{in}\ \mathsf{fst}\ y \star \mathsf{snd}\ y \\
= & \qquad (\text{left}, \beta^\times, \beta^\times) \\
 & \lambda^\bullet \Delta.\, L \star M
\end{array}
$$

D Static arrows proofs

D.1 The arrow laws follow from the laws of the metalanguage

$M = N$ implies $[\![M]\!]^{-1} = [\![N]\!]^{-1}$.

$\rightsquigarrow_{S1}$:

$\llbracket \mathit{force}\ (\mathit{delay}\ a) \rrbracket^{-1}$

$=$ (def force)

$\llbracket \mathit{arr}\ (\lambda x.\, \langle\langle\rangle, x\rangle) \ggg \mathit{delay}\ a \ggg \mathit{arr}\ (\lambda z.\, \mathsf{fst}\ z\ (\mathsf{snd}\ z)) \rrbracket^{-1}$

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\lambda^{\bullet} x.\, \mathsf{let}\ y = [\langle\langle\rangle, x\rangle]\ \mathsf{in}\ \mathsf{let}\ z = (\mathsf{let}\ w = (\lambda^{\bullet} u.\, \mathsf{run}\ a) \bullet (\mathsf{fst}\ y)\ \mathsf{in}\ [\langle w, \mathsf{snd}\ y\rangle])\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)]$

$=$ ($\beta^{\leadsto}$)

$\lambda^{\bullet} x.\, \mathsf{let}\ y = [\langle\langle\rangle, x\rangle]\ \mathsf{in}\ \mathsf{let}\ z = (\mathsf{let}\ w = \mathsf{run}\ a\ \mathsf{in}\ [\langle w, \mathsf{snd}\ y\rangle])\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)]$

$=$ (left, $\beta^{\times}$)

$\lambda^{\bullet} x.\, \mathsf{let}\ z = (\mathsf{let}\ w = \mathsf{run}\ a\ \mathsf{in}\ [\langle w, x\rangle])\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)]$

$=$ (assoc)

$\lambda^{\bullet} x.\, \mathsf{let}\ w = \mathsf{run}\ a\ \mathsf{in}\ \mathsf{let}\ z = [\langle w, x\rangle]\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)]$

$=$ (left, $\beta^{\times}, \beta^{\times}$)

$\lambda^{\bullet} x.\, \mathsf{let}\ w = \mathsf{run}\ a\ \mathsf{in}\ [w\ x]$

$=$ (ob_1)

$\lambda^{\bullet} x.\, a \bullet x$

$=$ ($\eta^{\leadsto}$)

a

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\llbracket a \rrbracket^{-1}$

$\leadsto_{S2}$:

$\llbracket \mathit{delay}\ (\mathit{force}\ a) \rrbracket^{-1}$

$=$ (def force)

$\llbracket \mathit{delay}\ (\mathit{arr}\ (\lambda x.\, \langle\langle\rangle, x\rangle) \ggg a \ggg \mathit{arr}\ (\lambda z.\, (\mathsf{fst}\ z\ (\mathsf{snd}\ z)))) \rrbracket^{-1}$

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\lambda^{\bullet} u.\, \mathsf{run}\ (\lambda^{\bullet} x.\, \mathsf{let}\ y = [\langle\langle\rangle, x\rangle]\ \mathsf{in}\ \mathsf{let}\ z = (\mathsf{let}\ w = a \bullet (\mathsf{fst}\ y)\ \mathsf{in}\ [\langle w, \mathsf{snd}\ y\rangle])\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)])$

$=$ (left, $\beta^{\times}$)

$\lambda^{\bullet} u.\, \mathsf{run}\ (\lambda^{\bullet} x.\, \mathsf{let}\ z = (\mathsf{let}\ w = a \bullet \langle\rangle\ \mathsf{in}\ [\langle w, x\rangle])\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)])$

$=$ (assoc)

$\lambda^{\bullet} u.\, \mathsf{run}\ (\lambda^{\bullet} x.\, \mathsf{let}\ w = a \bullet \langle\rangle\ \mathsf{in}\ \mathsf{let}\ z = [\langle w, x\rangle]\ \mathsf{in}\ [\mathsf{fst}\ z\ (\mathsf{snd}\ z)]$

$=$ (left, $\beta^{\times}, \beta^{\times}$)

$\lambda^{\bullet} u.\, \mathsf{run}\ (\lambda^{\bullet} x.\, \mathsf{let}\ w = a \bullet \langle\rangle\ \mathsf{in}\ [w\ x])$

$=$ (ob_3)

$\lambda^{\bullet} u.\, \mathsf{let}\ w = a \bullet \langle\rangle\ \mathsf{in}\ \mathsf{let}\ f = \mathsf{run}\ (\lambda^{\bullet} \langle x, w\rangle.\, [w\ x])\ \mathsf{in}\ [\lambda x.\, f\ \langle x, w\rangle]$

$=$ (ob_2)

$\lambda^{\bullet} u.\, \mathsf{let}\ w = a \bullet \langle\rangle\ \mathsf{in}\ \mathsf{let}\ f = [\lambda \langle x, w\rangle.\, w\ x]\ \mathsf{in}\ [\lambda x.\, f\ \langle x, w\rangle]$

$=$ (left, $\beta^{\rightarrow}$)

$\lambda^{\bullet} x.\, \mathsf{let}\ w = a \bullet x\ \mathsf{in}\ [\lambda x.\, w\ x]$

$=$ ($\eta^{\rightarrow}$)

$\lambda^{\bullet} x.\, \mathsf{let}\ w = a \bullet x\ \mathsf{in}\ [w]$

$=$ (right)

$\lambda^{\bullet} x.\, a \bullet x$

$=$ ($\eta^{\leadsto}$)

a

$=$ (def $\llbracket - \rrbracket^{-1}$)

$\llbracket a \rrbracket^{-1}$

D.2 The laws of the metalanguage follow from the arrow laws

$M = N$ implies $[\![M]\!] = [\![N]\!]$ and $P = Q$ implies $[\![P]\!]_\Delta = [\![Q]\!]_\Delta$.

ob_1:

$$
\begin{array}{ll}
 & [\![L \bullet M]\!]_\Delta \\
= & \quad (\text{def } [\![-]\!]_\Delta) \\
 & arr\ (\lambda\Delta.\, [\![M]\!]) \ggg [\![L]\!] \\
= & \quad (\rightsquigarrow_{S1}) \\
 & arr\ (\lambda\Delta.\, [\![M]\!]) \ggg force\ (delay\ [\![L]\!]) \\
= & \quad (\text{def } force) \\
 & arr\ (\lambda\Delta.\, [\![M]\!]) \ggg arr\ (\lambda a.\, (\langle\rangle, a)) \ggg first\ (delay\ [\![L]\!]) \ggg arr\ (\lambda\langle f, a\rangle.\, f\ a) \\
= & \quad (\text{classic arrow laws}) \\
 & ((arr\ (\lambda\Delta.\, \langle\rangle) \ggg delay\ [\![L]\!]) \mathbin{\&\&\&} arr\ (\lambda\Delta.\, [\![M]\!])) \ggg arr\ (\lambda\langle f, a\rangle.\, f\ a) \\
= & \quad (\text{classic arrow laws}) \\
 & arr\ (\lambda\Delta.\, [\![M]\!]) \mathbin{\&\&\&} (arr\ (\lambda\Delta.\, \langle\rangle) \ggg delay\ [\![L]\!])) \ggg arr\ (\lambda\langle a, f\rangle.\, f\ a) \\
= & \quad (\text{classic arrow laws}) \\
 & (arr\ id \mathbin{\&\&\&} (arr\ (\lambda\Delta.\, \langle\rangle) \ggg delay\ [\![L]\!])) \ggg arr\ (\lambda\langle \Delta, f\rangle.\, f\ [\![M]\!]) \\
= & \quad (\text{def } [\![-]\!]_\Delta) \\
 & [\![\mathsf{let}\ f = \mathsf{run}\ L\ \mathsf{in}\ [f\ M]]\!]_\Delta
\end{array}
$$

ob_2:

$$
\begin{array}{ll}
 & [\![\mathsf{run}\ (\lambda^\bullet x.\, [M])]\!]_\Delta \\
= & \quad (\text{def } [\![-]\!]_\Delta) \\
 & arr\ (\lambda\Delta.\, \langle\rangle) \ggg delay\ (arr\ (\lambda x.\, [\![M]\!])) \\
= & \quad (\text{def } force, \text{classic arrow laws}) \\
 & arr\ (\lambda\Delta.\, \langle\rangle) \ggg delay\ (force\ (arr\ (\lambda u.\, \lambda x.\, [\![M]\!]))) \\
= & \quad (\rightsquigarrow_{S2}) \\
 & arr\ (\lambda\Delta.\, \langle\rangle) \ggg arr\ (\lambda u.\, \lambda x.\, [\![M]\!]) \\
= & \quad (\text{classic arrow laws}) \\
 & arr\ (\lambda\Delta.\, \lambda x.\, [\![M]\!]) \\
= & \quad (\text{def } [\![-]\!]_\Delta) \\
 & [\![[\lambda x.\, M]]\!]_\Delta
\end{array}
$$

ob_3:

$$
\begin{array}{ll}
 & \llbracket \mathsf{run}\ (\lambda^{\bullet} x.\ \mathsf{let}\ y = P\ \mathsf{in}\ Q) \rrbracket_{\Delta} \\
= & \qquad (\text{def } \llbracket - \rrbracket_{\Delta}) \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ ((\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \llbracket P \rrbracket_{x}) \ggg \llbracket Q \rrbracket_{\langle x,y\rangle}) \\
= & \qquad (\text{Lemma } 3) \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ ((\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} (\mathit{arr}\ (\lambda x.\ \langle\rangle) \ggg \llbracket P \rrbracket_{\langle\rangle})) \ggg \llbracket Q \rrbracket_{\langle x,y\rangle}) \\
= & \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle x, \langle\rangle\rangle) \ggg \mathit{second}\ \llbracket P \rrbracket_{\langle\rangle} \ggg \llbracket Q \rrbracket_{\langle x,y\rangle}) \\
= & \qquad (\rightsquigarrow_{S1}) \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle x, \langle\rangle\rangle) \ggg \mathit{second}\ \llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{force}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle})) \\
= & \qquad (\text{def } \mathit{force}) \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle x, x\rangle) \ggg \\
 & \quad \mathit{first}\ (\mathit{arr}\ (\lambda x.\ \langle x, \langle\rangle\rangle) \ggg \mathit{second}\ \llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle})) \ggg \\
 & \qquad \mathit{arr}\ (\lambda \langle\langle f, (x,a)\rangle, z\rangle.\ f\ \langle x, a\rangle)) \\
= & \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle\langle\rangle, x\rangle) \ggg \\
 & \quad \mathit{first}\ (\llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle})) \ggg \\
 & \qquad \mathit{arr}\ (\lambda \langle\langle f, a\rangle, x\rangle.\ f\ \langle x, a\rangle) \\
= & \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle\langle\rangle, x\rangle) \ggg \\
 & \quad \mathit{first}\ (\llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle})) \ggg \\
 & \qquad \mathit{arr}\ (\lambda \langle\langle f, a\rangle, z\rangle.\ \langle \lambda x.\ f\ \langle x, a\rangle, z\rangle) \ggg \mathit{arr}\ (\lambda \langle f, a\rangle.\ f\ a)) \\
= & \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle\langle\rangle, x\rangle) \ggg \\
 & \quad \mathit{first}\ (\llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle})) \ggg \mathit{first}\ (\mathit{arr}\ (\lambda \langle f, a\rangle.\ \lambda x.\ f\ \langle x, a\rangle)) \ggg \\
 & \qquad \mathit{arr}\ (\lambda \langle f, a\rangle.\ f\ a)) \\
= & \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{arr}\ (\lambda x.\ \langle\langle\rangle, x\rangle) \ggg \\
 & \quad \mathit{first}\ (\llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \ggg \mathit{arr}\ (\lambda \langle f, a\rangle.\ \lambda x.\ f\ \langle x, a\rangle)) \ggg \\
 & \qquad \mathit{arr}\ (\lambda \langle f, a\rangle.\ f\ a)) \\
= & \qquad (\text{def } \mathit{force}) \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \mathit{delay}\ (\mathit{force}\ (\llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \ggg \\
 & \quad \mathit{arr}\ (\lambda \langle f, a\rangle.\ \lambda x.\ f\ \langle x, a\rangle))) \\
= & \qquad (\rightsquigarrow_{S2}) \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \llbracket P \rrbracket_{\langle\rangle} \ggg \mathit{arr}\ (\lambda a.\ \langle\langle\rangle, a\rangle) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \ggg \mathit{arr}\ (\lambda \langle f, a\rangle.\ \lambda x.\ f \langle x, a\rangle) \\
= & \\
 & \mathit{arr}\ (\lambda \Delta.\ \langle\rangle) \ggg \llbracket P \rrbracket_{\langle\rangle} \ggg (\mathit{arr}\ (\lambda y.\ \langle\rangle) \mathbin{\&\&\&} \mathit{arr}\ \mathit{id}) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \ggg \mathit{arr}\ (\lambda \langle f, y\rangle.\ \lambda x.\ f \langle x, y\rangle) \\
= & \qquad (\text{Lemma } 3) \\
 & \llbracket P \rrbracket_{\Delta} \ggg (\mathit{arr}\ (\lambda y.\ \langle\rangle) \mathbin{\&\&\&} \mathit{arr}\ \mathit{id}) \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \ggg \mathit{arr}\ (\lambda \langle f, y\rangle.\ \lambda x.\ f (x, y)) \\
= & \qquad (\text{classic arrow laws}) \\
 & (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \llbracket P \rrbracket_{\Delta}) \ggg \mathit{arr} (\lambda \langle \Delta, y\rangle.\ \langle\langle\rangle, y\rangle \ggg \mathit{first}\ (\mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \ggg \mathit{arr}\ (\lambda \langle f, y\rangle.\ \lambda x.\ f\ \langle x, y\rangle) \\
= & \qquad (\text{classic arrow laws}) \\
 & (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \llbracket P \rrbracket_{\Delta}) \ggg ((\mathit{arr}\ (\lambda \langle \Delta, y\rangle.\ \langle\rangle) \ggg \mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle}) \mathbin{\&\&\&} \mathit{arr}\ (\lambda \langle \Delta, y\rangle.\ y)) \ggg \\
 & \quad \mathit{arr}\ (\lambda \langle f, y\rangle.\ \lambda x.\ f\ \langle x, y\rangle) \\
= & \qquad (\text{classic arrow laws}) \\
 & (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} \llbracket P \rrbracket_{\Delta}) \ggg (\mathit{arr}\ \mathit{id} \mathbin{\&\&\&} (\mathit{arr}\ (\lambda \langle \Delta, y\rangle.\ \langle\rangle) \ggg \mathit{delay}\ \llbracket Q \rrbracket_{\langle x,y\rangle})) \ggg \mathit{arr}\ (\lambda \langle\langle \Delta, y\rangle, f\rangle.\ \lambda x.\ f \langle x, y\rangle) \\
= & \qquad (\text{def } \llbracket - \rrbracket_{\Delta}) \\
 & \llbracket \mathsf{let}\ y = P\ \mathsf{in}\ \mathsf{let}\ f = \mathsf{run}\ (\lambda^{\bullet} \langle x, y\rangle.\ Q)\ \mathsf{in}\ [\lambda x.\ f\ \langle x, y\rangle] \rrbracket_{\Delta}
\end{array}
$$

D.3 Translating classic arrows to arrow calculus and back

$[\![\,[\![M]\!]^{-1}\,]\!] = M$.

Case *delay*:

$$\begin{array}{ll}
 & [\![[\![\mathit{delay}]\!]^{-1}]\!] \\
= & \qquad (\text{def } [\![-]\!]^{-1}) \\
 & [\![\lambda a.\, \lambda^{\bullet} u.\, \mathsf{run}\ a]\!] \\
= & \qquad (\text{def } [\![-]\!]) \\
 & \lambda a.\, \mathit{arr}\ (\lambda u.\, \langle\rangle) \ggg \mathit{delay}\ a \\
= & \qquad (\leadsto_1, \eta^{\rightarrow}) \\
 & \mathit{delay}
\end{array}$$

D.4 Translating arrow calculus to classic arrows and back

$[\![\,[\![M]\!]\,]\!]^{-1} = M$ and $[\![\,[\![P]\!]_{\Delta}\,]\!]^{-1} = \lambda^{\bullet}\Delta.\, P$.

Case $\mathsf{run}\ L$:

$$\begin{array}{ll}
 & [\![[\![\mathsf{run}\ L]\!]_{\Delta}]\!]^{-1} \bullet \Delta \\
= & \qquad (\text{def } [\![-]\!]) \\
 & [\![\mathit{arr}\ (\lambda\Delta.\, \langle\rangle) \ggg \mathit{delay}\ [\![L]\!]]\!]^{-1} \bullet \Delta \\
= & \qquad (\text{def } [\![-]\!]_{\Delta}) \\
 & (\lambda^{\bullet}\Delta.\, \mathsf{let}\ x = [\langle\rangle]\ \mathsf{in}\ (\lambda^{\bullet} u.\, \mathsf{run}\ [\![[\![L]\!]]\!]^{-1}) \bullet x) \bullet \Delta \\
= & \qquad (\text{induction hypothesis}) \\
 & (\lambda^{\bullet}\Delta.\, \mathsf{let}\ x = [\langle\rangle]\ \mathsf{in}\ (\lambda^{\bullet} u.\, \mathsf{run}\ L) \bullet x) \bullet \Delta \\
= & \qquad (\text{left}, \beta^{\leadsto}, \beta^{\leadsto}) \\
 & \mathsf{run}\ L
\end{array}$$