# Linear Reduction

In this paper I 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

$\mathbb{N}$ denotes the set of natural numbers, including 0.

$\overline{\mathbb{N}}$ is defined to be $\mathbb{N} \cup \{\infty\}$.

$f\colon A \hookrightarrow B$ means that $f$ is a partial function from $A$ to $B$.

# 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; $\beta\colon \overline{\Sigma} \times \overline{\Sigma} \hookrightarrow \overline{\Sigma}$ is a partial function called the *reduction function* where $\overline{\Sigma} = \Sigma \times \overline{\mathbb{N}}$; and $\pi$ 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 $(\sigma^1, i_1) \dots (\sigma^n, i_n)$. 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 $\xi = \sigma^1_i \sigma^2_j \xi'$ 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 $\xi$ 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 $\sigma^1_i \sigma^2_j$ 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 $\pi\colon \Sigma \hookrightarrow \overline{\mathbb{N}}$ which gives characters their initial priority. We can then canonically extend this to a function $\pi\colon \Sigma^* \hookrightarrow (\Sigma \times \overline{\mathbb{N}})^*$ defined by $\pi(\sigma^1 \cdots \sigma^n) = \sigma^1_{\pi(\sigma^1)} \cdots \sigma^n_{\pi(\sigma^n)}$. Then a $\beta$-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:

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

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 $\beta$ 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 $\Sigma = \mathbb{N} \cup \{+, \cdot, (, )\} \cup \left\{ \underline{n+}, \underline{n\cdot}, \underline{n)} \,\middle|\, n \in \mathbb{N} \right\}$. We distinguish between parentheses and bold parentheses for readability. We extend $\beta$ 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 $2 \cdot ((1+2) \cdot 2) + 1$,

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

$\diamondsuit$

## 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. So we can view a state as a map from $\Sigma_P \longrightarrow \Sigma^*$, where if the map of a symbol is $\varepsilon$ (the empty word), this represents it not having a value. But in order to implement locality, we must be able to revert to a previous state. Thus a state will actually be a (non-empty) sequence of maps $\Sigma_P \longrightarrow \Sigma^*$. Maps $\Sigma_P \longrightarrow \Sigma^*$ are called *point states*. A *substitution* is a partial map $\nu\colon \Sigma_P \hookrightarrow \Sigma^*$ which represents changing the values in its domain to their new values in the substitution. For a point-state $\hat{s}$ define $\hat{s}\nu$ by $\hat{s}\nu(\sigma) = \nu(\sigma)$ if $\sigma$ is in $\nu$'s domain and $\hat{s}(\sigma)$ otherwise.

Substitutions are often written as $[\sigma_1 \mapsto v_1, \ldots, \sigma_n \mapsto v_n]$ which is to be understood as the partial function which maps $\sigma_i$ to $v_i$.

Let us define

$$\mathsf{State} = \{\Sigma_P \longrightarrow \Sigma^*\}^+$$

the set of all finite non-empty sequences of maps.

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

- for $\sigma \in \Sigma_P$ we define $s(\sigma) = s_n(\sigma)$,
- define *pop* $s = s_1 \cdots s_{n-1}$ (if $n > 1$),
- define *push* $s = s_1 \cdots s_n s_n$,
- if $\nu$ is a substitution, define $s\nu$ to be $s_1 \cdots s_{n-1}(s_n\nu)$.

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

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

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

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

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

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

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

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

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

Notice that **(2)** cares not about the priority of $\sigma$, 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 $\beta$ to $\langle \pi(\xi), s_0\rangle$.

**Example:** let

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

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 $\texttt{let} \mapsto \texttt{let}$. 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 $\pi(;) = 0$, $\pi(=) = 0$, $\pi(\texttt{let}) = \infty$, 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

| $\big\langle \sigma^1_i \sigma^2_j, s\big\rangle$ | $\beta\big\langle \sigma^1 \sigma^2, s\big\rangle$ |
|---|---|
| $\langle \sigma;, s\rangle$ | $\langle \sigma_j, s\rangle$ |
| $\big\langle \texttt{let}\ x^i, s\big\rangle$ | $\big\langle (\texttt{let}\ x^i)_j, s\big\rangle$ |
| $\big\langle (\texttt{let}\ x^i) =, s\big\rangle$ | $\big\langle (\texttt{let}\ x^i =)_j, s\big\rangle$ |
| $\big\langle (\texttt{let}\ x^i =)\sigma, s\big\rangle$ | $\big\langle \varepsilon, s[x^i \mapsto \sigma]\big\rangle$ |

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

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

$\Diamond$

**Example:** Why does the initial $\beta$ function map to strings $\overline{\Sigma}^* \times \mathsf{State}$ and not $\Sigma \times \mathsf{State}$? This is in order to implement functions; let us analyze the following example:

$$\begin{aligned}
\Sigma_P &= \mathbb{N} \cup \{+, \cdot, =, (, ), \{, \}, ,, ;, \texttt{let}, \texttt{fun}\} \cup \left\{x^i \;\middle|\; i \in \mathbb{N}\right\} \\
\Sigma_A &= \underline{\Sigma}_P^* \cup \left\{\underline{\{}', \underline{\}}'\right\}
\end{aligned}$$

We will denote strings in $\Sigma_A$ with an underline to distinguish them from elements of $\Sigma_P$ (hence the $\underline{\Sigma}_P^*$ in $\Sigma_A$). This is an extension of the previous examples, so the initial $\beta$ function acts the same on these characters, all we must do is add what happens to `fun` and the other characters we added.

The reason that we add the symbols $\underline{\{}'$ and $\underline{\}}'$ even though $\underline{\{}$ and $\underline{\}}$ exist already in $\underline{\Sigma}_P$ is because curly braces have two different yet related behaviors. The first is to begin and end scopes: $\{\ldots\}$ will start a new scope, then execute $\ldots$, then end the scope. The second is in code of the form $\texttt{fun}(\ldots)\{\ldots\}$, where we would like $\{\ldots\}$ to accumulate all the code in $\ldots$.

$$\begin{array}{ccl}
\left\langle \sigma^1_i \sigma^2_j, s \right\rangle & \beta\left\langle \sigma^1 \sigma^2, s \right\rangle & \\
\hline
\left\langle \underline{\texttt{fun}}_i (_j, s \right\rangle & \left\langle \underline{\texttt{fun}(}_j, s[\{ \mapsto \underline{\{}',\ \} \mapsto \underline{\}}'] \right\rangle & \\
\left\langle \underline{\texttt{fun}\xi}_i \sigma_j, s \right\rangle & \left\langle \underline{\texttt{fun}\xi\sigma}_j, s \right\rangle & \text{if } \xi \text{ doesn't end with )} \\
\left\langle \underline{\texttt{fun}\xi}_i\ \underline{\{\xi\}}, s \right\rangle & \left\langle \underline{\texttt{fun}\xi\{\xi'\}}, s \right\rangle & \\
\left\langle \underline{\texttt{fun}(x_1, \ldots, x_n)\{\xi\}}_i\ \underline{(v_1, \ldots, v_n)}_j, s \right\rangle & \langle \pi(\xi\}), (push\ s)[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n] \rangle & \\
\left\langle \underline{\{}_i \varepsilon, s \right\rangle & \langle \varepsilon, push\ s \rangle & \\
\left\langle \underline{\}}_i \varepsilon, s \right\rangle & \langle \varepsilon, pop\ s \rangle & \\
\left\langle \underline{\{'\xi}_i \sigma_j, s \right\rangle & \left\langle \underline{\{'\xi\sigma}_i, s \right\rangle & \text{for } \sigma \in \Sigma_P \text{ not } \{ \\
\left\langle \underline{\{'\xi}_i\ \underline{\{\xi'\}}_j, s \right\rangle & \left\langle \underline{\{'\xi\{\xi'\}}_i, s \right\rangle & \\
\left\langle \underline{\{'\xi}_i\ \underline{\}}', s \right\rangle & \left\langle \underline{\{\xi\}}, s \right\rangle &
\end{array}$$