Recursive Datatypes and Diamond Inference

Jason Reed

December 18, 2001

Abstract

We define a linear type system including recursive datatypes which tracks space usage via Hofmann's $\diamond$ type. The recursive types permitted are not as general as those found in SML, but more general than merely the binary numerals, lists, and trees in Hofmann's paper. Finally, we present an algorithm which can sometimes infer appropriate insertion of $\diamond$-variable passing in terms not carrying them.

1 Language

We begin by defining a simple linear functional programming language LinML with a restricted class of recursive types.

1.1 Types

We have all of the type constructors corresponding to the multiplicative fragment of linear logic together with additive conjunction, and the recursive types to be defined below.

$$\frac{}{\vdash 1 : \mathsf{type}}\ \mathsf{t\_unit}$$

$$\frac{\vdash \tau : \mathsf{type} \qquad \vdash \sigma : \mathsf{type}}{\vdash \tau \times \sigma : \mathsf{type}}\ \mathsf{t\_pair}$$

$$\frac{\vdash \tau : \mathsf{type} \qquad \vdash \sigma : \mathsf{type}}{\vdash \tau \otimes \sigma : \mathsf{type}}\ \mathsf{t\_tpair}$$

$$\frac{\vdash \tau : \mathsf{type} \qquad \vdash \sigma : \mathsf{type}}{\vdash \tau \multimap \sigma : \mathsf{type}}\ \mathsf{t\_lol}$$

$$\frac{d \in \Sigma \qquad \vdash \Sigma\, \mathsf{sig}}{\vdash d : \mathsf{type}}\ \mathsf{t\_base}$$

1.2 Typing

The typing rules are straightforward, but we present them here in full for the sake of completeness.

$$\frac{}{\Delta \vdash \star : 1}\;\mathsf{tp\_unit}$$

$$\frac{\Sigma \vdash \tau : \mathsf{type}}{\Delta, x : \tau, \Delta' \vdash x : \tau}\;\mathsf{tp\_var}$$

$$\frac{\Delta \vdash e_1 : \tau \qquad \Delta \vdash e_2 : \sigma}{\Delta \vdash \langle e_1, e_2 \rangle : \tau \times \sigma}\;\mathsf{tp\_pair}$$

$$\frac{\Delta \vdash e : \tau \times \sigma}{\Delta \vdash \mathbf{fst}\, e : \tau}\;\mathsf{tp\_fst} \qquad \frac{\Delta \vdash e : \tau \times \sigma}{\Delta \vdash \mathbf{snd}\, e : \sigma}\;\mathsf{tp\_snd}$$

$$\frac{\Delta \vdash e_1 : \tau \qquad \Delta' \vdash e_2 : \sigma}{\Delta, \Delta' \vdash \langle\!\langle e_1, e_2 \rangle\!\rangle : \tau \otimes \sigma}\;\mathsf{tp\_tpair}$$

$$\frac{\Delta \vdash e_1 : \tau \otimes \sigma \qquad \Delta', x : \tau, y : \sigma \vdash e_2 : \rho}{\Delta, \Delta' \vdash \mathbf{let} \langle\!\langle x, y \rangle\!\rangle = e_1\, \mathbf{in}\, e_2 : \rho}\;\mathsf{tp\_lett}$$

$$\frac{\Delta, x : \tau \vdash e : \sigma}{\Delta \vdash \lambda x{:}\tau.e : \tau \multimap \sigma}\;\mathsf{tp\_lam}$$

$$\frac{\Delta \vdash e_1 : \tau \multimap \sigma \qquad \Delta' \vdash e_2 : \tau}{\Delta, \Delta' \vdash e_1\ e_2 : \sigma}\;\mathsf{tp\_app}$$

1.3 Reduction and Evaluation

The definition of the reduction and evaluation rules is also straightforward. Reduction is

$$\frac{e \mapsto e'}{\mathbf{fst}\, e \mapsto \mathbf{fst}\, e'}\;\mathsf{cong\_fst} \qquad \frac{e \mapsto e'}{\mathbf{snd}\, e \mapsto \mathbf{snd}\, e'}\;\mathsf{cong\_snd}$$

$$\frac{}{\mathbf{fst}\, \langle e_1, e_2 \rangle \mapsto e_1}\;\mathsf{red\_fst} \qquad \frac{}{\mathbf{snd}\, \langle e_1, e_2 \rangle \mapsto e_2}\;\mathsf{red\_snd}$$

$$\frac{e_1 \mapsto e_1'}{\langle\!\langle e_1, e_2 \rangle\!\rangle \mapsto \langle\!\langle e_1', e_2 \rangle\!\rangle}\;\mathsf{cong\_tpair\_l} \qquad \frac{e_2 \mapsto e_2'}{\langle\!\langle e_1, e_2 \rangle\!\rangle \mapsto e_1 e_2'}\;\mathsf{cong\_tpair\_r}$$

$$\frac{e_1 \mapsto e_1'}{\mathbf{let} \langle\!\langle x, y \rangle\!\rangle = e_1\, \mathbf{in}\, e_2 \mapsto \mathbf{let} \langle\!\langle x, y \rangle\!\rangle = e_1'\, \mathbf{in}\, e_2}\;\mathsf{cong\_lett}$$

$$\frac{}{\mathbf{let} \langle\!\langle x, y \rangle\!\rangle = \langle\!\langle e_1, e_2 \rangle\!\rangle\, \mathbf{in}\, e \mapsto [e_1/x][e_2/y]e}\;\mathsf{red\_lett}$$

$$\frac{e_1 \mapsto e_1'}{e_1\ e_2 \mapsto e_1'\ e_2}\;\mathsf{cong\_app\_l} \qquad \frac{e_2 \mapsto e_2'}{e_1\ e_2 \mapsto e_1\ e_2'}\;\mathsf{cong\_app\_r}$$

$$\frac{}{(\lambda x{:}\tau.e_1)\ e_2 \mapsto [e_2/x]e_1}\ \textsf{red\_app}$$

and evaluation is defined by

$$\frac{}{\star \hookrightarrow \star}\ \textsf{ev\_unit}$$

$$\frac{}{\langle e_1, e_2\rangle \hookrightarrow \langle e_1, e_2\rangle}\ \textsf{ev\_pair}$$

$$\frac{e \hookrightarrow \langle e_1, e_2\rangle \qquad e_1 \hookrightarrow v_1}{\mathbf{fst}\, e \hookrightarrow v}\ \textsf{ev\_fst} \qquad \frac{e \hookrightarrow \langle e_1, e_2\rangle \qquad e_2 \hookrightarrow v}{\mathbf{snd}\, e \hookrightarrow v}\ \textsf{ev\_snd}$$

$$\frac{e_1 \hookrightarrow v_1 \qquad e_2 \hookrightarrow v_1}{\langle\!\langle e_1, e_2\rangle\!\rangle \hookrightarrow \langle\!\langle v_1, v_2\rangle\!\rangle}\ \textsf{ev\_tpair}$$

$$\frac{e_1 \hookrightarrow \langle\!\langle v_1, v_2\rangle\!\rangle \qquad [v_1/x][v_2/y]e_2 \hookrightarrow v}{\mathbf{let}\langle\!\langle x, y\rangle\!\rangle = e_1\ \mathbf{in}\, e_2 \hookrightarrow v}\ \textsf{ev\_lett}$$

$$\frac{}{\lambda x{:}\tau.e \hookrightarrow \lambda x{:}\tau.e}\ \textsf{ev\_lam}$$

$$\frac{e_1 \hookrightarrow \lambda x{:}\tau.e_1' \qquad [e_2/x]e_1' \hookrightarrow v}{e_1\ e_2 \hookrightarrow v}\ \textsf{ev\_app}$$

Several standard results hold.

Lemma 1.1 (Normal Forms) *Suppose e is normal (that is, none of the reduction rules apply). Then*

1. *If $\cdot \vdash e : 1$, then $e = \star$.*
2. *If $\cdot \vdash e : \tau_1 \otimes \tau_2$, then $e = \langle\!\langle v_1, v_2\rangle\!\rangle$ for some normal v_1, v_2.*
3. *If $\cdot \vdash e : \tau_1 \times \tau_2$, then $e = \langle e_1, e_2\rangle$ for some e_1, e_2.*
4. *If $\cdot \vdash e : \tau_1 \multimap \tau_2$, then $e = \lambda x{:}\tau_1.e_1$ for some e_1.*

Proof By induction on the typing derivation. The cases for tp_unit, tp_pair, tp_lam trivially hold. The case for tp_var holds because we assume e is closed. The case for tp_tpair by a simple appeal to the induction hypothesis. A typical case for one of the remaining (elimination) rules is

Case:

$$\frac{\begin{array}{c}\mathcal{D}_1\\ \cdot \vdash e : \tau \times \sigma\end{array}}{\cdot \vdash \mathbf{fst}\, e : \tau}\ \textsf{tp\_fst}$$

If $\mathbf{fst}\, e$ is irreducible, then so too is e — otherwise we could apply cong_fst. By the induction hypothesis applied to $\mathcal{D}_1$, we know that e is of the form $\langle e_1, e_2\rangle$, and therefore $\mathbf{fst}\,\langle e_1, e_2\rangle$ is reducible by red_fst, a contradiction.

■

Lemma 1.2 (Substitution) *If $\Delta, x : \tau \vdash e : \sigma$ and $\Delta' \vdash e' : \tau$, then $\Delta, \Delta' \vdash [e'/x]e : \sigma$.*

Proof Straightforward induction on $\mathcal{D} :: \Delta, x : \tau \vdash e : \sigma$. ∎

Lemma 1.3 (Subject Reduction) *If* $\cdot \vdash e : \tau$ *and* $e \mapsto e'$, *then* $\cdot \vdash e' : \tau$.

Proof Straightforward induction on the typing derivation, using the substitution lemma. ∎

Lemma 1.4 *If* $e \hookrightarrow v$, *then* $e \mapsto^* v$.

Proof Straightforward induction. ∎

Corollary 1.5 (Type Preservation) *If* $\cdot \vdash e : \tau$ *and* $e \hookrightarrow v$, *then* $\cdot \vdash v : \tau$.

1.4 Recursive Types

In the sequel we adopt the abbreviation $\tau^n := \overbrace{\tau \multimap \cdots \multimap \tau}^{n}$. when τ^n appears in the domain of a function type, and $\tau^n := \overbrace{\tau \otimes \cdots \otimes \tau}^{n}$ otherwise. In the former case, say $\tau^n \multimap \sigma$, we mean $\tau \multimap (\tau \multimap \cdots \multimap (\tau \multimap \sigma))$ not $(\tau \multimap (\tau \multimap \cdots \multimap \tau)) \multimap \sigma$. This is precisely so that τ^n is still "morally equivalent" to $\overbrace{\tau \otimes \cdots \otimes \tau}^{n}$, but we want all of our functions fully curried.

The definition of a valid signature contains the syntax of recursive type declarations:

$$\frac{}{\vdash \cdot\ \mathsf{sig}}\ \mathsf{sig\_nil}$$

$$\frac{\vdash \Sigma\ \mathsf{sig} \qquad \forall i \leq n. \forall j \leq m_i.(\vdash_\Sigma \tau_i : \mathsf{type})}{\vdash \Sigma, d = \mu\alpha. \bigoplus_{i=1}^{n} \alpha^{m_i} \otimes \tau_i\ \mathsf{sig}}\ \mathsf{sig\_cons}$$

Let **d** stand for the assumption that

$$d = \mu\alpha. \bigoplus_{i=1}^{n} \alpha^{m_i} \otimes \tau_i \in \Sigma$$

To clarify the intent of the notation, in a more ML-like language one might write the same definition as

$$\begin{array}{rcl}
\textbf{datatype}\ d & = & inj_1\ \textbf{of}\ \overbrace{d \otimes \cdots \otimes d}^{m_1} \otimes \tau_1 \\
& | & inj_2\ \textbf{of}\ \overbrace{d \otimes \cdots \otimes d}^{m_2} \otimes \tau_2 \\
& \vdots & \\
& | & inj_n\ \textbf{of}\ \overbrace{d \otimes \cdots \otimes d}^{m_n} \otimes \tau_n
\end{array}$$

We extend the language of terms by constructors **inj** and **iter** which are typed as follows:

$$\frac{\mathbf{d} \qquad \forall j \leq m_i.(\Delta_j \vdash p_j : d) \qquad \Delta \vdash q : \tau_i}{\Delta_1, \ldots, \Delta_{m_i}, \Delta \vdash \mathbf{inj}_{d,i}(p_1, \ldots, p_{m_i}, q) : d}\ \mathsf{tp\_inj}$$

$$\frac{\mathbf{d} \qquad \Delta \vdash e' : d \qquad \forall i \leq n.(\cdot \vdash e_i : \sigma^{m_i} \multimap \tau_i \multimap \sigma))}{\Delta \vdash \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e') : \sigma}\ \mathsf{tp\_iter}$$

The form of recursive types permitted is chosen so iteration always makes sense. We have new reduction rules

$$\frac{e_j \mapsto e'_j}{\mathbf{inj}_{d,i}(e_1, \ldots, e_{m_i}, t) \mapsto \mathbf{inj}_{d,i}(e_1, \ldots, e'_j, \ldots, e_{m_i}, t)}\ \mathsf{cong\_inj}$$

$$\frac{e \mapsto e'}{\mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e) \mapsto \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e')}\ \mathsf{cong\_iter}$$

$$\frac{}{\begin{array}{c}\mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, \mathbf{inj}_{d,i}(e'_1, \ldots, e'_{m_i}, e'')) \\ \mapsto e_i\ \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e'_1)\ \cdots\ \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e'_{m_i})\ e''\end{array}}\ \mathsf{red\_iter}$$

and a new evaluation rule

$$\frac{\begin{array}{c}e' \hookrightarrow \mathbf{inj}_{d,i}(v_1, \ldots, v_{m_i}, w) \\ e_i\ \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, v_1) \cdots \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, v_{m_i})\ w \hookrightarrow v\end{array}}{\mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e') \hookrightarrow v}\ \mathsf{ev\_iter}$$

Now we can extend Lemma 1.1 with the clauses

> If $\mathbf{d}$ and $\cdot \vdash e : d$, then $e = \mathbf{inj}_{d,i}(e_1, \ldots, e_{m_i}, e')$ for some i, some normal $e_1, \ldots, e_{m_i}$, and some e'.

and observe that the other basic lemmas still hold with the addition of recursive types.

1.5 Diamond

The contribution of [Hof00] is the constructorless type $\diamond$ required in constructors of recursive datatypes, and produced in destructors of them. We define a variation $\mathsf{LinML}^{\diamond}$ of LinML by adding a type $\diamond$ and changing the typing, reduction, and evaluation rules. We also annotate the typing turnstile with a $\diamond$ to indicate when we are working in $\mathsf{LinML}^{\diamond}$ rather than LinML. The rules $\mathsf{tp\_inj}$ and $\mathsf{tp\_iter}$ are replaced by

$$\frac{\mathbf{d} \qquad \forall j \leq m_i.(\Delta_j \vdash^{\diamond} r_j : \diamond) \qquad \forall j \leq m_i.(\Delta'_j \vdash^{\diamond} p_j : d) \qquad \Delta \vdash^{\diamond} q : \tau_i}{\Delta_1, \ldots, \Delta_{m_i}, \Delta'_1, \ldots, \Delta'_{m_i}, \Delta \vdash^{\diamond} \mathbf{inj}^{\diamond}_{d,i}(r_1, \ldots, r_{m_i}, p_1, \ldots, p_{m_i}, q) : d}\ \mathsf{tp\_inj}\diamond$$

$$\frac{\begin{array}{c}\mathbf{d} \qquad \Delta \vdash^{\diamond} e' : d \\ \forall i \leq n.(\cdot \vdash^{\diamond} e_i : \diamond^{m_i} \multimap \sigma^{m_i} \multimap \tau_i \multimap \sigma))\end{array}}{\Delta \vdash^{\diamond} \mathbf{iter}^{\diamond}_{d,\sigma}(e_1, \ldots, e_n, e') : \sigma}\ \mathsf{tp\_iter}\diamond$$

The reduction rules are replaced by

$$\frac{e_j \mapsto e'_j}{\begin{array}{c}\mathbf{inj}^{\diamond}_{d,i}(r_1,\ldots,r_{m_i},e_1,\ldots,e_{m_i},t) \\ \mapsto \mathbf{inj}^{\diamond}_{d,i}(r_1,\ldots,r_{m_i},e_1,\ldots,e'_j,\ldots,e_{m_i},t)\end{array}}\ \mathsf{cong\_inj}\diamond$$

$$\frac{e \mapsto e'}{\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e) \mapsto \mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e')}\ \mathsf{cong\_iter}\diamond$$

$$\frac{}{\begin{array}{c}\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,\mathbf{inj}^{\diamond}_{d,i}(r_1,\ldots,r_{m_i},e'_1,\ldots,e'_{m_i},e'')) \\ \mapsto e_i\ r_1\cdots r_{m_i}\ \mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e'_1)\ \cdots\ \mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e'_{m_i})\ e''\end{array}}\ \mathsf{red\_iter}\diamond$$

and the evaluation rule $\mathsf{ev\_iter}$ is replaced by

$$\frac{\begin{array}{c}e' \hookrightarrow \mathbf{inj}^{\diamond}_{d,i}(r_1,\ldots,r_{m_i},v_1,\ldots,v_{m_i},w) \\ e_i\ r_1\cdots r_{m_i}\ \mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,v_1)\cdots\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,v_{m_i})\ w \hookrightarrow v\end{array}}{\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e') \hookrightarrow v}\ \mathsf{ev\_iter}\diamond$$

The relevant normal forms clause is then

> If $\mathbf{d}$ and $\cdot \vdash^{\diamond} e : d$, then $e = \mathbf{inj}_{d,i}(r_1,\ldots,r_{m_i},e_1,\ldots,e_{m_i},e')$ (for some i, $r_1,\ldots,r_{m_i}$, normal $e_1,\ldots,e_{m_i}$, and some e')

and again it is easily seen that the other lemmas remain valid.

1.6 Polynomial-time Guarantees

We follow the technique and notation of [AS01]. The *size* $|t|$ of a term t is defined as follows:

$$\begin{array}{rcl}
|c| := |x| & := & 1 \\
|\lambda x{:}\,\tau.e| & := & 1+|e| \\
|e_1\ e_2| & := & |e_1|+|e_2| \\
|\langle\!\langle e_1,e_2\rangle\!\rangle| & := & 1+|e_1|+|e_2| \\
|\mathbf{fst}\,e| := |\mathbf{snd}\,e| & := & 1+|e| \\
|\langle e_1,e_2\rangle| & := & 1+\max\{|e_1|,|e_2|\} \\
|\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e')| & := & 1+|e'| \\
|\mathbf{inj}^{\diamond}_{d,i}(r_1,\ldots,r_{m_i},v_1,\ldots,v_{m_i},w)| & := & 1+|w|+\displaystyle\sum_{j=1}^{m_i}|r_j|+|v_j|
\end{array}$$

We define $\vartheta(t) \in \mathbb{N}[X]$, the *polynomial of* t by:

$$\begin{array}{rcl}
\vartheta(c) := \vartheta(x) & := & 0 \\
\vartheta(\lambda x{:}\,\tau.e) & := & \vartheta(e)
\end{array}$$

$$
\begin{aligned}
\vartheta(e_1\ e_2) &:= \vartheta(e_1) + \vartheta(e_2)\\
\vartheta(\langle\!\langle e_1, e_2\rangle\!\rangle) &:= \vartheta(e_1) + \vartheta(e_2)\\
\vartheta(\mathbf{fst}\, e) := \vartheta(\mathbf{snd}\, e) &:= \vartheta(e)\\
\vartheta(\langle e_1, e_2\rangle) &:= \sup\nolimits_{\preceq}\{\vartheta(e_1), \vartheta(e_2)\}\\
\vartheta(\mathbf{iter}^{\diamond}_{d,\sigma}(e_1, \ldots, e_n, e')) &:= \vartheta(e') + (X+1)\cdot\\
&\quad\ (\sup\nolimits_{\preceq}\{\vartheta(e_i) \mid i \leq n\} + \max\{|e_i| \mid i \leq n\})\\
\vartheta(\mathbf{inj}^{\diamond}_{d,i}(r_1, \ldots, r_{m_i},&\\
v_1, \ldots, v_{m_i}, w)) &:= \vartheta(w) + \sum_{j=1}^{m_i} \vartheta(r_j) + \vartheta(v_j)
\end{aligned}
$$

We first observe that, by affineness, term size and polynomial are nonincreasing under substitution into well-typed terms.

Lemma 1.6 *If $\Delta, x : \tau \vdash e : \sigma$ and $\Delta' \vdash e' : \tau$, then $|[e'/x]e| \leq |e| + |e'|$.*

Proof Straightforward induction. ■

Lemma 1.7 *If $\Delta, x : \tau \vdash e : \sigma$ and $\Delta' \vdash e' : \tau$, then $\vartheta([e'/x]e) \preceq \vartheta(e) + \vartheta(e')$.*

Proof Straightforward induction. ■

The central lemma is then the following:

Lemma 1.8 *Suppose $\Delta = x_1 : \diamond, \ldots, x_N : \diamond$ and $\Delta \vdash e : \tau$. Then $e \mapsto^k e'$ (that is, $\underbrace{e \mapsto \cdots \mapsto e'}_{k \text{ steps}}$) with e' normal and*

$$\vartheta(e')(N) + |e'| + k \leq \vartheta(e)(N) + |e|$$

for some e', k.

Proof By induction on $\vartheta(e)(N) + |e|$ and case analysis taking advantage of the syntax-directedness of the typing judgment. We show some representative cases:

Case:

$$\frac{\Delta \vdash e : \tau \times \sigma}{\Delta \vdash \mathbf{fst}\, e : \tau}\ \mathsf{tp\_fst}$$

Observe that $\vartheta(e)(N) + |e| < \vartheta(e)(N) + |e| + 1 = \vartheta(\mathbf{fst}\, e)(N) + |\mathbf{fst}\, e|$. Therefore we can apply the induction hypothesis to e, and find that $e \mapsto^k e'$ such that

$$\vartheta(e')(N) + |e'| + k \leq \vartheta(e)(N) + |e|$$

and e' normal. By the subject reduction and normal forms lemmas, we know that e' is of the form $\langle e_1, e_2\rangle$ for some e_1, e_2. Therefore by using

$\mathsf{cong\_fst}$ and $\mathsf{red\_fst}$ we have $\mathbf{fst}\, e \mapsto^k \mathbf{fst}\, \langle e_1, e_2 \rangle \mapsto e_1$. By the definition of term size and polynomial we know

$$|e'| = |\langle e_1, e_2 \rangle| = 1 + \max\{|e_1|, |e_2|\} \geq 1 + |e_1| > |e_1|$$

and

$$\vartheta(e') = \vartheta(\langle e_1, e_2 \rangle) = \sup_{\preceq}\{\vartheta(e_1), \vartheta(e_2)\} \succeq \vartheta(e_1)$$

so $\vartheta(e_1)(N) + |e_1| < \vartheta(e')(N) + |e'| \leq \vartheta(e')(N) + |e'| + k \leq \vartheta(e)(N) + |e| < \vartheta(\mathbf{fst}\, e)(N) + |\mathbf{fst}\, e|$. Therefore we apply the induction hypothesis again, to e_1, and find that $e_1 \mapsto^{k'} e_1'$ with e_1' normal such that

$$\vartheta(e_1')(N) + |e_1'| + k' \leq \vartheta(e_1)(N) + |e_1|$$

Now we can see that $\mathbf{fst}\, e \mapsto^k \mathbf{fst}\, \langle e_1, e_2 \rangle \mapsto e_1 \mapsto^{k'} e_1'$, or in other words $\mathbf{fst}\, e \mapsto^{k+k'+1} e_1'$. It remains to check

$$\begin{array}{rcl} \vartheta(e_1')(N) + |e_1'| + k + k' + 1 & \leq & \vartheta(e_1)(N) + |e_1| + k + 1 \\ & \leq & \vartheta(e')(N) + |e'| + k \\ & \leq & \vartheta(\mathbf{fst}\, e)(N) + |e| \end{array}$$

Case:

$$\frac{\begin{array}{c}\mathbf{d} \qquad \Delta \vdash^{\diamond} e' : d \\ \forall i \leq n.(\cdot \vdash^{\diamond} e_i : \diamond^{m_i} \multimap \sigma^{m_i} \multimap \tau_i \multimap \sigma))\end{array}}{\Delta \vdash^{\diamond} \mathbf{iter}^{\diamond}_{d,\sigma}(e_1, \ldots, e_n, e') : \sigma} \;\mathsf{tp\_iter}\diamond$$

By the induction hypothesis on e', we have $e' \mapsto^s e''$ such that

$$\vartheta(e'')(N) + |e''| + s \leq \vartheta(e')(N) + |e'| \qquad (*)$$

By the subject reduction and normal forms lemmas, we know that e'' is of the form $\mathbf{inj}^{\diamond}_{d,i}(r_1, \ldots, r_{m_i}, e_1', \ldots, e_{m_i}', t)$ for some i, $r_1, \ldots, r_{m_i}$, $e_1', \ldots$, e_{m_i}', and t, such that $e_1', \ldots, e_{m_i}'$ are also normal of type d, so they too are of the form $\mathbf{inj}^{\diamond} \cdots$, and so forth. That is, e'' can be thought of as a tree with nodes $\nu_1, \ldots, \nu_K$ for some K. Suppose without loss of generality that ν_1 is the root of the tree. Each node ν_k is an injection, say into case c_k of the type d, and so has m_{c_k} children, call them $\nu_{\chi_{k,1}}, \ldots, \nu_{\chi_{k,m_{c_k}}}$, and uses m_{c_k} terms $r_{k,1}, \ldots, r_{k,m_{c_k}}$ of type $\diamond$. Concretely, the term $T(k)$ corresponding to the node ν_k is

$$T(k) = \mathbf{inj}^{\diamond}_{d,c_k}(r_{k,1}, \ldots, r_{k,m_{c_k}}, T(\chi_{k,1}), \ldots, T(\chi_{k,m_{c_k}}), t_k)$$

Therefore we can see that

$$\mathbf{iter}^{\diamond}_{d,\sigma}(e_1, \ldots, e_n, e') \mapsto^s \mathbf{iter}^{\diamond}_{d,\sigma}(e_1, \ldots, e_n, T(1)) \mapsto^K U(1)$$

where

$$U(k) = e_{c_k}\ r_{k,1} \cdots r_{k,m_{c_k}}\ U(\chi_{k,1}) \cdots U(\chi_{k,m_{c_k}})\ t_k$$

Note that

$$\vartheta(U(1)) \preceq (N+1)\sup_{\preceq}\{\vartheta(e_i) \mid i \leq n\} + \vartheta(\vec{r}) + \vartheta(\vec{t})$$

(where $\vec{r}$ and $\vec{t}$ denote all subscripted r and t terms, respectively, and where the polynomial or size of a vector of terms is the evident sum) because the number N of free diamonds bounds the number of non-root nodes that can exist in the tree $\{\nu_k \mid k \leq K\}$. For the same reason, we have

$$|U(1)| \leq (N+1)\max\{|e_i| \mid i \leq n\} + |\vec{r}| + |\vec{t}|$$

We can also directly compute the polynomial and size of the term $e'' = T(1)$:

$$\vartheta(T(1)) = \vartheta(\vec{r}) + \vartheta(\vec{t})$$
$$|T(1)| = |\vec{r}| + |\vec{t}| + K$$

It is straightforward to check from what we already know that

$$\begin{aligned}\vartheta(U(1))(N) + |U(1)| < \vartheta(\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e'))(N)+ \\ |\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e')|\end{aligned}$$

So by the induction hypothesis, we normalize $U(1)$ as $U(1) \mapsto^{s'} v$ and learn

$$\vartheta(v)(N) + |v| + s' \leq \vartheta(U(1))(N) + |U(1)|$$

Finally, we observe that we know

$$\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e') \mapsto^{s+K+s'} v$$

and compute

$$\begin{array}{cl}
 & \vartheta(v)(N) + |v| + s + K + s' \\
\leq & \vartheta(U(1))(N) + |U(1)| + s + K \\
\leq & ((N+1)\sup_{\preceq}\{\vartheta(e_i) \mid i \leq n\})(N) + \vartheta(\vec{r})(N) + \vartheta(\vec{t})(N) + \\
 & (N+1)\max\{|e_i| \mid i \leq n\} + |\vec{r}| + |\vec{t}| + s + K \\
= & ((X+1)\cdot(\sup_{\preceq}\{\vartheta(e_i) \mid i \leq n\} + \max\{|e_i| \mid i \leq n\}))(N) + \\
 & +\vartheta(\vec{r})(N) + \vartheta(\vec{t})(N) + |\vec{r}| + |\vec{t}| + s + K \\
\leq & ((X+1)\cdot(\sup_{\preceq}\{\vartheta(e_i) \mid i \leq n\} + \max\{|e_i| \mid i \leq n\}))(N) + \\
 & +\vartheta(T(1))(N) + |T(1)| + s \\
\leq & ((X+1)\cdot(\sup_{\preceq}\{\vartheta(e_i) \mid i \leq n\} + \max\{|e_i| \mid i \leq n\}))(N) + \\
 & +\vartheta(e')(N) + |e'| \\
< & ((X+1)\cdot(\sup_{\preceq}\{\vartheta(e_i) \mid i \leq n\} + \max\{|e_i| \mid i \leq n\}))(N) + \\
 & +\vartheta(e')(N) + |e'| + 1 \\
= & \vartheta(\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e'))(N) + |\mathbf{iter}^{\diamond}_{d,\sigma}(e_1,\ldots,e_n,e')|
\end{array}$$

■

2 Diamond Inference

2.1 N-typing

We define a new judgment $\Delta \vdash^N e : \tau$ on contexts, expressions, and types from the language LinML. Intuitively, $\Delta \vdash^N e : \tau$ means that we do have $\Delta \vdash e : \tau$, and we would have $\Delta \vdash^{\diamond} e : \tau$ if only we had N terms of type $\diamond$ to spare.

$$\frac{}{\Delta \vdash^0 \star : 1}\ \mathsf{ntp\_unit}$$

$$\frac{}{\Delta, x : \tau, \Delta' \vdash^0 x : \tau}\ \mathsf{ntp\_var}$$

$$\frac{\Delta \vdash^{N_1} e_1 : \tau \qquad \Delta \vdash^{N_2} e_2 : \sigma}{\Delta \vdash^{\max\{N_1, N_2\}} \langle e_1, e_2 \rangle : \tau \times \sigma}\ \mathsf{ntp\_pair}$$

$$\frac{\Delta \vdash^N e : \tau \times \sigma}{\Delta \vdash^N \mathbf{fst}\, e : \tau}\ \mathsf{ntp\_fst} \qquad \frac{\Delta \vdash^N e : \tau \times \sigma}{\Delta \vdash^N \mathbf{snd}\, e : \sigma}\ \mathsf{ntp\_snd}$$

$$\frac{\Delta \vdash^{N_1} e_1 : \tau \qquad \Delta' \vdash^{N_2} e_2 : \sigma}{\Delta, \Delta' \vdash^{N_1+N_2} \langle\!\langle e_1, e_2 \rangle\!\rangle : \tau \otimes \sigma}\ \mathsf{ntp\_tpair}$$

$$\frac{\Delta \vdash^{N_1} e_1 : \tau \otimes \sigma \qquad \Delta', x : \tau, y : \sigma \vdash^{N_2} e_2 : \rho}{\Delta, \Delta' \vdash^{N_1+N_2} \mathbf{let} \langle\!\langle x, y \rangle\!\rangle = e_1\, \mathbf{in}\, e_2 : \rho}\ \mathsf{ntp\_lett}$$

$$\frac{\Delta, x : \tau \vdash^N e : \sigma}{\Delta \vdash^N \lambda x :: \tau. e : \tau \multimap \sigma}\ \mathsf{ntp\_lam}$$

$$\frac{\Delta \vdash^{N_1} e_1 : \tau \multimap \sigma \qquad \Delta' \vdash^{N_2} e_2 : \tau}{\Delta, \Delta' \vdash^{N_1+N_2} e_1\ e_2 : \sigma}\ \mathsf{ntp\_app}$$

$$\frac{\mathbf{d} \qquad \forall j \le m_i.(\Delta_j \vdash^{N_j} p_j : d) \qquad \Delta \vdash^M q : \tau_i}{\Delta_1, \ldots, \Delta_{m_i}, \Delta \vdash^{m_i + M + \sum_j N_j} \mathbf{inj}_{d,i}(p_1, \ldots, p_{m_i}, q) : d}\ \mathsf{ntp\_inj}$$

$$\frac{\begin{array}{c} \forall i \le n.(\cdot \vdash^{N_i} e_i : \sigma^{m_i} \multimap \tau_i \multimap \sigma)) \\ \mathbf{d} \qquad \Delta \vdash^M e' : d \qquad \forall i \le n.(N_i + \mu m_i \le m_i + \mu) \end{array}}{\Delta \vdash^{M+\mu} \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e') : \sigma}\ \mathsf{ntp\_iter}$$

where $\mu = \max\{N_i \,|\, m_i = 0\}$

Lemma 2.1 *If $\Delta \vdash^N e : \tau$, then $\Delta \vdash e : \tau$.*

Proof Straightforward induction. ■

Lemma 2.2 *If $\Delta \vdash^N e : \tau$, then there exists e^* such that $\Delta \vdash^{\diamond} e^* : \diamond^N \multimap \tau$.*

Proof By induction on the structure of $\mathcal{D} :: \Delta \vdash^N e : \tau$. In fact we construe the constructive proof of the statement of the lemma as a recursive function $—^*$ which takes a derivation $\mathcal{D}$ and returns an expression e^*. Since the resulting expression depends on the existence of a derivation for some Δ and not merely e, it is not strictly correct to write e^*, though we do so for brevity when the relevant derivation is easily inferred. Throughout the proof, we use the abbreviation $\lambda^N \vec{x}{:}\tau.\cdots e\ \vec{x} \cdots$ for $\lambda x_1{:}\tau.\cdots\lambda x_N{:}\tau.\cdots e\ x_1 \cdots x_n \cdots$.

Case:

$$\frac{}{\Delta \vdash^0 \star : \mathbf{1}}\,\textsf{ntp\_unit}$$

Put $\star^* := \star$.

Case:

$$\frac{}{\Delta, x : \tau, \Delta' \vdash^0 x : \tau}\,\textsf{ntp\_var}$$

Put $x^* := x$.

Case:

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ \Delta \vdash^{N_1} e_1 : \tau & \Delta \vdash^{N_2} e_2 : \sigma\end{array}}{\Delta \vdash^{\max\{N_1,N_2\}} \langle e_1, e_2\rangle : \tau \times \sigma}\,\textsf{ntp\_pair}$$

By the induction hypothesis on $\mathcal{D}_1, \mathcal{D}_2$, we have $\Delta \vdash^{\Diamond} e_1^* : \Diamond^{N_1} \to \tau$ and $\Delta \vdash^{\Diamond} e_2^* : \Diamond^{N_2} \to \sigma$. Put

$$\langle e_1, e_2\rangle^* := \lambda^{\max\{N_1,N_2\}}\vec{x}{:}\Diamond.\langle e_1^*\ x_1 \cdots x_{N_1}, e_2^*\ x_1 \cdots x_{N_2}\rangle$$

Case:

$$\frac{\begin{array}{c}\mathcal{D}_1\\ \Delta \vdash^{N} e : \tau \times \sigma\end{array}}{\Delta \vdash^{N} \mathbf{fst}\, e : \tau}\,\textsf{ntp\_fst}$$

By the induction hypothesis on $\mathcal{D}_1$, we have $\Delta \vdash^{\Diamond} e^* : \Diamond^N \to \tau$. Put

$$(\mathbf{fst}\, e_1)^* := \lambda^N \vec{x}{:}\Diamond.\, \mathbf{fst}(e\ \vec{x})$$

Case:

$$\frac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2\\ \Delta \vdash^{N_1} e_1 : \tau & \Delta' \vdash^{N_2} e_2 : \sigma\end{array}}{\Delta, \Delta' \vdash^{N_1+N_2} \langle\!\langle e_1, e_2\rangle\!\rangle : \tau \otimes \sigma}\,\textsf{ntp\_tpair}$$

By the induction hypothesis on $\mathcal{D}_1, \mathcal{D}_2$, we have $\Delta \vdash^{\Diamond} e_1^* : \Diamond^{N_1} \to \tau$ and $\Delta \vdash^{\Diamond} e_2^* : \Diamond^{N_2} \to \sigma$. Put

$$\langle\!\langle e_1, e_2\rangle\!\rangle^* := \lambda^{N_1}\vec{x}{:}\Diamond.\lambda^{N_2}\vec{y}{:}\Diamond.\langle\!\langle e_1^*\ \vec{x}, e_2^*\ \vec{y}\rangle\!\rangle$$

Case:

$$\frac{\mathbf{d} \qquad \forall j \leq m_i.(\overset{\mathcal{D}_j}{\Delta_j \vdash^{N_j} p_j : d}) \quad \overset{\mathcal{D}}{\Delta \vdash^{M} q : \tau_i}}{\Delta_1, \ldots, \Delta_{m_i}, \Delta \vdash^{m_i + M + \sum_j N_j} \mathbf{inj}_{d,i}(p_1, \ldots, p_{m_i}, q) : d}\ \mathsf{ntp\_inj}$$

By the induction hypothesis on $\mathcal{D}, \mathcal{D}_j$ we have $\Delta \vdash^{\diamond} p_j^* : \diamond^{N_j} \multimap d$ and $\Delta \vdash^{\diamond} q^* : \diamond^M \multimap \tau_i$. Put

$$(\mathbf{inj}_{d,i}(p_1, \ldots, p_{m_i}, q))^* :=$$

$$\lambda^{m_i} \vec{r} : \diamond . \lambda^{N_1} \vec{x}_1 : \diamond . \cdots \lambda^{N_{m_i}} \vec{x}_{m_i} : \diamond . \lambda^M \vec{y} : \diamond .$$

$$\mathbf{inj}^{\diamond}_{d,\sigma}(r_1, \ldots, r_{m_i}, p_1^*\ \vec{x}_1, \ldots, p_{m_i}^*\ \vec{x}_{m_i}, e'^*\ \vec{y})$$

Case:

$$\frac{\mathbf{d} \quad \overset{\mathcal{D}'}{\Delta \vdash^M e' : d} \qquad \overset{\forall i \leq n. \mathcal{D}_i :: (\cdot \vdash^{N_i} e_i : \sigma^{m_i} \multimap \tau_i \multimap \sigma))}{\forall i \leq n.(N_i + \mu m_i \leq m_i + \mu)}}{\Delta \vdash^{M+\mu} \mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e') : \sigma}\ \mathsf{ntp\_iter}$$

By the induction hypothesis on $\mathcal{D}', \mathcal{D}_1, \ldots, \mathcal{D}_n$ we have $\Delta \vdash^{\diamond} e'^* : \diamond^M \multimap d$ and also $\cdot \vdash^{\diamond} e_i^* : \diamond^{N_i} \multimap \sigma^{m_i} \multimap \tau_i \multimap \sigma$ for all $i \leq n$. Set

$$(\mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e'))^* := \lambda^M \vec{x} : \diamond . \mathbf{iter}^{\diamond}_{d, \diamond^\mu \multimap \sigma}(e_1^\sharp, \ldots, e_n^\sharp, e'^*\ \vec{x})$$

Where

$$e_i^\sharp = \lambda t : \tau_i . \lambda^\mu \vec{x} : \diamond . e_i^*\ x_1 \cdots x_{N_i}\ t$$

if $m_i = 0$ and

$$e_i^\sharp = \lambda^{m_i} \vec{x} : \diamond . \lambda^{m_i} \vec{f} : \diamond^\mu \multimap \sigma . \lambda t : \tau_i . \lambda x_{m_i+1} : \diamond . \cdots \lambda x_{m_i+\mu} : \diamond .$$

$$e_i^*\ x_1 \cdots x_{N_i} (f_1\ x_{N_i+1} \cdots x_{N_i+\mu}) \cdots (f_{m_i}\ x_{N_i+\mu(m_i-1)+1} \cdots x_{N_i+\mu m_i})\ t$$

otherwise. These are typed as required, namely

$$\cdot \vdash^{\diamond} e_i^\sharp : \diamond^{m_i} \multimap (\diamond^\mu \multimap \sigma)^{m_i} \multimap \tau_i \multimap \diamond^\mu \multimap \sigma$$

because of the conditions that $N_i \leq \mu$ if $m_i = 0$ and $N_i + \mu m_i \leq m_i + \mu$. Therefore

$$\Delta, x_1 : \diamond, \ldots, x_M : \diamond \vdash^{\diamond} \mathbf{iter}^{\diamond}_{d, \diamond^\mu \multimap \sigma}(e_1^\sharp, \ldots, e_n^\sharp, e'^*\ \vec{x}) : \diamond^\mu \multimap \sigma$$

and consequently

$$\Delta \vdash^{\diamond} (\mathbf{iter}_{d,\sigma}(e_1, \ldots, e_n, e'))^* : \diamond^{M+\mu} \multimap \sigma$$

as required.

■

2.2 An Example

Insertion sort: Suppose we have a type τ and a function $\mathsf{swap} : \tau \multimap \tau \multimap \tau \otimes \tau$ which implements an order $\preceq$ on τ by

$$\mathsf{swap}\ x\ y = \begin{cases} \langle\!\langle x, y \rangle\!\rangle & \text{if } x \preceq y; \\ \langle\!\langle y, x \rangle\!\rangle & \text{otherwise.} \end{cases}$$

We would like to implement insertion sort by

$$\begin{array}{rcl} \mathsf{insert}\,[]\ a & = & [a] \\ \mathsf{insert}\,(h :: t)\ a & = & \mathbf{let}\ \langle\!\langle a_1, a_2 \rangle\!\rangle = \mathsf{swap}\ a\ h\,\mathbf{in}\,a_1 :: \mathsf{insert}\ a_2\ t \\ \mathsf{sort}\,[] & = & [] \\ \mathsf{sort}(h :: t) & = & \mathsf{insert}\,(\mathsf{sort}\ t)\ h \end{array}$$

The type of lists is

$$L(\tau) = \mu\alpha.(\alpha^0 \otimes 1) \oplus (\alpha^1 \otimes \tau)$$

and we make the evident abbreviations

$$[] := \mathbf{inj}_{L(\tau),1}(\star)$$

$$h :: t := \mathbf{inj}_{L(\tau),2}(t, h)$$

$$[a_1, a_2, \ldots, a_n] := a_1 :: (a_2 :: (\cdots (a_n :: []) \cdots))$$

So we use the iteration primitive to obtain recursion as follows:

$$\mathsf{insert} := \lambda\ell{:}\, L(\tau).(\mathbf{iter}_{L(\tau),\tau \multimap L(\tau)}((\lambda a{:}\,\tau.[a]),$$

$$(\lambda f{:}\,\tau \multimap L(\tau).\lambda h{:}\,\tau.\lambda a{:}\,\tau.\,\mathbf{let}\ \langle\!\langle a_1, a_2 \rangle\!\rangle = \mathsf{swap}\ a\ h\,\mathbf{in}\,a_1 :: (f\ a_2)), \ell))$$

$$\mathsf{sort} := \lambda\ell{:}\, L(\tau).\mathbf{iter}_{L(\tau),L(\tau)}([], \mathsf{insert}, \ell)$$

We can obtain annotated typing judgments for these terms by

$a : \tau \vdash^0 a : \tau$ ntp_var

$\cdot \vdash^0 [] : L(\tau)$ ntp_inj

$a : \tau \vdash^1 a :: [] : L(\tau)$ ntp_inj

$\cdot \vdash^1 \lambda a{:}\tau.a :: [] : \tau \multimap L(\tau)$ ntp_lam

$a_1 : \tau \vdash^0 a_1 : \tau$ ntp_var

$a_2 : \tau \vdash^0 a_2 : \tau$ ntp_var

$f : \tau \multimap L(\tau) \vdash^0 f : \tau \to L(\tau)$ ntp_var

$a_2 : \tau, f : \tau \multimap L(\tau) \vdash^0 f\ a_2 : L(\tau)$ ntp_app

$a_1 : \tau, a_2 : \tau, f : \tau \multimap L(\tau) \vdash^1 a_1 :: (f\ a_2)$ ntp_inj

$h : \tau \vdash^0 h : \tau$ ntp_var

$\cdot \vdash^0 \mathsf{swap} : \tau \multimap \tau \multimap \tau \otimes \tau$ Assumption

$a : \tau, h : \tau \vdash^0 \mathsf{swap}\ a\ h : \tau \otimes \tau$ ntp_app

$a : \tau, h : \tau, f : \tau \multimap L(\tau) \vdash^1 \mathbf{let} \langle\!\langle a_1, a_2 \rangle\!\rangle = \mathsf{swap}\ a\ h\ \mathbf{in}$
$\quad a_1 :: (f\ a_2) : L(\tau)$ ntp_lett

$\cdot \vdash^1 \lambda f{:}\tau \multimap L(\tau).\lambda h{:}\tau.\lambda a{:}\tau.\, \mathbf{let} \langle\!\langle a_1, a_2 \rangle\!\rangle = \mathsf{swap}\ a\ h\ \mathbf{in}$
$\quad a_1 :: (f\ a_2) : (\tau \multimap L(\tau)) \multimap \tau \multimap (\tau \multimap L(\tau))$ ntp_lam

$\ell : L(\tau) \vdash^1 \mathbf{iter}_{L(\tau), \tau \multimap L(\tau)}((\lambda a{:}\tau.[a]), \lambda f \cdots, \ell) : \tau \multimap L(\tau)$ ntp_iter

$\cdot \vdash^1 \mathsf{insert} : L(\tau) \multimap \tau \multimap L(\tau)$ ntp_lam

$\ell : L(\tau) \vdash^0 \mathbf{iter}_{L(\tau), L(\tau)}([], \mathsf{insert}, \ell) : L(\tau)$ ntp_iter

$\cdot \vdash^0 \mathsf{sort} : L(\tau) \multimap L(\tau)$ ntp_lam

2.3 Future Work

The above definition of e^* still critically lacks a proof of any connection between the reduction behavior of e and e^*, but it should not be difficult to produce one. Specifically, we expect some kind of logical relations argument would establish that the denotation of a term e of LinML would be equal to that of e^* up to the evident translation between LinML and $\mathsf{LinML}^\diamond$ of values of recursive datatypes.

References

[AS01] Klaus Aehlig and Helmut Schwichtenberg. A syntactical analysis of non-size-increasing polynomial time computation. 2001. Submitted.

[Hof00] Martin Hofmann. Linear types and non-size-increasing polynomial time computation. *Theoretical Computer Science*, 2000. To appear.