Staging as a Mechanism for Algorithm Derivation

Nicolas Feltman

August 29, 2013

1 Introduction

This document presents some preliminary work on a staged language, Stag, useful for deriving complicated algorithms. Section 2 covers the work on the language so far. We then compare this to some relevent literature in Section 3. In Section 4, we give a proposal for how to move forward with the language.

2 Current Work

2.1 Terms, Types, and Stages

A grammar for Stag is shown in Figure 1. It is currently a zeroth-order language, having no mechanisms to introduce functions. As a stopgap, we have included a notion of "external functions" which can be eliminated through application, but have no introduction forms. These functional definiciencies aside, the language supports both products and sums.

Types in the language are split into two parts: the *pretype* and the *stage*. The pretype is just the standard algebraic data type that we all know and love; it indicates *what* a term will reduce to. The stage, currently at the coarse granularity of $\mathbb{1}$ and $\mathbb{2}$, is an indication of *when* a term will be fully computed.

The rules relating types and terms are given in Figure 2. The structure of the core typing judgement is entirely standard. The nullary terms (currently just the unit value) are all explicitly annotated with a stage. The non-nullary terms essentially preserve the stage of their inputs, except for `pause`, which transtions an expression from $\mathbb{1}$ to $\mathbb{2}$. From this, it is apparent that a stage $\mathbb{1}$ term cannot depend on any stage $\mathbb{2}$ terms. This property can be justified by the notion that the program gets "more reduced" as time goes on, and that a term can be no more reduced than those it depends on.

Stag has one additional peculiar feature. The predicate of a case statement can be stage $\mathbb{2}$ while the branches have subterms with stage $\mathbb{1}$. This implies that some computation in a branch must be performed even before we know which branch to take. This feature, called *speculation*, will be explored more later.

Figure 1: Stag Grammar

$$
\begin{aligned}
\langle stage\rangle &= \mathbb{1} \mid \mathbb{2} \\
\langle ptype\rangle &= \text{unit} \mid \langle ptype\rangle \times \langle ptype\rangle \mid \langle ptype\rangle + \langle ptype\rangle \\
\langle type\rangle &= \langle ptype\rangle \ @\ \langle stage\rangle \\
\langle exp\rangle &= \langle func\rangle\ \langle exp\rangle \\
&\mid\ ()_{\langle stage\rangle} \\
&\mid\ (\langle exp\rangle, \langle exp\rangle) \\
&\mid\ \pi_1\ \langle exp\rangle \\
&\mid\ \pi_2\ \langle exp\rangle \\
&\mid\ \iota_1\ \langle exp\rangle \\
&\mid\ \iota_2\ \langle exp\rangle \\
&\mid\ \texttt{let}\ \langle var\rangle = \langle exp\rangle\ \texttt{in}\ \langle exp\rangle \\
&\mid\ \langle var\rangle \\
&\mid\ \texttt{case}\ \langle exp\rangle\ \texttt{of}\ \langle var\rangle.\langle exp\rangle\ \text{`}|\text{'}\ \langle var\rangle.\langle exp\rangle \\
&\mid\ \texttt{pause}\ \langle exp\rangle \\
\langle cont\rangle &= \cdot \\
&\mid\ \langle cont\rangle, \langle var\rangle : \langle type\rangle
\end{aligned}
$$

2.2 Evaluation

This section is not complete. Were it complete, it would contain a definition of values (which we predict to be unchanged by the staging system), and a big-step sematics relating terms to values. The big-step evaluation should be indexed by the stage at which it completes. That is, we have both $\Downarrow_{\mathbb{1}}$ and $\Downarrow_{\mathbb{2}}$. Also, the bigstep semantics will reflect that speculation occurs down both branches of a case as part of $\Downarrow_{\mathbb{1}}$ reduction.

2.3 Stage Splitting

The core operation of interest is the process of "stage splitting," wherein a term with stage $\mathbb{2}$ is converted to a precomputed part, and a residual that depends on that precomputed part. Specifically, we introduce a judgement "$\Gamma \vdash e \rightsquigarrow [p, x.r]$", which can be read "under the context Γ, e stage-splits into a precomputation p, and a residual r which is open on x". The idea is that p contains the parts of e that are stage $\mathbb{1}$, r contains the parts of e that are stage $\mathbb{2}$, and the reduced value of p is bound to x when evaluating r.

Note that the precomputation and residual are not actually terms in Stag. They are actually terms in a simpler language, called Doe, which is essentially Stag without the staging constructs. The grammar for Doe is shown in Figure 3. The typing rules are not shown, but can be guessed.

The full rules for splitting are shown in Figure 4.

Figure 2: Typing Rules

$$\frac{.}{\Gamma \vdash ()_\sigma : \text{unit} \mathbin{@} \sigma} \tag{1}$$

$$\frac{\Gamma \vdash e : A \mathbin{@} \sigma \quad f : A \to B}{\Gamma \vdash f\ e : B \mathbin{@} \sigma} \tag{2}$$

$$\frac{\Gamma \vdash e : A \times B \mathbin{@} \sigma}{\Gamma \vdash \pi_1\ e : A \mathbin{@} \sigma} \tag{3}$$

$$\frac{\Gamma \vdash e : A \times B \mathbin{@} \sigma}{\Gamma \vdash \pi_2\ e : B \mathbin{@} \sigma} \tag{4}$$

$$\frac{\Gamma \vdash e : A \mathbin{@} \sigma}{\Gamma \vdash \iota_1\ e : A + B \mathbin{@} \sigma} \tag{5}$$

$$\frac{\Gamma \vdash e : B \mathbin{@} \sigma}{\Gamma \vdash \iota_1\ e : A + B \mathbin{@} \sigma} \tag{6}$$

$$\frac{\Gamma \vdash e_1 : A \mathbin{@} \sigma \quad \Gamma \vdash e_2 : B \mathbin{@} \sigma}{\Gamma \vdash (e_1, e_2) : A \times B \mathbin{@} \sigma} \tag{7}$$

$$\frac{\Gamma \vdash e_1 : A \mathbin{@} \mathbb{1} \quad \Gamma, x : A \mathbin{@} \mathbb{1} \vdash e_2 : \mathbb{1}}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : B \mathbin{@} \mathbb{1}} \tag{8}$$

$$\frac{\Gamma \vdash e_1 : A \mathbin{@} \mathbb{1} \quad \Gamma, x : A \mathbin{@} \mathbb{1} \vdash e_2 : \mathbb{2}}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : B \mathbin{@} \mathbb{2}} \tag{9}$$

$$\frac{\Gamma \vdash e_1 : A \mathbin{@} \mathbb{2} \quad \Gamma, x : A \mathbin{@} \mathbb{2} \vdash e_2 : \mathbb{2}}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : B \mathbin{@} \mathbb{2}} \tag{10}$$

$$\frac{\Gamma \vdash e_1 : A + B \mathbin{@} \mathbb{1} \quad \Gamma, x_2 : A \mathbin{@} \mathbb{1} \vdash e_2 : C \mathbin{@} \mathbb{1} \quad \Gamma, x_3 : B \mathbin{@} \mathbb{1} \vdash e_3 : C \mathbin{@} \mathbb{1}}{\Gamma \vdash \texttt{case}\ e_1\ \texttt{of}\ x_2.e_2 \mid x_3.e_3 : C \mathbin{@} \mathbb{1}} \tag{11}$$

$$\frac{\Gamma \vdash e_1 : A + B \mathbin{@} \mathbb{1} \quad \Gamma, x_2 : A \mathbin{@} \mathbb{1} \vdash e_2 : C \mathbin{@} \mathbb{2} \quad \Gamma, x_3 : B \mathbin{@} \mathbb{1} \vdash e_3 : C \mathbin{@} \mathbb{2}}{\Gamma \vdash \texttt{case}\ e_1\ \texttt{of}\ x_2.e_2 \mid x_3.e_3 : C \mathbin{@} \mathbb{2}} \tag{12}$$

$$\frac{\Gamma \vdash e_1 : A + B \mathbin{@} \mathbb{2} \quad \Gamma, x_2 : A \mathbin{@} \mathbb{2} \vdash e_2 : C \mathbin{@} \mathbb{2} \quad \Gamma, x_3 : B \mathbin{@} \mathbb{2} \vdash e_3 : C \mathbin{@} \mathbb{2}}{\Gamma \vdash \texttt{case}\ e_1\ \texttt{of}\ x_2.e_2 \mid x_3.e_3 : C \mathbin{@} \mathbb{2}} \tag{13}$$

$$\frac{x : A \mathbin{@} \sigma \in \Gamma}{\Gamma \vdash x : A \mathbin{@} \sigma} \tag{14}$$

$$\frac{\Gamma \vdash e : A \mathbin{@} \mathbb{1}}{\Gamma \vdash \texttt{pause}\ e : A \mathbin{@} \mathbb{2}} \tag{15}$$

Figure 3: STAG Grammar

$$
\begin{aligned}
\langle stage \rangle &= \mathbb{1} \mid \mathbb{2} \\
\langle ptype \rangle &= \text{unit} \mid \langle ptype \rangle \times \langle ptype \rangle \mid \langle ptype \rangle + \langle ptype \rangle \\
\langle type \rangle &= \langle ptype \rangle \ @ \ \langle stage \rangle \\
\langle exp \rangle &= \langle func \rangle \ \langle exp \rangle \\
&\mid \ ()_{\langle stage \rangle} \\
&\mid \ (\langle exp \rangle, \langle exp \rangle) \\
&\mid \ \pi_1 \ \langle exp \rangle \\
&\mid \ \pi_2 \ \langle exp \rangle \\
&\mid \ \iota_1 \ \langle exp \rangle \\
&\mid \ \iota_2 \ \langle exp \rangle \\
&\mid \ \texttt{let} \ \langle var \rangle = \langle exp \rangle \ \texttt{in} \ \langle exp \rangle \\
&\mid \ \langle var \rangle \\
&\mid \ \texttt{case} \ \langle exp \rangle \ \texttt{of} \ \langle var \rangle.\langle exp \rangle \ `|\text{'} \ \langle var \rangle.\langle exp \rangle \\
&\mid \ \texttt{pause} \ \langle exp \rangle \\
\langle cont \rangle &= \cdot \\
&\mid \ \langle cont \rangle, \langle var \rangle : \langle type \rangle
\end{aligned}
$$

2.4 Implementation

I have an implementation of stage-splitting in SML. Currently, the code only keeps track of stage, and not the pre-type.

[Add results.]

3 Related Work

4 Paths For Extension

4.1 Functions

This language will obviously need functions (aside from the currect “external functions” stopgap). I’m confident that we could add a second-class function system on top of this, and everything would go well. That said, it would be much more interesting to add true first-class functions. Either way, a required feature of these functions would be the ability to cross stage boundaries. For instance, we’d probably see functions types like “$(A \ @ \ \mathbb{1} \to B \ @ \ \mathbb{2}) \ @ \ \mathbb{1}$”.

Figure 4: Term Splitting

$$\frac{\cdot}{\Gamma \vdash ()_{\mathbb{2}} \rightsquigarrow [(), \_.()]} \tag{16}$$

$$\frac{\Gamma(x) = \mathbb{2}}{\Gamma \vdash x \rightsquigarrow [(), \_.x]} \tag{17}$$

$$\frac{\Gamma \vdash e \rightsquigarrow [p, x.r]}{\Gamma \vdash f\ e \rightsquigarrow [p, x.f\ r]} \tag{18}$$

$$\frac{\Gamma \vdash e \rightsquigarrow [p, x.r]}{\Gamma \vdash \pi_i\ e \rightsquigarrow [p, x.\pi_i\ r]} \tag{19}$$

$$\frac{\Gamma \vdash e \rightsquigarrow [p, x.r]}{\Gamma \vdash \iota_i\ e \rightsquigarrow [p, x.\iota_i\ r]} \tag{20}$$

$$\frac{\Gamma \vdash e_1 \rightsquigarrow [p_1, x_1, r_1] \quad \Gamma \vdash e_2 \rightsquigarrow [p_2, x_2, r_2]}{\Gamma \vdash (e_1, e_2) \rightsquigarrow [(p_1, p_2), l.(\texttt{let}\ x_1 = \pi_1\ l\ \texttt{in}\ r_1, \texttt{let}\ x_2 = \pi_2\ l\ \texttt{in}\ r_2)]} \tag{21}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow e_1' \quad \Gamma, x : \mathbb{1} \vdash e_2 \rightsquigarrow [p_2, y_2.r_2]}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \rightsquigarrow [\texttt{let}\ x = e_1'\ \texttt{in}\ p_2, y_2.r_2]} \tag{22}$$

$$\frac{\Gamma \vdash e_1 \rightsquigarrow [p_1, y_1.r_1] \quad \Gamma, x : \sigma_1 \vdash e_2 \rightsquigarrow [p_2, y_2.r_2]}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \rightsquigarrow [(p_1, p_2), l.\texttt{let}\ x = (\texttt{let}\ y_1 = \pi_1\ l\ \texttt{in}\ r_1)\ \texttt{in}\ \texttt{let}\ y_2 = \pi_2\ l\ \texttt{in}\ r_2]} \tag{23}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow e_1' \quad \Gamma, x_2 : \mathbb{1} \vdash e_2 \rightsquigarrow [p_2, y_2, r_2] \quad \Gamma, x_3 : \mathbb{1} \vdash e_3 \rightsquigarrow [p_3, y_3, r_3]}{\Gamma \vdash \texttt{case}\ e_1\ \texttt{of}\ x_2.e_2 \mid x_3.e_3 \rightsquigarrow \left[\begin{pmatrix} \texttt{case}\ e_1'\ \texttt{of} \\ x_2.\iota_1\ p_2 \\ \mid x_3.\iota_2\ p_3 \end{pmatrix}, l.\texttt{case}\ l\ \texttt{of}\ y_2.r_2 \mid y_3.r_3\right]} \tag{24}$$

$$\frac{\Gamma \vdash e_1 \rightsquigarrow [p_1, y_1, r_1] \quad \Gamma, x_2 : \mathbb{2} \vdash e_2 \rightsquigarrow [p_2, y_2, r_2] \quad \Gamma, x_3 : \mathbb{2} \vdash e_3 \rightsquigarrow [p_3, y_3, r_3]}{\Gamma \vdash \texttt{case}\ e_1\ \texttt{of}\ x_2.e_2 \mid x_3.e_3 \rightsquigarrow \left[(p_1, (p_2, p_3)), l. \begin{pmatrix} \texttt{case}\ (\texttt{let}\ y_1 = \pi_1 l\ \texttt{in}\ r_1)\ \texttt{of} \\ x_2.\texttt{let}\ y_2 = \pi_1\pi_2 l\ \texttt{in}\ r_2 \\ \mid x_3.\texttt{let}\ y_3 = \pi_2\pi_2 l\ \texttt{in}\ r_3 \end{pmatrix}\right]} \tag{25}$$

4.2 Split-Phase Sums and Products

With functions added, we'd certainly want to support functions with multiple arguments at different phase. The type of this might look like "$(A \text{ @ } \mathbb{1} \times B \text{ @ } \mathbb{2} \to C \text{ @ } \mathbb{2})$". But this is unsettling, since under our current rules the domain "$A \text{ @ } \mathbb{1} \times B \text{ @ } \mathbb{2}$" isn't even allowed! There are two possible solutions here. The more hackish route would be to define the domain of a function as a list of arguments, each with its own phase. The principled route would be to give meaning to all split-phase products, wherever they appear. If we chose to support first-class functions and true split-phase products, then a good measure of success would be the definability of the `curry` and `uncurry` functions for any phase combination. Hopefully, split-phase sums wouldn't be much harder that such products.

4.3 N-Ary Staging

Another goal, which seems orthogonal to those above, would be to allow any number of stages (rather than the current 2) such that the `pause` operator increments the stage of its argument by 1. In this case, the current splitting operation would become part of the inductive step of a multi-stage split.

4.4 Stage Polymorphism

Finally, it seems that we'd want the ability to abstract over stage. In the 2-level language, it probably doesn't make sense to define both "$\mathit{fsum1} : \texttt{float} \text{ @ } \mathbb{1} \times \texttt{float} \text{ @ } \mathbb{1} \to \texttt{float} \text{ @ } \mathbb{1}$" and "$\mathit{fsum2} : \texttt{float} \text{ @ } \mathbb{2} \times \texttt{float} \text{ @ } \mathbb{2} \to \texttt{float} \text{ @ } \mathbb{2}$". Perhaps's we'd want something like "$\mathit{fsum} : \forall n.\texttt{float} \text{ @ } n \times \texttt{float} \text{ @ } n \to \texttt{float} \text{ @ } n$". It's also possible that we really want just the *fsum1* definition, and function application will "lift" up to later stages as appropriate.