General Stage Untangling with Applications to Algorithm Derivation

Nicolas Feltman et al.

February 7, 2014

1 Introduction

Many computational tasks are premised on the notion of multiple domains, places and times at which computation occurs. Yet programming languages traditionally view their execution environments as monolithic, specifying the What and the How, but not the When nor the Where. As always, where the language falls short, a programmer instead must bear the burden. Inevitably, this comes at the cost of expressivity, safety, or both. In this work, we present a language for specifying certain multi-domain computations, sacrificing neither the expresivity nor the safety that programmers have come to expect in single-domain languages.

Consider the programming of a pipeline system. Generally in such a system, data comes in to one processor, is transformed to an intermediate result, and then fed to later processors, possibly with some buffering, duplication, and reording in between. We can view an algorithm for such as system as one computation, split accross two spatial domains (processors). For a concrete example, we draw inspiration from graphics systems. Lets say that we want to map a function f : PixelCoord $\rightarrow$ Color over every pixel on a 1024x768 screen. Due to geometric coherence inherent in the problem, it's often advantageous to compute some information at the level of coarse 32x32 tiles (e.g. a list of the scene primitives which interact with that part of the screen) whose results can be used to accelerate the final per-pixel calculation. In adition to yielding algorithmic benefits, this decomposition strategy maps well to parallel hardware.

Often when programming high-performance systems, a programmer will wish to specialize a bivariate function $f : \alpha \times \beta \rightarrow \gamma$ to one of its inputs, $x : \alpha$, while leaving the other, $y : \beta$, for later. This need may arise because x varies at a higher frequency than y, and we would wish to avoid repeated calculation. It may also be the case that x is available at a time when computational resources are less expensive than when y becomes available. Here we say that the calculation is split accross two temporal domains. Pulling another example from graphics, we consider a raytracer wherein x : `Geom` is a specification of the scene geometry (perhaps a list of triangles), y : `Ray` is a ray cast from some point in the scene, and f computes the first intersection of the ray and the geometry. In modern raytracers, we may cast millions of rays without varying the scene geometry, so it could be advantageous to precompute as much of the algorithm as possible based on only scene geometry.

In addition to the partitioning of a calculation across multiple domains, the examples above exhibit one further similarity: communication between domains is one-directional. In the spatial case, information always flows from one processor to the next, but never backwards. In the temporal case too, information made available at the second time cannot be observed earlier (indeed this seems to be a defining feature of time, generally). For this reason, we refer to the domains as stages. This property, that information flow is acyclic, informs the rest of this paper.

[Talk about how languages without inherent support for domains must instead tackle the problem by writing various passes/shaders. This is bad because it's not compositional. For instance, I cannot write a function that talks about multiple stages at once.]

Figure 1: $L^{\mathbb{1}\mathbb{2}}$ Syntax

$$
\begin{array}{l}
\langle \mathbb{1}\text{-}type \rangle ::= \text{unit} \mid \text{int} \mid \text{bool} \\
\qquad \mid \langle \mathbb{1}\text{-}type \rangle \times \langle \mathbb{1}\text{-}type \rangle \\
\qquad \mid \bigcirc \langle \mathbb{2}\text{-}type \rangle \\
\langle \mathbb{1}\text{-}exp \rangle ::= () \mid \langle int \rangle \mid \texttt{true} \mid \texttt{false} \\
\qquad \mid \texttt{let } \langle var \rangle = \langle \mathbb{1}\text{-}exp \rangle \texttt{ in } \langle \mathbb{1}\text{-}exp \rangle \\
\qquad \mid \langle var \rangle \\
\qquad \mid (\langle \mathbb{1}\text{-}exp \rangle, \langle \mathbb{1}\text{-}exp \rangle) \\
\qquad \mid \pi_1 \langle \mathbb{1}\text{-}exp \rangle \mid \pi_2 \langle \mathbb{1}\text{-}exp \rangle \\
\qquad \mid \texttt{if } \langle \mathbb{1}\text{-}exp \rangle \texttt{ then } \langle \mathbb{1}\text{-}exp \rangle \texttt{ else } \langle \mathbb{1}\text{-}exp \rangle \\
\qquad \mid \texttt{next } \langle \mathbb{2}\text{-}exp \rangle \\
\langle \mathbb{1}\text{-}val \rangle ::= () \mid \langle int \rangle \mid \textbf{true} \mid \textbf{false} \\
\qquad \mid (\langle \mathbb{1}\text{-}val \rangle, \langle \mathbb{1}\text{-}val \rangle) \\
\qquad \mid \textbf{next } \langle \mathbb{2}\text{-}val \rangle \\
\langle cont \rangle ::= \text{empty} \\
\qquad \mid \langle cont \rangle, \langle var \rangle : \langle \mathbb{1}\text{-}type \rangle^{\mathbb{1}} \\
\qquad \mid \langle cont \rangle, \langle var \rangle : \langle \mathbb{2}\text{-}type \rangle^{\mathbb{2}}
\end{array}
$$

$$
\begin{array}{l}
\langle \mathbb{2}\text{-}type \rangle ::= \text{unit} \mid \text{int} \mid \text{bool} \\
\qquad \mid \langle \mathbb{2}\text{-}type \rangle \times \langle \mathbb{2}\text{-}type \rangle \\
\\
\langle \mathbb{2}\text{-}exp \rangle ::= () \mid \langle int \rangle \mid \texttt{true} \mid \texttt{false} \\
\qquad \mid \texttt{let } \langle var \rangle = \langle \mathbb{2}\text{-}exp \rangle \texttt{ in } \langle \mathbb{2}\text{-}exp \rangle \\
\qquad \mid \langle var \rangle \\
\qquad \mid (\langle \mathbb{2}\text{-}exp \rangle, \langle \mathbb{2}\text{-}exp \rangle) \\
\qquad \mid \pi_1 \langle \mathbb{2}\text{-}exp \rangle \mid \pi_2 \langle \mathbb{2}\text{-}exp \rangle \\
\qquad \mid \texttt{if } \langle \mathbb{2}\text{-}exp \rangle \texttt{ then } \langle \mathbb{2}\text{-}exp \rangle \texttt{ else } \langle \mathbb{2}\text{-}exp \rangle \\
\qquad \mid \texttt{prev } \langle \mathbb{1}\text{-}exp \rangle \\
\langle \mathbb{2}\text{-}val \rangle ::= () \mid \langle int \rangle \mid \textbf{true} \mid \textbf{false} \\
\qquad \mid (\langle \mathbb{2}\text{-}val \rangle, \langle \mathbb{2}\text{-}val \rangle)
\end{array}
$$

[Because of the acyclic nature of the problem, we can take algorithms specified in our multi-domain language and tease them apart, essentially recovering the old way of doing things. This is called stage untangling. It gets us all the safety and expression without messing up the rest of the compiler.]

[Talk about standard partial evaluation and pass separation, then place our work in the second, modulo binding time analysis.]

2 A Simple Two-Stage Language

We'll start by analyzing a simple two-stage language called $L^{\mathbb{1}\mathbb{2}}$. We say it is simple in that it supports boolean, integer, and product types, as well as let bindings, if expressions, and various primitive operations. Notably, we leave out functions, fixed point operators, and full sum types.

The grammar for $L^{\mathbb{1}\mathbb{2}}$is given in Figure 1

2.1 Static Semantics

Introduce the type system. We got it from [1].

2.2 Erasure Dynamic Semantics

Check Figure 3

2.3 Splitting

Introduce L, in figure 4. We're gonna split into two terms of that.

2.4 Splitting Theorems

1. It always pops out a well typed term.

Figure 2: $\mathrm{L}^{\mathbb{1}\mathbb{2}}$ Static Semantics

$$\frac{\cdot}{\Gamma \vdash^{\sigma} () : \mathrm{unit}}\ \mathrm{unit} \qquad \frac{\cdot}{\Gamma \vdash^{\sigma} i : \mathrm{int}}\ \mathrm{int} \qquad \frac{\cdot}{\Gamma \vdash^{\sigma} \mathtt{true} : \mathrm{bool}}\ \mathrm{bool}_1 \qquad \frac{\cdot}{\Gamma \vdash^{\sigma} \mathtt{false} : \mathrm{bool}}\ \mathrm{bool}_2$$

$$\frac{\Gamma \vdash^{\sigma} e_1 : A \quad \Gamma, x : A^{\sigma} \vdash e_2 : B}{\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 : B}\ \mathrm{let} \qquad \frac{x : A^{\sigma} \in \Gamma}{\Gamma \vdash^{\sigma} x : A}\ \mathrm{hyp} \qquad \frac{\Gamma \vdash^{\sigma} e_1 : A \quad \Gamma \vdash^{\sigma} e_2 : B}{\Gamma \vdash^{\sigma} (e_1, e_2) : A \times B}\ \times\mathrm{I}$$

$$\frac{\Gamma \vdash^{\sigma} e : A \times B}{\Gamma \vdash^{\sigma} \pi_1\ e : A}\ \times\mathrm{E}_1 \qquad \frac{\Gamma \vdash^{\sigma} e : A \times B}{\Gamma \vdash^{\sigma} \pi_2\ e : B}\ \times\mathrm{E}_2 \qquad \frac{\Gamma \vdash^{\sigma} e_1 : \mathrm{bool} \quad \Gamma \vdash^{\sigma} e_2 : A \quad \Gamma \vdash^{\sigma} e_3 : A}{\Gamma \vdash^{\sigma} \mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 : A}\ \mathrm{if}$$

$$\frac{\Gamma \vdash^{\mathbb{2}} e : A}{\Gamma \vdash^{\mathbb{1}} \mathtt{next}\ e : \bigcirc A}\ \bigcirc\mathrm{I} \qquad \frac{\Gamma \vdash^{\mathbb{1}} e : \bigcirc A}{\Gamma \vdash^{\mathbb{2}} \mathtt{prev}\ e : A}\ \bigcirc\mathrm{E}$$

Figure 3: $\mathrm{L}^{\mathbb{1}\mathbb{2}}$ Erasure Dynamic Semantics

$$\frac{\cdot}{() \Downarrow ()}\ \mathrm{unit} \qquad \frac{\cdot}{i \Downarrow i}\ \mathrm{int} \qquad \frac{\cdot}{\mathtt{true} \Downarrow \mathbf{true}}\ \mathrm{bool}_1 \qquad \frac{\cdot}{\mathtt{false} \Downarrow \mathbf{false}}\ \mathrm{bool}_2$$

$$\frac{e_1 \Downarrow v_1 \quad [v_1/x]e_2 \Downarrow v_2}{\mathtt{let}\ x = e_1\ \mathtt{in}\ e_2 \Downarrow v_2}\ \mathrm{let} \qquad \frac{e_1 \Downarrow v_1 \quad e_2 \Downarrow v_2}{(e_1, e_2) \Downarrow (v_1, v_2)}\ \times\mathrm{I} \qquad \frac{e \Downarrow (v_1, v_2)}{\pi_1\ e \Downarrow v_1}\ \times\mathrm{E}_1 \qquad \frac{e \Downarrow (v_1, v_2)}{\pi_2\ e \Downarrow v_2}\ \times\mathrm{E}_2$$

$$\frac{e_1 \Downarrow \mathbf{true} \quad e_2 \Downarrow v}{\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \Downarrow v}\ \mathrm{if}_1 \qquad \frac{e_1 \Downarrow \mathbf{false} \quad e_3 \Downarrow v}{\mathtt{if}\ e_1\ \mathtt{then}\ e_2\ \mathtt{else}\ e_3 \Downarrow v}\ \mathrm{if}_2$$

$$\frac{e \Downarrow v}{\mathtt{next}\ e \Downarrow \mathbf{next}\ v}\ \bigcirc\mathrm{I} \qquad \frac{e \Downarrow \mathbf{next}\ v}{\mathtt{prev}\ e \Downarrow v}\ \bigcirc\mathrm{E}$$

Figure 4: L Syntax

$$\begin{array}{lll}
\langle exp \rangle ::= () \mid \langle int \rangle \mid \mathtt{true} \mid \mathtt{false} & & \langle type \rangle ::= \mathrm{unit} \mid \mathrm{int} \mid \mathrm{bool} \\
\quad \mid\ \mathtt{let}\ \langle var \rangle = \langle exp \rangle\ \mathtt{in}\ \langle exp \rangle & & \quad \mid\ \langle type \rangle \times \langle type \rangle \\
\quad \mid\ \langle var \rangle & & \langle val \rangle ::= () \mid \langle int \rangle \mid \mathbf{true} \mid \mathbf{false} \\
\quad \mid\ (\langle exp \rangle, \langle exp \rangle) & & \quad \mid\ (\langle val \rangle, \langle val \rangle) \\
\quad \mid\ \pi_1\ \langle exp \rangle \ \mid\ \pi_2\ \langle exp \rangle & & \langle cont \rangle ::= \mathrm{empty} \\
\quad \mid\ \mathtt{if}\ \langle exp \rangle\ \mathtt{then}\ \langle exp \rangle\ \mathtt{else}\ \langle exp \rangle & & \quad \mid\ \langle cont \rangle, \langle var \rangle : \langle type \rangle
\end{array}$$

Figure 5: Value Splitting

$$\frac{\cdot}{() \overset{\mathbb{1}}{\rightsquigarrow} [(), ()]}\ \mathrm{unit} \overset{\mathbb{1}}{\rightsquigarrow} \qquad \frac{\cdot}{i \overset{\mathbb{1}}{\rightsquigarrow} [i, ()]}\ \mathrm{int} \overset{\mathbb{2}}{\rightsquigarrow} \qquad \frac{\cdot}{b \overset{\mathbb{1}}{\rightsquigarrow} [b, ()]}\ \mathrm{bool} \overset{\mathbb{2}}{\rightsquigarrow}$$

$$\frac{v_1 \overset{\mathbb{1}}{\rightsquigarrow} [u_1, w_1] \quad v_2 \overset{\mathbb{1}}{\rightsquigarrow} [u_2, w_2]}{(v_1, v_2) \overset{\mathbb{1}}{\rightsquigarrow} [(u_1, u_2), (w_1, w_2)]}\ \mathrm{bool} \overset{\mathbb{2}}{\rightsquigarrow} \qquad \frac{v \overset{\mathbb{2}}{\rightsquigarrow} v'}{\mathbf{next}\ v \overset{\mathbb{1}}{\rightsquigarrow} [(), v']}\ \mathrm{bool} \overset{\mathbb{2}}{\rightsquigarrow}$$

Figure 6: Term Splitting

$$\frac{\cdot}{\Gamma \vdash^{\mathbb{1}} () : \text{unit} \overset{\mathbb{1}}{\rightsquigarrow} [((),()), \_.()]}\ \text{unit} \overset{\mathbb{1}}{\rightsquigarrow} \qquad \frac{\cdot}{\Gamma \vdash^{\mathbb{2}} () : \text{unit} \overset{\mathbb{2}}{\rightsquigarrow} [(), \_.()]}\ \text{unit} \overset{\mathbb{2}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{1}} e_1 : A \overset{\mathbb{1}}{\rightsquigarrow} [c_1, l_1.r_1] \quad \Gamma \vdash^{\mathbb{1}} e_2 : B \overset{\mathbb{1}}{\rightsquigarrow} [c_2, l_2.r_2]}{\Gamma \vdash^{\mathbb{1}} (e_1, e_2) : A \times B \overset{\mathbb{1}}{\rightsquigarrow} [((\pi_1 c, \pi_1 c), (\pi_2 c_1, \pi_2 c_2)), l.(\texttt{let}\ l_1 = \pi_1 l\ \texttt{in}\ r_1, \texttt{let}\ l_2 = \pi_2 l\ \texttt{in}\ r_2)]}\ \times\text{I} \overset{\mathbb{1}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{2}} e_1 : A \overset{\mathbb{2}}{\rightsquigarrow} [p_1, l_1.r_1] \quad \Gamma \vdash^{\mathbb{2}} e_2 : B \overset{\mathbb{2}}{\rightsquigarrow} [p_2, l_2.r_2]}{\Gamma \vdash^{\mathbb{2}} (e_1, e_2) : A \times B \overset{\mathbb{2}}{\rightsquigarrow} [(p_1, p_2), l.(\texttt{let}\ l_1 = \pi_1\ l\ \texttt{in}\ r_1, \texttt{let}\ l_2 = \pi_2\ l\ \texttt{in}\ r_2)]}\ \times\text{I} \overset{\mathbb{2}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{1}} e : A \times B \overset{\mathbb{1}}{\rightsquigarrow} [c, l.r]}{\Gamma \vdash^{\mathbb{1}} \pi_1\ e : A \overset{\mathbb{1}}{\rightsquigarrow} [(\pi_1(\pi_1 c), \pi_2 c), l.\pi_1\ r]}\ \times\text{E}_1 \overset{\mathbb{1}}{\rightsquigarrow} \qquad \frac{\Gamma \vdash^{\mathbb{2}} e : A \times B \overset{\mathbb{2}}{\rightsquigarrow} [p, l.r]}{\Gamma \vdash^{\mathbb{2}} \pi_1\ e : A \overset{\mathbb{2}}{\rightsquigarrow} [p, l.\pi_1\ r]}\ \times\text{E}_1 \overset{\mathbb{2}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{1}} e : A \times B \overset{\mathbb{1}}{\rightsquigarrow} [c, l.r]}{\Gamma \vdash^{\mathbb{1}} \pi_2\ e : B \overset{\mathbb{1}}{\rightsquigarrow} [(\pi_2(\pi_1 c), \pi_2 c), l.\pi_2\ r]}\ \times\text{E}_2 \overset{\mathbb{1}}{\rightsquigarrow} \qquad \frac{\Gamma \vdash^{\mathbb{2}} e : A \times B \overset{\mathbb{2}}{\rightsquigarrow} [p, l.r]}{\Gamma \vdash^{\mathbb{2}} \pi_2\ e : B \overset{\mathbb{2}}{\rightsquigarrow} [p, l.\pi_2\ r]}\ \times\text{E}_2 \overset{\mathbb{2}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{2}} e : A \overset{\mathbb{2}}{\rightsquigarrow} [p, l.r]}{\Gamma \vdash^{\mathbb{1}} \texttt{next}\ e : \bigcirc A \overset{\mathbb{1}}{\rightsquigarrow} [((), p), l.r]}\ \bigcirc\text{I} \overset{\mathbb{1}}{\rightsquigarrow} \qquad \frac{\Gamma \vdash^{\mathbb{1}} e : \bigcirc A \overset{\mathbb{1}}{\rightsquigarrow} [c, l.r]}{\Gamma \vdash^{\mathbb{2}} \texttt{prev}\ e : A \overset{\mathbb{2}}{\rightsquigarrow} [\pi_2\ c, l.r]}\ \bigcirc\text{E} \overset{\mathbb{2}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{1}} e_1 : \text{bool} \overset{\mathbb{1}}{\rightsquigarrow} [c_1, l_1.r_1] \quad \Gamma \vdash^{\mathbb{1}} e_2 : A \overset{\mathbb{1}}{\rightsquigarrow} [c_2, l_2.r_2] \quad \Gamma \vdash^{\mathbb{1}} e_3 : A \overset{\mathbb{1}}{\rightsquigarrow} [c_3, l_3.r_3]}{\Gamma \vdash^{\mathbb{1}} \left(\begin{array}{l}\texttt{if}\ e_1\\ \texttt{then}\ e_2\\ \texttt{else}\ e_3\end{array}\right) : A \overset{\mathbb{1}}{\rightsquigarrow} \left[\begin{array}{l}\left(\begin{array}{l}\texttt{if}\ \pi_1 c_1\\ \texttt{then let}\ y = c_2\ \texttt{in}\ (\pi_1 y, \iota_1(\pi_2 y))\\ \texttt{else let}\ y = c_3\ \texttt{in}\ (\pi_1 y, \iota_2(\pi_2 y))\end{array}\right),\\ l.(\texttt{case}\ \pi_1 l\ \texttt{of}\ l_2.r_2 \mid l_3.r_3)\end{array}\right]}\ +\text{E} \overset{\mathbb{1}}{\rightsquigarrow}$$

$$\frac{\Gamma \vdash^{\mathbb{2}} e_1 : \text{bool} \overset{\mathbb{2}}{\rightsquigarrow} [p_1, l_1.r_1] \quad \Gamma \vdash^{\mathbb{2}} e_2 : A \overset{\mathbb{2}}{\rightsquigarrow} [p_2, l_2.r_2] \quad \Gamma \vdash^{\mathbb{2}} e_3 : A \overset{\mathbb{2}}{\rightsquigarrow} [p_3, l_3.r_3]}{\Gamma \vdash^{\mathbb{2}} \left(\begin{array}{l}\texttt{if}\ e_1\\ \texttt{then}\ e_2\\ \texttt{else}\ e_3\end{array}\right) : A \overset{\mathbb{2}}{\rightsquigarrow} \left[(p_1, p_2, p_3), l.\left(\begin{array}{l}\texttt{if}\ (\texttt{let}\ l_1 = \pi_1 l\ \texttt{in}\ r_1)\\ \texttt{then}\ (\texttt{let}\ l_2 = \pi_2 l\ \texttt{in}\ r_2)\\ \texttt{else}\ (\texttt{let}\ l_3 = \pi_3 l\ \texttt{in}\ r_3)\end{array}\right)\right]}\ +\text{E} \overset{\mathbb{2}}{\rightsquigarrow}$$

2. Meaning is preserved, up to some translation.

3 Implementation

Not much to discuss. It exsists. Talk about how crufty the naive transformation is.

4 Related Work in Stage Seperation

Our starting language, λ^{12}, is essentially a two-stage version of $\lambda^{\bigcirc}$ from [1], but beefed up with products, sums, and fixed-points. That work motivates $\lambda^{\bigcirc}$ as an image of linear temporal logic under the Curry-Howard correspondence, and it makes the argument that $\lambda^{\bigcirc}$ is a good model for binding time analysis. Although $\lambda^{\bigcirc}$ was designed with partial evaluation in mind, it turns out to be a good model for pass separation as well. In contrast, [2] presents $\lambda^{\Box}$, which is based off of branching temporal logic. As that work shows, $\lambda^{\Box}$ is a good system for program generation, although we find the closed-code requirement to be more restricting than we need.

Pass seperation was introduced in [4]. They focused on motivating pass separation as a technique for compiler generation, and hinted at wider applicability. Their approach was not mechanized or automated in any sense, and their examples were separated by hand. We believe that out work represents the fulfillment of their prediction that "the [pass separation] approach will elude full automation for some time."

[3] also uses a pass separation technique for generating compiler/evaluator pairs from interpretters specified as term-rewrite systems. This heavy restriction on the form of the input prevents generalization of their style of pass separation. [I don't know enough about term-rewrite systems yet to be able to comment further.] Their method, while amenable to mechanization, was not actually implemented.

[5] represents the first attempt at fully automatic pass separation for code approaching general-purpose. In particular, they implemented pass separation for a C-like shading language including basic arithmetic and if statements. The main goal of their work, like ours, was to minimize recomputation. That said, their starting language was sufficiently restricted for recursive boundary types to be inexpressible. Given that memory is at a premium in shading languages, so much of that work was also focused on minimizing the memory-footprint of the boundary type.

References

[1] Rowan Davies. A temporal-logic approach to binding-time analysis. In *LICS*, pages 184–195, 1996.

[2] Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *J. ACM*, 48(3):555–604, 2001.

[3] John Hannan. Operational semantics-directed compilers and machine architectures. *ACM Trans. Program. Lang. Syst.*, 16(4):1215–1247, 1994.

[4] Ulrik Jørring and William L. Scherlis. Compilers and staging transformations. In *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '86, pages 86–96, New York, NY, USA, 1986. ACM.

[5] Todd B. Knoblock and Erik Ruf. Data specialization. In *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, PLDI '96, pages 215–225, New York, NY, USA, 1996. ACM.