Linear Expansion

Ari Feiglin

In this paper I will define the concept of linear expansion 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

1 Theoretical Background

The idea of linear expansion 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 *expander* to be a tuple (Σ, β) where Σ is an alphabet and $\beta\colon \Sigma\times\Sigma \hookrightarrow \Sigma\cup\overline{\mathbb{N}}$ is the *reduction function* where $\overline{\mathbb{N}} := \mathbb{N}\cup\{\infty\}$. A *program* over an expander is a string over $\Sigma\times\overline{\mathbb{N}}$. 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 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 $\beta(\sigma) = \sigma_0$.

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

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

Notice that β cares not about the priorities of its inputs, otherwise it would be a much more complicated function.

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*.

Let us give an 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	$\beta(\sigma_1, \sigma_2)$
$n, +$	$(n+)$
$n, \cdot$	$(n\cdot)$
$(n+), m$	$n+m$
$(n\cdot), m$	$n\cdot m$
$(n\cdot), (m+)$	$(n\cdot m, +)$
$(n+), (m+)$	$(n+m, +)$
$(n\cdot), (m\cdot)$	$(n\cdot m, \cdot)$

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

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.

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

σ^1_i, σ^2_j	$\beta(\sigma_1, \sigma_2)$
$n, +$	$\boldsymbol{(}n+\boldsymbol{)}_j$
$n, \cdot$	$\boldsymbol{(}n\cdot\boldsymbol{)}_j$
$\boldsymbol{(}n+\boldsymbol{)}, m$	$(n+m)_j$
$\boldsymbol{(}n\cdot\boldsymbol{)}, m$	$(n\cdot m)_j$
$\boldsymbol{(}n\cdot\boldsymbol{)}, \boldsymbol{(}m+\boldsymbol{)}$	$\boldsymbol{(}n\cdot m, +\boldsymbol{)}_j$
$\boldsymbol{(}n+\boldsymbol{)}, \boldsymbol{(}m+\boldsymbol{)}$	$\boldsymbol{(}n+m, +\boldsymbol{)}_j$
$\boldsymbol{(}n\cdot\boldsymbol{)}, \boldsymbol{(}m\cdot\boldsymbol{)}$	$\boldsymbol{(}n\cdot m, \cdot\boldsymbol{)}_j$
$n,)$	$\boldsymbol{(}n)\boldsymbol{)}_j$
$\boldsymbol{(}n+\boldsymbol{)}, \boldsymbol{(}m)\boldsymbol{)}$	$\boldsymbol{(}n+m)\boldsymbol{)}_j$
$\boldsymbol{(}n\cdot\boldsymbol{)}, \boldsymbol{(}m)\boldsymbol{)}$	$\boldsymbol{(}n\cdot m)\boldsymbol{)}_j$
$(, \boldsymbol{(}n)\boldsymbol{)}$	n_i

$(n+m)_j$ means $n+m$ with a priority of j, not $\left(n+m\right)_j$. So for example expanding $2\cdot((1+2)\cdot 2)+1$,

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