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

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

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

(3) $\mathbb{Z}$ denotes the set of integers.

(4) $\overline{\mathbb{Z}}$ is defined to be $\mathbb{Z} \cup \{\pm\infty\}$.

(5) $f\colon A \rightharpoonup B$ means that f is a partial function from A to B.

(6) If X is a set and x is some symbol, then $\mathsf{X}_\mathsf{x} = \mathsf{X}^\mathsf{x} = \mathsf{X} \cup \{\mathsf{x}\}$.

(7) ε is the empty string, it and $\varnothing$ are also used to denote "nothing" in whatever context that may be.

(8) $(x_1, \ldots, x_n)$ denotes a list.

(9) If ℓ_1, ℓ_2 are lists, $\ell_1@\ell_2$ is their concatenation.

(10) $t{::}\ell$ is the list whose first element is t and whose tail is ℓ.

1 The Algorithm

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 $(\sigma^1, i_1) \dots (\sigma^n, i_n)$. 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 $\beta(\xi) = \sigma_0$.

(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. ♢

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

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).

We define the *initial priority function* as a function $\pi\colon \mathcal{T}_{\mathcal{P}} \longrightarrow \overline{\mathbb{Z}}$. This can be extended canonically to a function $\pi\colon \mathcal{T}_{\mathcal{P}}^* \longrightarrow \overline{\Pi}^*$.

In stateful 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

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

We extend this to a derived β-reducer,

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

We also define β^* where given an input $\langle \xi \mid s\rangle$, it runs β on it iteratively until convergence (of ξ). β is defined 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 $\rho(i) = k$ and $\beta^*\langle \pi\zeta \mid s'\rangle = \langle \zeta' \mid s''\rangle$ then
$$\beta\langle \xi \mid s\rangle = \langle \alpha_k(w)\zeta'\xi' \mid s''\rangle.$$

(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 $f(v, u, s) = (w, \zeta, s')$, $\rho(i, j) = k$, and $\beta^*\langle \pi\zeta \mid s'\rangle = \langle \zeta' \mid s''\rangle$ then
$$\beta\langle \xi \mid s\rangle = \langle \alpha_k(w)\zeta'\xi' \mid s''\rangle.$$

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

1.2.1 States

Similar to before, we define point-states as partial maps $\mathcal{T}_{\mathcal{P}} \rightharpoonup \Pi_{\mathcal{I}}$. And if s_1, s_2 are two point-states and $\sigma \in \mathcal{T}_{\mathcal{P}}$ then
$$s_1 s_2(\sigma) = \begin{cases} s_2(\sigma) & \sigma \in \operatorname{dom} s_2 \\ s_1(\sigma) & \sigma \in \operatorname{dom} s_1 \end{cases}$$

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 $\sigma \in \operatorname{dom} s_i$.

2 The Grammar

In this section we discuss the grammar of the language. This is not naturally imposed by the parser, but it will properly parse programs of this form.

Identifiers

$$
\begin{aligned}
\mathit{str} &::= (a \ldots z \mid A \ldots Z \mid \_) \\
\mathit{digit} &::= (0 \ldots 9) \\
\mathit{ident} &::= \mathit{str}\ (\mathit{str} \mid \mathit{digit})^*
\end{aligned}
$$

Constant Expressions

$$
\begin{aligned}
\mathit{const} &::= (\mathit{number} \mid \mathit{product} \mid \mathit{list}) \\
\mathit{number} &::= (\mathit{digit})^*[.(\mathit{digit})^*] \\
\mathit{product} &:= (\mathit{production}\ (, \mathit{production})^*) \\
\mathit{production} &:= (\mathit{expr} \mid \mathit{product})
\end{aligned}
$$

Expressions

$$
\begin{aligned}
\mathit{op} &::= + \mid * \mid / \\
\mathit{pop} &::= - \\
\mathit{expr} &::= \ \ \mathit{ident} \\
&\quad \mid \mathit{const} \\
&\quad \mid \mathit{expr}; \\
&\quad \mid \mathit{expr}\ \mathit{expr} \\
&\quad \mid \mathit{primexpr} \\
&\quad \mid (\mathit{expr}) \\
&\quad \mid \mathit{expr}\ (\mathit{op} \mid \mathit{pop})\ \mathit{expr} \\
&\quad \mid \mathit{pop}\ \mathit{expr} \\
&\quad \mid \mathit{expr}.\mathit{expr} \\
&\quad \mid \textbf{if}\ (\mathit{expr})\ \{\mathit{expr}\}\{\mathit{expr}\} \\
&\quad \mid \textbf{fun}\ \mathit{ident}\ (\mathit{pattern})\ \{\mathit{expr}\} \\
\mathit{primexpr} &::= \_\textbf{prim}\_\mathit{ident}
\end{aligned}
$$

3 Initializing the Algorithm

3.1 The Initial Beta Reducer

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

End:

- σ end $\longrightarrow \sigma$ 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 $\longrightarrow \sigma$ snd $\big((u, f), v, s \to f(u, v), \varepsilon, s\big)$
- pop σ $\longrightarrow \sigma$ snd $\big((f, g), u, s \to f(u), \varepsilon, s\big)$
- σ pop $\longrightarrow$ opσ one $\big(u, (f, g), s \to (u, g), \varepsilon, s\big)$
- σ rparen $\longrightarrow$ rparenσ snd $(u, \_, s \to u, \varepsilon, s)$
- opσ rparenσ $\longrightarrow$ rparenσ snd $\big((f, u), v, s \to f(u, v), \varepsilon, s\big)$
- pop rparenσ $\longrightarrow$ rparenσ snd $\big((f, g), u, s \to f(u), \varepsilon, s\big)$
- lparen rparenσ $\longrightarrow \sigma$ 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 $\longrightarrow \sigma$ 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 $\longrightarrow \varepsilon\ \varnothing$ $\big((x, \ell), v, s \to \varepsilon, \varepsilon, s'\big)$ where s' is $s[x \mapsto \sigma(v)]$ if $\ell = \varnothing$ and otherwise let t be the result of setting $s(x).\ell_1.\dots.\ell_n$ to v, then $s' = s[x \mapsto t]$.

Scoping:

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

Products:

- σ comma $\longrightarrow$ comma(σ) snd $(u, \_, s \to (u), \varepsilon, s)$
- opσ comma(σ) $\longrightarrow$ comma(σ) snd $((f, u), (v) \to (f(u, v)), \varepsilon, s)$
- pop comma(σ) $\longrightarrow$ comma(σ) snd $((f, g), (u) \to (f(u), \varepsilon, s))$
- commaΩ comma(σ) $\longrightarrow$ comma(Ω, σ) snd $(\ell, \ell', s \to (\ell, \ell'), \varepsilon, s)$
- commaΩ rparenσ $\longrightarrow$ listrparen(Ω, σ) snd $(\ell, v \to (\ell, v), \varepsilon, s)$
- lparen listrparenΩ $\longrightarrow$ productΩ infty $(\_, \ell, s \to \ell, \varepsilon, s)$

Primitives:

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

Code Capture

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

Parameter Capture

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

Function Definitions

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

Function Calls

- closure $\sigma \longrightarrow \varepsilon\ \varnothing\ \big(\langle \ell, \xi, ps\rangle, u, s \mapsto \varepsilon, \xi\}, s +_c ps[\ell \mapsto \sigma(u)]\big)$ where $\ell \mapsto \sigma(u)$ means that if $\ell = (x)$ then $x \mapsto \sigma(u)$. Otherwise $\ell = (x_1, \ldots, x_n)$, $\sigma = \mathsf{product}\sigma_1 \cdots \sigma_n$, and $u = (u_1, \ldots, u_n)$ and $x_i \mapsto \sigma_i(u_i)$ (recursively).

If Statements

- if $\sigma \longrightarrow$ ifbool fst $(\_, n, s \to n, \varepsilon, s + [\{\mapsto \mathsf{lbrace}^a, (\mapsto \mathsf{lparen}^a)]$
- ifbool code $\longrightarrow$ ifthen fst $(n, \xi, s \to (n, \xi), \varepsilon, s)$
- ifthen code $\longrightarrow \varepsilon\ \_\ ((n, \xi_1), \xi_2, s \to \varnothing, (n = 0?\ \xi_2 : \xi_1), pop\ s)$

Types

- typeσ type$\tau \longrightarrow$ type$\sigma(\tau)$ snd $(\_, \_, s \to \sigma(\tau), \varepsilon, s)$
- typeσ product(type$\tau_1, \ldots,$ type$\tau_n) \longrightarrow$ type$\sigma(\tau_1, \ldots, \tau_n)$ snd $(\_, \_, s \to \sigma(\tau_1, \ldots, \tau_n), \varepsilon, s)$
- colon type$\sigma \longrightarrow$ typerσ snd $(\_, u, s \to u, \varepsilon, s)$
- σ typer$\tau \longrightarrow \tau$ snd $(u, \_, s \to u, \varepsilon, s)$

3.2 The Initial State

The initial state is a partial state, defined as follows:

End

- $; \mapsto (\mathsf{end}, \varnothing)$

Arithmetic

- $(\mapsto (\mathsf{lparen}, \varnothing)$
- $) \mapsto (\mathsf{rparen}, \varnothing)$
- $+ \mapsto (\mathsf{op}, (n, m \to n + m))$
- $+ \mapsto (\mathsf{op}, (n, m \to n + m))$
- $* \mapsto (\mathsf{op}, (n, m \to n * m))$
- $/ \mapsto (\mathsf{op}, (n, m \to n/m))$
- $- \mapsto (\mathsf{pop}, (n \to -n), (n, m \to n - m))$
- $@ \mapsto (\mathsf{op}, (\ell_1, \ell_2 \to \ell_1 @ \ell_2))$
- $!= \ \mapsto (\mathsf{op}, (u, v \to u \neq v))$
- $<= \ \mapsto (\mathsf{op}, (n, m \to n \leq m))$
- $>= \ \mapsto (\mathsf{op}, (n, m \to n \geq m))$
- $== \ \mapsto (\mathsf{op}, (u, v \to u = v))$
- $< \ \mapsto (\mathsf{op}, (n, m \to n < m))$
- $> \ \mapsto (\mathsf{op}, (n, m \to n > m))$

Lists

- $[\mapsto (\mathsf{lbrack}, [])$
- $] \mapsto (\mathsf{rbrack}, \varnothing)$
- $. \mapsto (\mathsf{period}, \varnothing)$

Variables

- let $\mapsto (\mathsf{let}, \varnothing)$
- $= \ \mapsto (\mathsf{equal}, \varnothing)$

Scoping

- $\{ \mapsto (\mathsf{lbrace}, \varnothing)$
- $\} \mapsto (\mathsf{rbrace}, \varnothing)$

Products

- $, \mapsto (\mathsf{comma}, \varnothing)$

Primitives

- $\_\mathrm{prim\_print} \mapsto$ $(\textsf{primitive}, (a, v \to \mathrm{print}(v); (\varnothing, \varnothing)))$
- $\_\mathrm{prim\_len} \mapsto (\textsf{primitive}, (a, \ell \to \textsf{num}, |\ell|))$
- $\_\mathrm{prim\_tail} \mapsto (\textsf{primitive}, (\sigma, t :: \ell \to \sigma, \ell))$
- $\_\mathrm{prim\_type} \mapsto (\textsf{primitive}, (\sigma, \_ \to \textsf{type}\sigma, \sigma))$

Keywords

- $\mathrm{fun} \mapsto (\textsf{fun}, \varnothing)$
- $\mathrm{if} \mapsto (\textsf{if}, \varnothing)$

Types

- $: \ \mapsto (:, \varnothing)$
- $\mathrm{Num} \mapsto (\textsf{type num}, \textsf{num})$
- $\mathrm{List} \mapsto (\textsf{type list}, \textsf{list})$
- $\mathrm{Closure} \mapsto (\textsf{type closure}, \textsf{closure})$
- $\mathrm{Product} \mapsto (\textsf{type product}, \textsf{product})$
- $\mathrm{Primitive} \mapsto (\textsf{type primitive}, \textsf{primitive})$
- $\mathrm{Type} \mapsto (\textsf{type type}, \textsf{type})$

3.3 The Initial Priorities

The initial priority function, π, is defined as follows:

End

- $; \ \mapsto -\infty$

Arithmetic

- $(\ \mapsto \infty$
- $) \mapsto 0$
- $+ \mapsto 1$
- $* \mapsto 2$
- $- \mapsto 1$
- $/ \mapsto 2$
- $@ \mapsto 1$
- $== \ \mapsto 0$
- $!= \ \mapsto 0$
- $<= \ \mapsto 0$
- $>= \ \mapsto 0$
- $< \ \mapsto 0$
- $> \ \mapsto 0$

Lists

- $[\mapsto 0$
- $] \mapsto 0$
- $. \mapsto 0$

Variables

- $= \ \mapsto -\infty$

Scoping

- $\{ \mapsto 0$
- $\} \mapsto 0$

Products

- $, \ \mapsto 0$

Everything else is mapped to ∞