Epigram 2 Design Issues

A Doubtless Contentious Sequence of Names

May 31, 2006

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 first or last such document.

2 Epigram's Core Type Theory

A teacup in which lurks one of Type Theory's more mediocre meteorological misfortunes is circumscribed 'Typed λ-abstractions or untyped λ-abstractions?'. We say 'Yes.'. That is to say, we so favour an explicitly **bidirectional** typechecking, separating the processes of checking a type prescription and inferring a type description, that we even separate the syntactic categories of terms (respectively $\textsc{Term}^{\uparrow}$ and $\textsc{Term}^{\downarrow}$) to which they apply. Here is the shape of things to come:

$$
\begin{array}{rcl}
\textsc{Term}^{\uparrow} & ::= & \textsc{Term}^{\downarrow} : \textsc{Term}^{\downarrow} \\
& | & \textsc{Var} \\
& | & \textsc{Term}^{\uparrow}\ \textsc{Elim}(\textsc{Term}^{\downarrow}) \\
& | & \textsc{Const} \\
& | & \Pi \textsc{Var} : \textsc{Term}^{\downarrow} \Rightarrow \textsc{Term}^{\downarrow} \\
& | & \lambda \textsc{Var} : \textsc{Term}^{\downarrow} \Rightarrow \textsc{Term}^{\uparrow}
\end{array}
\qquad
\begin{array}{rcl}
\textsc{Term}^{\downarrow} & ::= & \textsc{Term}^{\uparrow} \\
& | & \lambda \textsc{Var} \Rightarrow \textsc{Term}^{\downarrow} \\
\textsc{Elim}(X) & ::= & X \\
\textsc{Const} & ::= & \star
\end{array}
$$

Some observations:

1. If you can infer a type, you can certainly check one. If you need to infer a type from a term which is only checkable, you had better provide the type explicitly.
2. Suitably labelled things live in $\textsc{Term}^{\uparrow}$: we know what Π and $\star$ make, and we know what their pieces should be. Just like type annotations, suitably informative pieces of syntax effect the 'good' change of direction.
3. Elimination behaviour is postfix, in the traditional style of application and projection. We only have application here so far. The definition is parametric because we'll be recycling it in a minute. In general, we infer the type of the target and this tells us how to type the pieces of its eliminator. That is, we don't infer types of things from the way they are used, but we do check that things are used properly.

Brick my shorts! There's a vaguely sensible way to give a small-step semantics to this syntax. We don't need one, but here it is anyway. The only way the untyped canonical forms can get into a position to be used is to have a type annotation, so we can produce the labelled version.

$$
\begin{array}{rcl}
(\lambda x : S \Rightarrow t)\ s & \rightsquigarrow_{\beta} & t[s : S] \\
\lambda x \Rightarrow t : \Pi x : S \Rightarrow T & \rightsquigarrow_{\tau} & \lambda x : S \Rightarrow t
\end{array}
$$

2.1 Evaluation Semantics

What we really use is an evaluation semantics. What's a value?

$$\begin{array}{rcl} \textsc{Val} & ::= & \textsc{Var Spine} \\ & | & \textsc{Const} \\ & | & \Pi \textsc{Var} : \textsc{Val} \Rightarrow \textsc{Val} \\ & | & \lambda \textsc{Var} \Rightarrow \textsc{Val} \\ \textsc{Spine} & ::= & \Big(\textsc{Elim}(\textsc{Val}) \Big)^* \end{array}$$

As you may notice, every VAL corresponds to a $\textsc{Term}^{\downarrow}$, in particular, one with no explicit typings or labelled λs. Correspondingly, we shall let $\textsc{Val} \subset \textsc{Term}^{\downarrow}$ henceforth. We define two operations mutually, **elimination** and **instantiation**. Elimination is denoted by juxtaposing a VAL with an ELIM(VAL).

$$\begin{array}{lcl} x\,\vec{e}\;\;e & \Longrightarrow & x\;\;\vec{e}\,e \\ (\lambda x \Rightarrow v)\,u & \Longrightarrow & v[u] \end{array}$$

Elimination readily lifts to a sequence of eliminators in the obvious left-nested way. Instantiation is denoted by postfixing $-[v_n; \ldots; v_0]$ an **environment** of values to instantiate the innermost free variables. We write γ to stand for such an environment and γx to indicate the value given for x. Environments which are 'too short' pad out, mapping outer free variables to themselves. Instantiation lifts functorially to sequences of eliminators.

$$\begin{array}{lcl} (x\,\vec{e})[\gamma] & \Longrightarrow & \gamma x\,\vec{e}[\gamma] \\ c[\gamma] & \Longrightarrow & c \\ (\Pi x : u \Rightarrow v)[\gamma] & \Longrightarrow & \Pi x : u[\gamma] \Rightarrow v[\gamma; x] \\ (\lambda x \Rightarrow v)[\gamma] & \Longrightarrow & \lambda x \Rightarrow v[\gamma; x] \end{array}$$

Note that the only interesting case is the variable case, which is where instantiation might cause some elimination. The remaining cases are reassuringly structural. This is a general pattern: in order to give semantics to an extended theory, we shall need only to extend elimination explicitly.

We may now say what evaluation $t^\dagger$ is for terms (of either orientation) and lifting functorially to eliminators. It's just the obvious structural thing, discarding unnecssary annotations and replacing textual elimination with semantic elimination. Of course, $-^\dagger$ is the identity on values.

$$\begin{array}{lcl} \textsc{Term}^{\uparrow} & & \textsc{Val} \\ (t : T)^\dagger & \Longrightarrow & t^\dagger \\ x^\dagger & \Longrightarrow & x \\ (t\,e)^\dagger & \Longrightarrow & t^\dagger\,e^\dagger \\ c^\dagger & \Longrightarrow & c \\ (\Pi x : S \Rightarrow T)^\dagger & \Longrightarrow & \Pi x : S^\dagger \Rightarrow T^\dagger \\ (\lambda x : S \Rightarrow t)^\dagger & \Longrightarrow & \lambda x \Rightarrow t^\dagger \end{array} \qquad \begin{array}{lcl} \textsc{Term}^{\downarrow} & & \textsc{Val} \\ t^\dagger & \Longrightarrow & t^\dagger \\ (\lambda x \Rightarrow t)^\dagger & \Longrightarrow & \lambda x \Rightarrow t^\dagger \end{array}$$

2.2 Typing Rules

We aim to give an algorithmic presentation of typing, which is not to say that we shall show you the code, but rather that we shall take a little more care than is usual in the presentation of judgment forms and rules. Each judgment corresponds to a monadic computation in some monad supporting a context (assigning a type value to each variable) and a notion of failure (here represented by the absence of an applicable rule). We shall thus be careful to distinguish the inputs from the outputs in each jugment form and to write the latter after the former. The traditional box is thus split in two!

type inference $\boxed{\textsc{Ctxt} \vdash \textsc{Term}^{\uparrow} \in \mid \textsc{Val}}$
Compute the type which describes a $\textsc{Term}^{\uparrow}$.

type checking $\boxed{\textsc{Ctxt} \vdash \textsc{Val} \ni \textsc{Term}^{\downarrow} \mid \quad}$
Decide whether a given $\textsc{Term}^{\downarrow}$ has the prescribed type.

value equality $\boxed{\textsc{Ctxt} \vdash \textsc{Val} \ni \textsc{Val} \equiv \textsc{Val} \mid \quad}$
Check the equality of values in a given type.

spine equality $\boxed{\vdash \textsc{Var Spine} \equiv \textsc{Var Spine} \in \mid \textsc{Val}}$
Check the equality of values in a given type.

context lookup $\boxed{\dashv \textsc{Var} : \mid \textsc{Val}}$
Look up the type of a variable in the ambient context.

context extension $\boxed{\vdash \textsc{Ctxt} \vdash \mid \quad}$
Check the validity of a context extension, relative to the ambient context.

The notion of $\textsc{Ctxt}$ used here has syntax

$$\textsc{Ctxt} \quad ::= \quad \Big(\textsc{Var} : \textsc{Val} \Big)^{;*}$$

and it refers to a **local** extension of the ambient context, plumbed via the monad.

Let's try to identify some of the conditions which rule systems like these should satisfy.

clockwise flow Schematic variables appearing in rules may only be schematically bound in the inputs of conclusions or the outputs of premises. Scope flows clockwise from the inputs of the conclusions, left-to-right through the premises, then down to the outputs of the conclusions. Clockwise flow prohibits non-rules such as

$$(\times) \qquad \frac{\vdash \star \ni S \quad x : S^{\dagger} \vdash t \in T}{\vdash \lambda x \Rightarrow t \in \Pi x : S \Rightarrow T} \qquad \frac{\vdash \Pi x : S \Rightarrow T \ni f \quad \vdash S \ni s}{\vdash f\ s \in T[s^{\dagger}]}$$

which just make stuff up out of nothing. Similarly, the 'equality reflection' rule of Extensional Type Theory is excluded by such considerations.

relative scoping Schematic variables must be used in a context compatible with their schematic binding. You can check this by counting object bindings, introduced by a local context, a binding operator or (anonymously) by instantiation. The application rule is a good example. Non-examples include

$$(\times) \qquad \frac{}{\vdash \Pi x : S \Rightarrow T \ni \lambda x \Rightarrow f\ x \equiv f}$$

no search For given inputs to a conclusion, at most one rule should be applicable.

No explicit mention of context extension jugments occurs in the rules of the theory. Here are the rules

$$\frac{}{\vdash \vdash} \qquad \frac{\vdash \Delta \vdash \quad \Delta \vdash \star \ni S}{\vdash \Delta; x : S \vdash}$$

Where do these rules show up? Even to consider, let alone derive a judgment, one must fulfil its contract.

- $\Delta \vdash t \in T$
  requires $\vdash \Delta \vdash$ and either T is $\star$ or $\Delta \vdash \star \ni T$

$$\boxed{\textsc{Ctxt} \vdash \textsc{Term}^{\uparrow} \in \mid \textsc{Val}}$$

$$\frac{\vdash \star \ni T \quad \vdash T^{\dagger} \ni t}{\vdash t : T \in T^{\dagger}} \qquad \frac{\dashv x : S}{\vdash x \in S} \qquad \frac{}{\vdash \star \in \star}$$

$$\frac{\vdash \star \ni S \quad x : S^{\dagger} \vdash \star \ni T}{\vdash \Pi x : S \Rightarrow T \in \star} \qquad \frac{\vdash \star \ni S \quad x : S^{\dagger} \vdash t \in T}{\vdash \lambda x : S \Rightarrow t \in \Pi x : S^{\dagger} \Rightarrow T}$$

$$\frac{\vdash f \in \Pi x : S \Rightarrow T \quad \vdash S \ni s}{\vdash f\ s \in T[s^{\dagger}]}$$

Figure 1: type inference

$$\boxed{\textsc{Ctxt} \vdash \textsc{Val} \ni \textsc{Term}^{\downarrow} \mid \ }$$

$$\frac{\vdash s \in S \quad \vdash \star \ni S \equiv T}{\vdash T \ni s} \qquad \frac{x : S \vdash T \ni t}{\vdash \Pi x : S \Rightarrow T \ni \lambda x \Rightarrow t}$$

Figure 2: type checking

$$\boxed{\textsc{Ctxt} \vdash \textsc{Val} \ni \textsc{Val} \equiv \textsc{Val} \mid \ }$$

$$\frac{\vdash x_0\ \vec{e}_0 \equiv x_1\ \vec{e}_1 \in T}{\vdash X\ \vec{e} \ni x_0\ \vec{e}_0 \equiv x_1\ \vec{e}_1}$$

$$\frac{}{\vdash \star \ni \star \equiv \star} \qquad \frac{\vdash \star \ni S_0 \equiv S_1 \quad x : S_0 \vdash \star \ni T_0 \equiv T_1}{\vdash \star \ni \Pi x : S_0 \Rightarrow T_0 \equiv \Pi x : S_1 \Rightarrow T_1}$$

$$\frac{x : S \vdash T \ni f_0\ x \equiv f_1\ x}{\vdash \Pi x : S \Rightarrow T \ni f_0 \equiv f_1}$$

Figure 3: value equality

$$\boxed{\vdash \textsc{Var Spine} \equiv \textsc{Var Spine} \in \mid \textsc{Val}}$$

$$\frac{\dashv x : S}{\vdash x \equiv x \in S} \qquad \frac{\vdash x_0\ \vec{e}_0 \equiv x_1\ \vec{e}_1 \in \Pi x : S \Rightarrow T \quad \vdash S \ni s_0 \equiv s_1}{\vdash x_0\ \vec{e}_0\ s_0 \equiv x_1\ \vec{e}_1\ s_1 \in T[s_0]}$$

Figure 4: spine equality

- $\Delta \vdash T \ni t$
  requires $\vdash \Delta \vdash$ and either T is $\star$ or $\Delta \vdash \star \ni T$
- $\Delta \vdash T \ni t_0 \equiv t_1$
  requires $\Delta \vdash T \ni t_0$ and $\Delta \vdash T \ni t_1$
- $\vdash x_0\ \vec{e}_0 \equiv x_1\ \vec{e}_1 \in T$
  requires $\dashv x_0 : S_0$ and $\dashv x_1 : S_1$ for some S_0, S_1
  guarantees $\vdash x_0\ \vec{e}_0 \in T$ and $\vdash x_1\ \vec{e}_1 \in T$
- $\dashv x : S$
  guarantees $\vdash \star \ni S$

To check that a rule satisfies its contracts, assume the requirements of the conclusion and check the requirements of the premises left-to-right, then deliver the guarantees of the conclusion.