

# 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

<b>0</b>	<b>Notation</b>	<b>2</b>
<b>1</b>	<b>Theoretical Background</b>	<b>3</b>
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  $\mathbb{N} \cup \{\infty\}$ .

$f: A \longrightarrow B$  means that  $f$  is a partial function from  $A$  to  $B$ .

If  $X$  is a set and  $x$  is some symbol, then  $X_x = X^x = X \cup \{x\}$ .

# 1 Theoretical Background

## 1.1 Stateless Reduction

The idea of linear reduction is simple: given a string  $\xi$  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  $(\Sigma, \beta, \pi)$  where  $\Sigma$  is an alphabet;  $\beta: \bar{\Sigma} \times \bar{\Sigma} \longrightarrow \bar{\Sigma}$  is a partial function called the *reduction function* where  $\bar{\Sigma} = \Sigma \times \bar{\mathbb{N}}$ ; and  $\pi$  is the *initial priority function*. A *program* over an reducer is a string over  $\bar{\Sigma}$ . We write a program like  $\sigma_{i_1}^1 \cdots \sigma_{i_n}^n$  instead of as pairs  $(\sigma^1, i_1) \dots (\sigma^n, i_n)$ . In the character  $\sigma_i$ , we call  $i$  the *priority* of  $\sigma$ .

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  $\xi = \sigma_i^1 \sigma_j^2 \xi'$  where  $i \geq j$  and  $\beta(\sigma_i^1, \sigma_j^2) = \sigma_k^3$  is defined then  $\beta(\xi) = \sigma_k^3 \xi'$ .
- (3) Otherwise, for  $\xi = \sigma_i^1 \sigma_j^2 \xi'$ ,  $\beta(\xi) = \sigma_i^1 \beta(\sigma_j^2 \xi')$ .

A string  $\xi$  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  $\sigma_i^1 \sigma_j^2$  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  $\pi: \Sigma \longrightarrow \bar{\mathbb{N}}$  which gives characters their initial priority. We can then canonically extend this to a function  $\pi: \Sigma^* \longrightarrow (\Sigma \times \bar{\mathbb{N}})^*$  defined by  $\pi(\sigma^1 \cdots \sigma^n) = \sigma_{\pi(\sigma^1)}^1 \cdots \sigma_{\pi(\sigma^n)}^n$ . Then a  $\beta$ -reduction of a string  $\xi \in \Sigma^*$  is taken to mean a  $\beta$ -reduction of  $\pi(\xi)$ .

Notice that once again we require that  $\pi$  only be a partial function. This is since that we don't always need every character in  $\Sigma$  to have an initial priority; some symbols are only given their priority through the  $\beta$ -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  $\xi$  is a program.

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

$\sigma_i^1, \sigma_j^2$	$\beta(\sigma_i^1, \sigma_j^2)$
$n, +$	$(n+)$
$n, \cdot$	$(n\cdot)$
$(n+), m$	$n + m$
$(n\cdot), m$	$n \cdot m$
$(n\cdot), (m+)$	$(n \cdot m, +)$
$(n+), (m+)$	$(n + m, +)$
$(n\cdot), (m\cdot)$	$(n \cdot m, \cdot)$

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  $\beta$  we supplied seem to be sufficient for computing arithmetic expressions following the order of operations.  $\diamond$

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

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

$(n + m)_j$  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  $2 \cdot ((1 + 2) \cdot 2) + 1$ ,

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

◇

## 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  $\sigma$  (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  $\Sigma_P$  and  $\Sigma_A$  be two disjoint sets of symbols:  $\Sigma_P$  the set of *printable symbols* and  $\Sigma_A$  the set of *abstract symbols*.  $\Sigma_P$  will generally be a set consisting of the string representations of abstract symbols, be it operators like  $+$  and  $\cdot$  or variable names.  $\Sigma_A$  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 **let**  $x = 1$  should change the state so that  $x$  maps to the abstract symbol representing 1.

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

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

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

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

the set of all states.

Let  $\bar{s} = (s_1 \cdots s_n) \in \mathbf{State}$  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  $\text{pop } \bar{s} = (s_1, \dots, s_{n-1})$ ,
- define  $\text{push } \bar{s} = (s_1, \dots, s_n, \emptyset)$  ( $\emptyset$  is the empty state),
- if  $s$  is a point state,  $\bar{s}s = (s_1, \dots, s_{n-1}, s_n s)$ ,
- if  $s$  is a point state,  $\bar{s} + s = (s_1, \dots, s_n, s)$  (so  $\text{push } \bar{s} = \bar{s} + \emptyset$ ).

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  $\beta$  function which is a partial function

$$\beta: \bar{\Sigma}_A \times (\bar{\Sigma} \cup \{\varepsilon\}) \times \mathbf{State} \longrightarrow \bar{\Sigma}^* \times \mathbf{State}$$

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

$$\beta: \bar{\Sigma}^* \times \mathbf{State} \longrightarrow \bar{\Sigma}^* \times \mathbf{State}$$

We do this as follows: given  $\xi \in (\Sigma \times \bar{\mathbb{N}})^*$  and  $s \in \mathbf{State}$  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  $\xi = \sigma_i^1 \sigma_j^2 \xi'$  for  $\sigma^1 \in \Sigma_A$ ,  $i \geq j$ , such that  $\beta\langle \sigma_i^1 \sigma_j^2 \mid s \rangle = \langle \xi'' \mid s' \rangle$  is defined, then  $\beta\langle \xi \mid s \rangle = \langle \xi'' \xi' \mid s' \rangle$ ,
- (4) otherwise for  $\xi = \sigma_i^1 \sigma_j^2 \xi'$ , if  $\beta\langle \sigma_j^2 \xi' \mid s \rangle = \langle \xi'' \mid s' \rangle$  then  $\beta\langle \xi \mid s \rangle = \langle \sigma_i^1 \xi'' \mid s' \rangle$ .

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

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

**Example:** let

$$\begin{aligned} \Sigma_P &= \mathbb{N} \cup \{+, \cdot, =, ;\} \cup \{\text{let}\} \cup \{x^i \mid i \in \mathbb{N}\}, \\ \Sigma_A &= \mathbb{N} \cup \{(n+), (n\cdot) \mid n \in \mathbb{N}\} \cup \{\text{let}\} \cup \{(\text{let}x^i), (\text{let}x^i =) \mid i \in \mathbb{N}\} \end{aligned}$$

where the natural numbers in  $\Sigma_A$  are not the same as the natural numbers in  $\Sigma_P$  since they must be disjoint, same for **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  $\Sigma_P$ , the right-hand  $n$  is in  $\Sigma_A$ ) and **let**  $\mapsto$  **let**. All other printable symbols are mapped to  $\varepsilon$ .

And similar to the previous example we define  $\pi(n) = \infty$ ,  $\pi(+)=1$ , and  $\pi(\cdot)=2$ . We extend this to  $\pi(;)=0$ ,  $\pi(=)=0$ ,  $\pi(\text{let})=\infty$ , 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 doesn't change). We further add the transitions

$$\frac{\langle \sigma_i^1 \sigma_j^2 \mid s \rangle \quad \beta\langle \sigma^1 \sigma^2 \mid s \rangle}{\begin{array}{cc} \langle \sigma; \mid s \rangle & \langle \sigma_j \mid s \rangle \\ \langle \text{let}x^i \mid s \rangle & \langle (\text{let}x^i)_j \mid s \rangle \\ \langle (\text{let}x^i) = \mid s \rangle & \langle (\text{let}x^i =)_j \mid s \rangle \\ \langle (\text{let}x^i =) \sigma \mid s \rangle & \langle \varepsilon \mid s[x^i \mapsto \sigma] \rangle \end{array}}$$

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

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

### 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

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

And let  $\Sigma$  be a set of *atomic types*, each with an associated arity, which may be  $\infty$ . Let  $\Sigma^n$  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  $\bar{\Pi}$  will be written like  $\sigma_n(v)$  where  $\sigma$  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

$$\hat{\beta}: \mathcal{T}_{\mathcal{I}} \times \mathcal{T}^{\varepsilon} \longrightarrow \mathcal{T}_{\mathcal{I}}^{\varepsilon} \times (\bar{\mathbb{Z}} \times \bar{\mathbb{Z}} \rightarrow \bar{\mathbb{Z}}) \times (\mathcal{U} \times \mathcal{U} \times \text{State} \rightarrow \mathcal{U} \times \mathcal{T}_{\mathcal{P}}^* \times \text{State})$$

We extend this to a derived  $\beta$ -reducer,

$$\beta: \bar{\Pi}^* \times \text{State} \longrightarrow \bar{\Pi}^* \times \text{State}$$

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  $\hat{\beta}(\sigma, \varepsilon) = (\alpha, \rho, f)$  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  $\hat{\beta}(\sigma, \tau) = (\alpha, \rho, f)$  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

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

We will denote finite point states as  $[\sigma_1 \mapsto \varkappa_1, \dots, \sigma_n \mapsto \varkappa_n]$ , and this denotes the point-state which maps  $\sigma_i$  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  $\bar{s} = [(s_1, \dots, s_n), I = (i_1, \dots, i_k)]$ , let us define

- (1)  $\bar{s} + s = [(s_1, \dots, s_n, s), I]$
- (2)  $\bar{s} +_c s = [(s_1, \dots, s_n, s), (i_1, \dots, i_k, n + 1)]$
- (3)  $\text{pop } \bar{s} = [(s_1, \dots, s_{n-1}), I]$  if  $i_k < n$  otherwise,  $[(s_1, \dots, s_{n-1}), (i_1, \dots, i_{k-1})]$
- (4)  $\bar{s}s = [(s_1, \dots, s_n s), I]$
- (5)  $\bar{s}(\sigma) = s_1 \cdots s_n(\sigma)$  for  $\sigma \in \Sigma_P$

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

### 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:**

- $\sigma \text{ end} \longrightarrow \sigma \text{ minfty } (u, \_, s \rightarrow u, \varepsilon, s)$

**Arithmetic:**

- $\sigma \text{ op} \longrightarrow \text{op}\sigma \text{ snd } (u, f, s \rightarrow (u, f), \varepsilon, s)$
- $\text{op}\sigma \text{ op}\sigma \longrightarrow \text{op}\sigma \text{ snd } ((u, f), (v, g), s \rightarrow (f(u, v), g), \varepsilon, s)$
- $\text{op}\sigma \sigma \longrightarrow \sigma \text{ snd } ((u, f), v, s \rightarrow f(u, v), \varepsilon, s)$
- $\sigma \text{ rparen} \longrightarrow \text{rparen}\sigma \text{ snd } (u, \_, s \rightarrow u, \varepsilon, s)$
- $\text{op}\sigma \text{ rparen}\sigma \longrightarrow \text{rparen}\sigma \text{ snd } ((f, u), v, s \rightarrow f(u, v), \varepsilon, s)$
- $\text{lparen rparen}\sigma \longrightarrow \sigma \text{ fst } (\_, u, s \rightarrow u, \varepsilon, s)$

**Lists:**

- $\text{lbrack } \sigma \longrightarrow \text{lbrack}\sigma \text{ fst } (\_, u, s \rightarrow (u), \varepsilon, s)$
- $\text{lbrack}\sigma \sigma \longrightarrow \text{lbrack}\sigma \text{ fst } (\ell, u, s \rightarrow (\ell, u), \varepsilon, s)$
- $\text{lbrack}\sigma \text{ rbrack} \longrightarrow \text{list}\sigma \text{ infty } (\ell, \_, s \rightarrow \ell, \varepsilon, s)$
- $\text{period num} \longrightarrow \text{index zero } (\_, n, s \rightarrow n, \varepsilon, s)$
- $\text{list}\sigma \text{ index} \longrightarrow \sigma \text{ fst } (\ell, i, s \rightarrow \ell_i, \varepsilon, s)$

**Variables:**

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

**Scoping:**

- $\text{lbrace } \varepsilon \longrightarrow \varepsilon \emptyset (\_, \_, s \rightarrow \varepsilon, \varepsilon, s + \emptyset)$
- $\text{rbrace } \varepsilon \longrightarrow \varepsilon \emptyset (\_, \_, s \rightarrow \varepsilon, \varepsilon, \text{pop } s)$