

## Epigram 2

Conor McBride, Peter Morris, Nicolas Oury and [Insert Your Name Here]

November 15, 2007

# Chapter 1

## Introduction

The purpose of this document is to pull together various vague ideas which have been circulating about what Epigram 2 might comprise. It is unlikely to be the second or last such document.

## Chapter 2

# Epigram's Core Type Theory

### 2.1 Bidirectional Syntax

In Epigram 2, we make an explicit separation between the `ExTM` terms, whose types we expect to infer, from the `InTM` terms, whose types we can only hope to check. The crude `LATEX` syntax is like this:

<code>InTM</code>	<code>::=</code>	<code>ExTM</code>	term with inferrable type
		<code>λUID[x] ↦ [x]InTM</code>	functional abstraction
		<code>⟨UID⟩</code>	tag
		<code>[]</code>	empty tuple
		<code>[InTM InTM]</code>	head-and-tail pair
		<code>InTM/InTM</code>	node-and-ford-proof pair
		<code>INT (+InTM)?</code>	element of enumeration
		<code>ιInTM</code>	packed equality proof
		<code>(InTM)</code>	grouping
<code>ExTM</code>	<code>::=</code>	<code>InTM : InTM</code>	cast
		<code>(UID[x] : InTM) → [x]InTM</code>	dependent function space
		<code>(InTM : InTM) = (InTM : InTM)</code>	equation
		<code>CON</code>	type constructor (blue)
		<code>UID(INT)?</code>	variable (with index)
		<code>ExTM PROJ(InTM)</code>	projection
		<code>InTM [InTM : InTM = InTM]</code>	
		<code>OP</code>	functional operator (green)
		<code>⎯InTM : InTM</code>	proof by reflexivity
		<code>λUID[x] : InTM ↦ [x]ExTM</code>	typed abstraction
		<code>(ExTM)</code>	grouping
<code>PROJ(T)</code>	<code>::=</code>	<code>T</code>	application
		<code>•</code>	head
		<code>—</code>	tail
		<code>∨</code>	ex falso quodlibet
		<code>?</code>	unpack equality proof

Of course, we shall be as generous as we can with syntactic sugar. Note that syntactic sugar is *syntactic*: it must be context-free, requiring only adjustment to the parser.

**Remark 1 (Symbolic scoping)** We write `UID[x]` to indicate the binding use of an identifier, which we name `x`, so that we can write `[x]THING` to indicate where

this bound variable is in scope. We can then introduce syntactic categories for more exciting binding forms, e.g. sequences and telescopes

$$\begin{aligned} \text{UIDS}[x] &\supseteq \text{UID}[x] \\ \text{UIDS}[x; \vec{x}] &\supseteq \text{UID}[x] \text{UIDS}[\vec{x}] \\ \text{TEL}[x] &\supseteq \text{UID}[x] : \text{INTM} \\ \text{TEL}[x; \vec{x}] &\supseteq \text{UID}[x] : \text{INTM}; [x] \text{TEL}[\vec{x}] \end{aligned}$$

The key invariant is that the advertised variables must all be bound somewhere in the expanded production.

**Sugar 2 (Non-dependent function space, multiple bindings)** We allow the usual abbreviation for non-dependent function spaces

$$\text{EXTM} \supseteq \text{INTM} \rightarrow \text{INTM} \quad \text{where} \quad S \rightarrow T \implies (\_ : S) \rightarrow T$$

and for function spaces and abstractions of greater arity

$$\begin{aligned} \text{EXTM} &\supseteq (\text{TEL}[\vec{x}]) \rightarrow [\vec{x}] \text{INTM} \\ \text{EXTM} &\supseteq \lambda \text{TEL}[\vec{x}] \mapsto [\vec{x}] \text{EXTM} \\ \text{INTM} &\supseteq \lambda \text{UIDS}[\vec{x}] \mapsto [\vec{x}] \text{INTM} \end{aligned}$$

with the obvious iterative translation.

**Remark 3 (Repetition)** The Epigram 2 convention to handle repeated bindings of the same identifier is to attach a de Bruijn index [?] to each identifier. More precisely,  $x^i$  refers to the thing which was called  $x$  before  $i$  more  $x$ 's were locally bound, and  $x$  is just short for  $x^0$ . So you can play the one-note samba if you like that sort of thing

$$\lambda x x \mapsto x : (x : \star; x : x) \rightarrow x^1$$

The programmer chooses the names: the sum of the resultant indices may measure the prudence of those choices, but there is no danger of ambiguity or shadowing.