Linear Reduction

Ari Feiglin and Noam Kaplinski

In this paper we will define the concept of linear reduction in the context of syntax parsing. We will progress through more and more complicated examples, beginning from the programming of a simple calculator until we ultimately have created an extensible programming language.

Table of Contents

0 Notation **2**

1 Theoretical Background **3**

1.1 Stateless Reduction . . . 3
1.2 Stateful Reduction . . . 4
1.3 Valued Reduction . . . 5

0 Notation

$\mathbb{N}$ denotes the set of natural numbers, including 0.
$\overline{\mathbb{N}}$ is defined to be $\mathbb{N} \cup \{\infty\}$.
$f\colon A \longrightarrow B$ means that f is a partial function from A to B.
If X is a set and x is some symbol, then $\mathsf{X}_{\mathsf{x}} = \mathsf{X}^{\mathsf{x}} = \mathsf{X} \cup \{\mathsf{x}\}$.

1 Theoretical Background

1.1 Stateless Reduction

The idea of linear reduction is simple: given a string ξ the first character looks if it can bind with the second character to produce a new character, and the process repeats itself. There is of course, nuance. This nuance hides in the statement "if it can bind": we must define the rules for binding.

Let us define an *reducer* to be a tuple (Σ, β, π) where Σ is an alphabet; $\beta\colon \overline{\Sigma} \times \overline{\Sigma} \longrightarrow \overline{\Sigma}$ is a partial function called the *reduction function* where $\overline{\Sigma} = \Sigma \times \overline{\mathbb{N}}$; and π is the *initial priority function.* A *program* over an reducer is a string over $\overline{\Sigma}$. We write a program like $\sigma^1_{i_1} \cdots \sigma^n_{i_n}$ instead of as pairs $(\sigma^1, i_1) \ldots (\sigma^n, i_n)$. In the character σ_i, we call i the *priority* of σ.

Then the rules of reduction are as follows, meaning we define $\beta(\xi)$ for a program: We do so in cases:

(1) If $\xi = \sigma_i$ then $\beta(\xi) = \sigma_0$.

(2) If $\xi = \sigma^1_i \sigma^2_j \xi'$ where $i \geq j$ and $\beta(\sigma^1_i, \sigma^2_j) = \sigma^3_k$ is defined then $\beta(\xi) = \sigma^3_k \xi'$.

(3) Otherwise, for $\xi = \sigma^1_i \sigma^2_j \xi'$, $\beta(\xi) = \sigma^1_i \beta(\sigma^2_j \xi')$.

A string ξ such that $\beta(\xi) = \xi$ is called *irreducible.* Notice that it is possible for a string of length more than 1 to be irreducible: for example if $\beta(\sigma^1, \sigma^2)$ is not defined then $\sigma^1_i \sigma^2_j$ is irreducible.

$$\beta(\sigma_1 \tau_2) \xrightarrow{(3)} \sigma_1 \beta(\tau_2) \xrightarrow{(1)} \sigma_1 \tau_2$$

But such strings are not desired, since in the end we'd like a string to give us a value. So an irreducible string which is not a single character is called *ill-written*, and a string which is not ill-written is *well-written.*

Now the initial priority function is $\pi\colon \Sigma \longrightarrow \overline{\mathbb{N}}$ which gives characters their initial priority. We can then canonically extend this to a function $\pi\colon \Sigma^* \longrightarrow (\Sigma \times \overline{\mathbb{N}})^*$ defined by $\pi(\sigma^1 \cdots \sigma^n) = \sigma^1_{\pi(\sigma^1)} \cdots \sigma^n_{\pi(\sigma^n)}$. Then a β-reduction of a string $\xi \in \Sigma^*$ is taken to mean a β-reduction of $\pi(\xi)$.

Notice that once again we require that π only be a partial function. This is since that we don't always need every character in Σ to have an initial priority; some symbols are only given their priority through the β-reduction of another pair of symbols. So we now provide a new definition of a *program*, which is a string $\xi = \sigma^1 \cdots \sigma^n \in \Sigma^*$ such that $\pi(\sigma^i)$ exists for all $1 \leq i \leq n$. We can only of course discuss the reductions of programs, as $\pi(\xi)$ is only defined if ξ is a program.

Example: let $\Sigma = \mathbb{N} \cup \{+, \cdot\} \cup \{(n+), (n\cdot) \mid n \in \mathbb{N}\}$. β as follows:

σ^1_i, σ^2_j	$\beta(\sigma^1_i, \sigma^2_j)$
$n, +$	$(n+)$
$n, \cdot$	$(n\cdot)$
$(n+), m$	$n+m$
$(n\cdot), m$	$n \cdot m$
$(n\cdot), (m+)$	$(n \cdot m, +)$
$(n+), (m+)$	$(n+m, +)$
$(n\cdot), (m\cdot)$	$(n \cdot m, \cdot)$

Where n, m range over all values in $\mathbb{N}$. Here $\beta(\sigma_i, \sigma_j)$'s priority is j. We define the initial priorities

$$\pi(n) = \infty, \quad \pi(+) = 1, \quad \pi(\cdot) = 2$$

Now let us look at the string $1 + 2 \cdot 3 + 4$;. Here,

$$\begin{aligned}
1_\infty +_1 2_\infty \cdot_2 3_\infty +_1 4_\infty &\longrightarrow (1+)_1 2_\infty \cdot_2 3_\infty +_1 4_\infty \\
&\longrightarrow (1+)_1 (2\cdot)_2 3_\infty +_1 4_\infty \\
&\longrightarrow (1+)_1 (2\cdot)_2 (3+)_1 4_\infty \\
&\longrightarrow (1+)_1 (6+)_1 4_\infty \\
&\longrightarrow (7+)_1 4_\infty \\
&\longrightarrow (7+)_1 4_0 \\
&\longrightarrow (11)_0
\end{aligned}$$

So the rules for β we supplied seem to be sufficient for computing arithmetic expressions following the order of operations. $\diamondsuit$

Example: We can also expand our language to include parentheses. So our alphabet becomes $\Sigma = \mathbb{N} \cup \{+, \cdot, (,)\} \cup \left\{ \underline{n+}, \underline{n\cdot}, \underline{n)} \;\middle|\; n \in \mathbb{N} \right\}$. We distinguish between parentheses and bold parentheses for readability. We extend β as follows:

σ_i^1, σ_j^2	$\beta(\sigma_i^1, \sigma_j^2)$
$n, +$	$\underline{n+}_j$
$n, \cdot$	$\underline{n\cdot}_j$
$\underline{n+}, m$	$(n+m)_j$
$\underline{n\cdot}, m$	$(n \cdot m)_j$
$\underline{n\cdot}, \underline{m+}$	$\underline{n \cdot m, +}_j$
$\underline{n+}, \underline{m+}$	$\underline{n + m, +}_j$
$\underline{n\cdot}, \underline{m\cdot}$	$\underline{n \cdot m, \cdot}_j$
$n,)$	$\underline{n)}_j$
$\underline{n+}, \underline{m)}$	$\underline{n+m)}_j$
$\underline{n\cdot}, \underline{m)}$	$\underline{n \cdot m)}_j$
$(, \underline{n)}$	n_i

$(n+m)_j$ means $n+m$ with a priority of j, not $\underline{n+m}_j$. And we define the initial priorities

$$\pi(n) = \infty, \quad \pi(+) = 1, \quad \pi(\cdot) = 2, \quad \pi(() = \infty, \quad \pi()) = 0$$

So for example reducing $2 \cdot ((1+2) \cdot 2) + 1$,

$$\begin{aligned}
2_\infty *_2 (_\infty (_\infty 1_\infty +_1 2_\infty)_0 *_2 2_\infty)_0 +_1 1_\infty &\longrightarrow \underline{2*}_2 (_\infty (_\infty 1_\infty +_1 2_\infty)_0 *_2 2_\infty)_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty (_\infty \underline{1+}_1 2_\infty)_0 *_2 2_\infty)_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty (_\infty \underline{1+}_1 \underline{2)}_0 *_2 2_\infty)_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty (_\infty \underline{3)}_0 *_2 2_\infty)_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty 3_\infty *_2 2_\infty)_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty \underline{3*}_2 2_\infty)_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty \underline{3*}_2 \underline{2)}_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 (_\infty \underline{6)}_0 +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 6_\infty +_1 1_\infty \\
&\longrightarrow \underline{2*}_2 \underline{6+}_1 1_\infty \\
&\longrightarrow \underline{12+}_1 1_\infty \\
&\longrightarrow \underline{12+}_1 1_0 \\
&\longrightarrow 13_0
\end{aligned}$$

$\Diamond$

1.2 Stateful Reduction

Suppose we'd like to reduce a program with variables in it. Then we cannot just use the previous definitions, as the actions of σ (which is to be understood as the function $\beta(\sigma, \bullet)$) are determined before any reduction occurs. We need a way to store the value of variables, a state.

This leads us to the following definition: let Σ_P and Σ_A be two disjoint sets of symbols: Σ_P the set of *printable symbols* and Σ_A the set of *abstract symbols*. Σ_P will generally be a set consisting of the string representations of abstract symbols, be it operators like $+$ and $\cdot$ or variable names. Σ_A are the actual objects which can "execute something". Let us further define $\Sigma = \Sigma_P \cup \Sigma_A$.

Now a state is a mapping from printable symbols to strings. So for example, if x is a printable symbol a line like $\textsf{let}x = 1$ should change the state so that x maps to the abstract symbol representing 1.

A *point state* is a partial function $s \colon \Sigma_P \longrightarrow \Sigma_A$. If s_1, s_2 are point states, define their composition to be a point state $s_1 s_2$ such that

$$s_1 s_2(\sigma) = \begin{cases} s_2(\sigma) & \sigma \in \mathrm{dom}(s_2) \\ s_1(\sigma) & \sigma \in \mathrm{dom}(s_1) \end{cases}$$

A *state* is a sequence of point states: $\overline{s} = (s_1, \ldots, s_n)$. Let us define

$$\mathsf{State} = \{\Sigma_P \longrightarrow \Sigma_A\}^+$$

the set of all states.

Let $\overline{s} = (s_1 \cdots s_n) \in \mathsf{State}$ be a state, then define

- for $\sigma \in \Sigma_P$ we define $s(\sigma) = s_1 \cdots s_n(\sigma)$ (the composition of states),
- define *pop* $\overline{s} = (s_1, \ldots, s_{n-1})$,
- define *push* $\overline{s} = (s_1, \ldots, s_n, \varnothing)$ ($\varnothing$ is the empty state),
- if s is a point state, $\overline{s}s = (s_1, \ldots, s_{n-1}, s_n s)$,
- if s is a point state, $\overline{s} + s = (s_1, \ldots, s_n, s)$ (so *push* $\overline{s} = \overline{s} + \varnothing$).

So if we'd like to revert to a previous state, we simply pop from the current state. And substituting the current state only alters the current (topmost) point state.

Now we begin with an initial β function which is a partial function

$$\beta\colon \overline{\Sigma}_A \times (\overline{\Sigma} \cup \{\varepsilon\}) \times \mathsf{State} \longrightarrow \overline{\Sigma}^* \times \mathsf{State}$$

Recall that $\overline{\Sigma}$ is $\Sigma \times \mathbb{N}$. We will denote tuples in $X \times \mathsf{State}$ by $\langle x, \mid s\rangle$ for $x \in X$ and $s \in \mathsf{State}$ for the sake of readability. So we now wish to extend to a β function

$$\beta\colon \overline{\Sigma}^* \times \mathsf{State} \longrightarrow \overline{\Sigma}^* \times \mathsf{State}$$

We do this as follows: given $\xi \in (\Sigma \times \overline{\mathbb{N}})^*$ and $s \in \mathsf{State}$ we define $\beta\langle \xi \mid s\rangle$ as follows:

(1) if $\xi = \sigma_i \xi'$ for $\sigma \in \Sigma_P$ then $\beta\langle \xi \mid s\rangle = \langle s(\sigma)_i \xi' \mid s\rangle$,

(2) if $\xi = \sigma_i \xi'$ such that $\beta\langle \sigma_i \varepsilon \mid s\rangle = \langle \xi'' \mid s'\rangle$ is defined then $\beta\langle \xi \mid s\rangle = \langle \xi'' \xi' \mid s'\rangle$,

(3) if $\xi = \sigma^1_i \sigma^2_j \xi'$ for $\sigma^1 \in \Sigma_A$, $i \geq j$, such that $\beta\big\langle \sigma^1_i \sigma^2_j \bigm| s\big\rangle = \langle \xi'' \mid s'\rangle$ is defined, then $\beta\langle \xi \mid s\rangle = \langle \xi'' \xi' \mid s'\rangle$,

(4) otherwise for $\xi = \sigma^1_i \sigma^2_j \xi'$, if $\beta\big\langle \sigma^2_j \xi' \bigm| s\big\rangle = \langle \xi'' \mid s'\rangle$ then $\beta\langle \xi \mid s\rangle = \big\langle \sigma^1_i \xi'' \bigm| s'\big\rangle$.

Notice that **(2)** cares not about the priority of σ, and neither if $\beta(\sigma_i, \tau_j)$ is defined for some $\tau \neq \varepsilon$.

We also define the *initial priority function* to be a map $\pi\colon \Sigma_P \longrightarrow \overline{\mathbb{N}}$ (this is not a partial function: every printable symbol must be given a priority). This is once again canonically extended to a function $\pi\colon \Sigma_P^* \longrightarrow (\Sigma_P \times \overline{\mathbb{N}})^*$. And an *initial state* s_0 which is a point state. The quintuple $(\Sigma_P, \Sigma_A, \beta, \pi, s_0)$ is called an *reducer*. The reduction of a string $\xi \in S$ is the process of iteratively applying β to $\langle \pi(\xi) \mid s_0\rangle$.

Example: let

$$\Sigma_P = \mathbb{N} \cup \{+, \cdot, =, ;\} \cup \{\texttt{let}\} \cup \big\{x^i \bigm| i \in \mathbb{N}\big\},$$
$$\Sigma_A = \mathbb{N} \cup \{(n+), (n\cdot) \mid n \in \mathbb{N}\} \cup \{\texttt{let}\} \cup \big\{(\texttt{let}x^i), (\texttt{let}x^i =) \bigm| i \in \mathbb{N}\big\}$$

where the natural numbers in Σ_A are not the same as the natural numbers in Σ_P since they must be disjoint, same for $\texttt{let}$. But they both essentially represent the same thing: s_0 maps $n \mapsto n$ for $n \in \mathbb{N}$ (the left-hand n is in Σ_p, the right-hand n is in Σ_A) and $\texttt{let} \mapsto \texttt{let}$. All other printable symbols are mapped to ε.

And similar to the previous example we define $\pi(n) = \infty$, $\pi(+) = 1$, and $\pi(\cdot) = 2$. We extend this to $\pi(;) = 0$, $\pi(=) = 0$, $\pi(\texttt{let}) = \infty$, and $\pi(x^i) = \infty$.

Let us take the same transitions as the example in the previous section for $n, (n+), (n\cdot)$ (we have to add the condition that the state doesnt change). We further add the transitions

$\big\langle \sigma^1_i \sigma^2_j \bigm\vert s\big\rangle$	$\beta\big\langle \sigma^1 \sigma^2 \bigm\vert s\big\rangle$
$\langle \sigma; \mid s\rangle$	$\langle \sigma_j \mid s\rangle$
$\big\langle \mathsf{let}x^i \bigm\vert s\big\rangle$	$\big\langle (\mathsf{let}x^i)_j \bigm\vert s\big\rangle$
$\big\langle (\mathsf{let}x^i) = \bigm\vert s\big\rangle$	$\big\langle (\mathsf{let}x^i =)_j \bigm\vert s\big\rangle$
$\big\langle (\mathsf{let}x^i =)\sigma \bigm\vert s\big\rangle$	$\big\langle \varepsilon \bigm\vert s[x^i \mapsto \sigma]\big\rangle$

In the final transition, $n \in \Sigma_A$. Then for example (we will be skipping trivial reductions):

$$\begin{aligned}
\mathsf{let}x^1 = 1 + 2;\ \mathsf{let}x^2 = 2;\ x^1 \cdot x^2; &\longrightarrow \texttt{let}_\infty x^1_\infty =_0 1_\infty +_1 2_\infty ;_0\ \texttt{let}_\infty x^2_\infty =_0 2_\infty ;_0\ x^1_\infty \cdot_2 x^2_\infty ;_0 \\
s_0 &\longrightarrow (\texttt{let}x^1 =)_0 1_\infty +_1 2_\infty ;_0\ \texttt{let}_\infty x^2_\infty =_0 2_\infty ;_0\ x^1_\infty \cdot_2 x^2_\infty ;_0 \\
s_0 &\longrightarrow (\texttt{let}x^1 =)_0 3_0\ \texttt{let}_\infty x^2_\infty =_0 2_\infty ;_0\ x^1_\infty \cdot_2 x^2_\infty ;_0 \\
s_0[x^1 \mapsto 3] &\longrightarrow \texttt{let}_\infty x^2_\infty =_0 2_\infty ;_0\ x^1_\infty \cdot_2 x^2_\infty ;_0 \\
s_0[x^1 \mapsto 3,\ x^2 \mapsto 2] &\longrightarrow x^1_\infty \cdot_2 x^2_\infty ;_0 \\
s_0[x^1 \mapsto 3,\ x^2 \mapsto 2] &\longrightarrow 3_\infty \cdot_2 x^2_\infty ;_0 \\
s_0[x^1 \mapsto 3,\ x^2 \mapsto 2] &\longrightarrow (3\cdot)_2 x^2_\infty ;_0 \\
s_0[x^1 \mapsto 3,\ x^2 \mapsto 2] &\longrightarrow (3\cdot)_2 2_\infty ;_0 \\
s_0[x^1 \mapsto 3,\ x^2 \mapsto 2] &\longrightarrow 6_0
\end{aligned}$$

$\diamondsuit$

1.3 Valued Reduction

We define the following four base sets:

(**1**) $\mathcal{U}$ the universe of *values*, these are all the internal values an object may have.

(**2**) $\mathcal{T}_\mathcal{P}$ the set of *printable terms*, these are the tokens which a programmer may pass to the reducer.

(**3**) $\mathcal{T}_\Sigma$ the set of *type terms*.

(**4**) $\mathcal{T}_\mathcal{A}$ the set of *abstract terms*.

The sets $\mathcal{T}_\mathcal{P}, \mathcal{T}_\Sigma, \mathcal{T}_\mathcal{A}$ are all disjoint, we place no such restriction on $\mathcal{U}$ as the purpose it serves is different. Let $\mathcal{A}$ be a set of *atomic abstract terms*, then the construction of abstract terms is

$$\mathcal{T}_\mathcal{A} ::= \mathcal{A} \mid \mathcal{A}\mathcal{T}_\Sigma$$

And let Σ be a set of *atomic types*, each with an associated arity, which may be ∞. Let Σ^n be the set of atomic types of arity n, then the construction of type terms is

$$\mathcal{T}_\Sigma ::= \Sigma^0 \mid \Sigma^n \mathcal{T}_\Sigma^1 \cdots \mathcal{T}_\Sigma^n \mid \Sigma^\infty \mathcal{T}_\Sigma^1 \cdots \mathcal{T}_\Sigma^n$$

as n ranges over all $\mathbb{N}_{>0}$.

Define

(**1**) $\mathcal{T} := \mathcal{T}_\mathcal{P} \cup \mathcal{T}_\Sigma \cup \mathcal{T}_\mathcal{A}$ the set of *basic terms*.

(**2**) $\mathcal{T}_\mathcal{I} := \mathcal{T}_\Sigma \cup \mathcal{T}_\mathcal{A}$ the set of *internal terms*.

(**3**) $\Pi_\mathcal{I} := \mathcal{T}_\mathcal{I} \times \mathcal{U}$ the set of *termed values*.

(**4**) $\Pi := \Pi_\mathcal{I} \cup \mathcal{T}_\mathcal{P}$ the set of *atomic expressions*.

Elements of $\overline{\Pi}$ will be written like $\sigma_n(v)$ where σ is the term, n the priority, and v the value (nothing for printable terms).

In valued reduction, we abstract away some inputs to the initial beta-reducer in order to allow for easier implementation. An initial beta-reducer is a partial function

$$\widehat{\beta}\colon \mathcal{T}_\mathcal{I} \times \mathcal{T}^\varepsilon \longrightarrow \mathcal{T}_\mathcal{I}^\varepsilon \times (\overline{\mathbb{Z}} \times \overline{\mathbb{Z}} \to \overline{\mathbb{Z}}) \times (\mathcal{U} \times \mathcal{U} \times \text{State} \rightharpoonup \mathcal{U} \times \mathcal{T}_\mathcal{P}^* \times \text{State})$$

We extend this to a derived β-reducer,

$$\beta\colon \overline{\Pi}^* \times \text{State} \longrightarrow \overline{\Pi}^* \times \text{State}$$

with the following rules: given an input $\langle \xi \mid s\rangle$ its image is

(**1**) If $\xi = \sigma_n \xi'$ for $\sigma \in \mathcal{T}_\mathcal{P}$ then
$$\beta\langle \xi \mid s\rangle = \langle s(\sigma)_n \xi' \mid s\rangle.$$

(**2**) If $\xi = \sigma_i(v)\xi'$ and $\widehat{\beta}(\sigma, \varepsilon) = (\alpha, \rho, f)$ is defined, then if $f(v, \_, s) = (w, \zeta, s')$ and $\rho(i) = k$ then
$$\beta\langle \xi \mid s\rangle = \langle \alpha_k(w)\pi(\zeta)\xi' \mid s'\rangle.$$

(**3**) If $\xi = \sigma_i(v)\tau_j(u)\xi'$ and $i \geq j$ and $\widehat{\beta}(\sigma, \tau) = (\alpha, \rho, f)$ is defined, then if $f(v, u, s) = (w, \zeta, s')$ and $\rho(i, j) = k$ then
$$\beta\langle \xi \mid s\rangle = \langle \alpha_k(w)\pi(\zeta)\xi' \mid s\rangle.$$

(**4**) Otherwise, if $\xi = \sigma_i(v)\xi'$ and $\beta\langle \xi' \mid s\rangle = \langle \xi'' \mid s'\rangle$,
$$\beta\langle \xi \mid s\rangle = \langle \sigma_i(v)\xi'' \mid s'\rangle.$$

1.3.1 States

Similar to before, we define point-states as partial maps $\mathcal{T}_\mathcal{P} \longrightarrow \Pi_\mathcal{I}$. And if s_1, s_2 are two point-states and $\sigma \in \mathcal{T}_\mathcal{P}$ then

$$s_1 s_2(\sigma) = \begin{cases} s_2(\sigma) & \sigma \in \mathrm{dom} s_2 \\ s_1(\sigma) & \sigma \in \mathrm{dom} s_1 \end{cases}$$

We will denote finite point states as $[\sigma_1 \mapsto \varkappa_1, \ldots, \sigma_n \mapsto \varkappa_n]$, and this denotes the point-state which maps σ_i to $\varkappa_i$.

A state will now have two fields: a sequence of point-states, as well as a sequence of indexes. For a state $\bar{s} = \big[(s_1, \ldots, s_n), I = (i_1, \ldots, i_k)\big]$, let us define

(1) $\bar{s} + s = \big[(s_1, \ldots, s_n, s), I\big]$

(2) $\bar{s} +_c s = \big[(s_1, \ldots, s_n, s), (i_1, \ldots, i_k, n+1)\big]$

(3) $pop\ \bar{s} = \big[(s_1, \ldots, s_{n-1}), I\big]$ if $i_k < n$ otherwise, $\big[(s_1, \ldots, s_{n-1}), (i_1, \ldots, i_{k-1})\big]$

(4) $\bar{s}s = \big[(s_1, \ldots, s_n s), I\big]$

(5) $\bar{s}(\sigma) = s_1 \cdots s_n(\sigma)$ for $\sigma \in \Sigma_P$

(6) $\bar{s}_c = s_{i_k} \cdots s_n$

Furthermore, if $\sigma \in \mathcal{T}_\mathcal{P}$ and $\varkappa \in \Pi_\mathcal{I}$ let us define $\bar{s}\{\sigma \mapsto \varkappa\}$ as $(s_1, \ldots, s_i[\sigma \mapsto \varkappa], \ldots, s_n)$ where i is the maximum index such that $\sigma \in \mathrm{dom} s_i$.

1.3.2 The Initial Beta Reducer

We now describe the initial beta reducer. By convention, atomic abstract terms will be red, type terms will be green, internal terms will be blue.

End:

- σ end $\longrightarrow$ σ minfty $(u, \_, s \to u, \varepsilon, s)$

Arithmetic:

- σ op $\longrightarrow$ opσ snd $(u, f, s \to (u, f), \varepsilon, s)$
- opσ opσ $\longrightarrow$ opσ snd $\big((u, f), (v, g), s \to (f(u, v), g), \varepsilon, s\big)$
- opσ σ $\longrightarrow$ σ snd $\big((u, f), v, s \to f(u, v), \varepsilon, s\big)$
- σ rparen $\longrightarrow$ rparenσ snd $(u, \_, s \to u, \varepsilon, s)$
- opσ rparenσ $\longrightarrow$ rparenσ snd $\big((f, u), v, s \to f(u, v), \varepsilon, s\big)$
- lparen rparenσ $\longrightarrow$ σ fst $(\_, u, s \to u, \varepsilon, s)$

Lists:

- lbrack σ $\longrightarrow$ lbrackσ fst $(\_, u, s \to (u), \varepsilon, s)$
- lbrackσ σ $\longrightarrow$ lbrackσ fst $(\ell, u, s \to (\ell, u), \varepsilon, s)$
- lbrackσ rbrack $\longrightarrow$ listσ infty $(\ell, \_, s \to \ell, \varepsilon, s)$
- period num $\longrightarrow$ index zero $(\_, n, s \to n, \varepsilon, s)$
- listσ index $\longrightarrow$ σ fst $(\ell, i, s \to \ell_i, \varepsilon, s)$

Variables:

- let x $\longrightarrow$ letvar snd $(\_, \_, s \to (x, \varnothing), \varepsilon, s)$
- letvar index $\longrightarrow$ letvar fst $\big((x, \ell), n, s \to (x, (\ell, n)), \varepsilon, s\big)$
- letvar equal $\longrightarrow$ leteq minfty $\big((x, \ell), \_, s \to (x, \ell), \varepsilon, s\big)$
- leteq σ $\longrightarrow$ ε $\varnothing$ $\big((x, \ell), v, s \to \varepsilon, \varepsilon, s'\big)$ where s' is $s[x \mapsto \sigma(v)]$ if $\ell = \varnothing$ and otherwise let t be the result of setting $s(x).\ell_1.\ldots.\ell_n$ to v, then $s' = s[x \mapsto t]$.

Scoping:

- lbrace ε $\longrightarrow$ ε $\varnothing$ $(\_, \_, s \to \varepsilon, \varepsilon, s + \varnothing)$
- rbrace ε $\longrightarrow$ ε $\varnothing$ $(\_, \_, s \to \varepsilon, \varepsilon, pop\ s)$

Products:

- σ comma $\longrightarrow$ comma(σ) snd $(u, \_, s \to (u), \varepsilon, s)$
- opσ comma(σ) $\longrightarrow$ comma(σ) snd $((f, u), (v) \to (f(u, v)), \varepsilon, s)$
- commaΩ comma(σ) $\longrightarrow$ comma(Ω, σ) snd $(\ell, \ell', s \to (\ell, \ell'), \varepsilon, s)$

- $\mathsf{comma}\Omega$ $\mathsf{rparen}\sigma \longrightarrow \mathsf{listrparen}(\Omega, \sigma)$ snd $(\ell, v \rightarrow (\ell, v), \varepsilon, s)$
- lparen $\mathsf{listrparen}\Omega \longrightarrow \mathsf{product}\Omega$ infty $(\_, \ell, s \rightarrow \ell, \varepsilon, s)$

Primitives:

- $\mathsf{primitive}$ $\sigma \longrightarrow \varepsilon$ $\varnothing$ $(f, v, s \rightarrow \varepsilon, w, s)$ where $f(\sigma, v) = (w, s')$ (the purpose is for f to have a side effect)

Code Capture

- $\mathsf{lbrace}^{\mathsf{a}}$ $x \longrightarrow \mathsf{lbrace}^{\mathsf{a}}$ infty $(\xi, \_, s \rightarrow \xi x, \varepsilon, s)$ if $x \neq \{, \}$
- $\mathsf{lbrace}^{\mathsf{a}}$ $x \longrightarrow \mathsf{code}$ infty $(\xi, \_, s \rightarrow \xi, \varepsilon, s)$
- $\mathsf{lbrace}^{\mathsf{a}}$ $\mathsf{code} \longrightarrow \mathsf{lbrace}^{\mathsf{a}}$ infty $(\xi, \xi', s \rightarrow \xi\{\xi'\}, \varepsilon, s)$

Parameter Capture

- $\mathsf{lparen}^{\mathsf{a}}$ $x \longrightarrow \mathsf{lparen}^{\mathsf{a}}$ fst $(\ell, \_, s \rightarrow (\ell, x), \varepsilon, s)$ for $x \neq (,)$
- $\mathsf{lparen}^{\mathsf{a}}$ $) \longrightarrow \mathsf{plist}$ fst $(\ell, \_, s \rightarrow \ell, \varepsilon, s)$
- $\mathsf{lparen}^{\mathsf{a}}$ $\mathsf{plist} \longrightarrow \mathsf{lparen}^{\mathsf{a}}$ fst $(\ell, \ell', s \rightarrow (\ell, (\ell')), \varepsilon, s)$

Function Definitions

- fun $x \longrightarrow \mathsf{funname}$ infty $(\_, \_, s \rightarrow (x, \varepsilon), \varepsilon, s + [\{ \mapsto \mathsf{lbrace}^{\mathsf{a}}, \} \mapsto \mathsf{rbrace}^{\mathsf{a}}, (\mapsto \mathsf{lparen}^{\mathsf{a}},) \mapsto \mathsf{rparen}^{\mathsf{a}}])$
- $\mathsf{funname}$ $\mathsf{plist} \longrightarrow \mathsf{funvars}$ infty $\big((x, \varepsilon), u, s \rightarrow (x, u), \varepsilon, s\big)$
- $\mathsf{funvars}$ $\mathsf{code} \longrightarrow \mathsf{closure}$ fst $\big((x, \ell), \xi, s \rightarrow C = \langle \ell, \xi, s'[x \mapsto \mathsf{closure}(C)]\rangle, \varepsilon, \mathit{pop}\ s[x \mapsto \mathsf{closure}(C)]\big)$ where $s' = (\mathit{pop}\ s)_c$.

Function Calls

- $\mathsf{closure}$ $\sigma \longrightarrow \varepsilon$ $\varnothing$ $\big(\langle \ell, \xi, s\rangle, u \mapsto \varepsilon, \xi\}, s + [\ell \mapsto u]\big)$ where $\ell \mapsto u$ means that if $\ell = (x)$ then $x \mapsto u$. Otherwise $\ell = (x_1, \ldots, x_n)$ and $u = (u_1, \ldots, u_n)$ and $x_i \mapsto u_i$ (recursively).