

Terms

(Stolen from the Incremental Flattening paper¹ :))

$$\begin{aligned}
 bop &::= + \mid - \mid * \mid / \mid \mathbf{i} \mid \dots \\
 op &::= \mathbf{transpose} \mid \mathbf{rearrange} (d, \dots, d) \mid \mathbf{replicate} \\
 soac &::= \mathbf{map} \mid \mathbf{reduce} \mid \mathbf{scan} \mid \mathbf{redomap} \mid \mathbf{scanomap} \\
 e &::= x \mid d \mid b \mid (e, \dots, e) \mid e[e] \mid e \mathbf{bop} e \mid op\ e \ \dots\ e \mid \mathbf{0} \\
 &\quad \mid \mathbf{loop} \ x_1 \ \dots\ x_n = e \ \mathbf{for} \ y < e \ \mathbf{do} \ e \\
 &\quad \mid \mathbf{loop} \ x_1 \ \dots\ x_n = e \ \mathbf{for} \ y = e \ \mathbf{to} \ 0 \ \mathbf{do} \ e \\
 &\quad \mid \mathbf{let} \ x_1 \ \dots\ x_n = e \ \mathbf{in} \ e \mid \mathbf{if} \ e \ \mathbf{then} \ e \ \mathbf{else} \ e \\
 &\quad \mid soac \ f \ e \ \dots\ e \\
 f &::= \lambda x_1 \ \dots\ x_n \rightarrow e \mid soac \ f \ e \ \dots\ e \mid e \mathbf{bop} \mid bop\ e
 \end{aligned}$$

The $\mathbf{0}$ expression is an array of zeros of arbitrary (and polymorphic!) shape.

Reverse-mode Rules

We define *tape maps* ($\parallel \Omega$) and *adjoint contexts* ($\Lambda \vdash$) as

$$\begin{aligned}
 \Omega &::= \varepsilon \mid \Omega, (x \mapsto x_s) \\
 \Lambda &::= \varepsilon \mid \Lambda, (x \mapsto \hat{x})
 \end{aligned}$$

The union of two maps prefers the right map in the instance of key conflicts:

$$(\Lambda_1 \cup \Lambda_2)[x] = \begin{cases} \Lambda_2[x] & \text{if } (x \mapsto \hat{x}) \in \Lambda_2 \\ \Lambda_1[x] & \text{otherwise} \end{cases}$$

Mappings of lists of variables is sugar for a list of mappings:

$$x_1 x_2 \dots x_n \mapsto \hat{x}_1 \hat{x}_2 \dots \hat{x}_n = \epsilon, (x_1 \mapsto \hat{x}_1), (x_2 \mapsto \hat{x}_2), \dots, (x_n \mapsto \hat{x}_n)$$

dom returns all keys of a map, i.e.,

$$\text{dom}(\epsilon, (x_1 \mapsto \hat{x}_1), (x_2 \mapsto \hat{x}_2)) = \{x_1, x_2\}$$

im returns all elements:

$$\text{im}(\epsilon, (x_1 \mapsto \hat{x}_1), (x_2 \mapsto \hat{x}_2)) = \{\hat{x}_1, \hat{x}_2\}$$

We're sloppy and overload the notation somewhat, so expressions like

$$\mathbf{let} \ \text{dom}(\epsilon, (x_1 \mapsto \hat{x}_1), (x_2 \mapsto \hat{x}_2)) = e_1 \ \mathbf{in} \ e_2$$

are to be understood as

$$\mathbf{let} \ x_1 \ x_2 = e_1 \ \mathbf{in} \ e_2$$

or

$$\mathbf{let} \ (x_1, x_2) = e_1 \ \mathbf{in} \ e_2$$

depending on the context. The difference of two maps is defined as

$$\Lambda_2 \setminus \Lambda_1 = \cup \{x \mapsto \hat{x} \mid \Lambda_2[x] \neq \Lambda_1[x], \Lambda_2[x] = \hat{x}\}$$

Reading a variable that isn't in a map always returns 0:

$$\varepsilon[x] = 0$$

Forward pass (\Rightarrow_F)

¹<https://futhark-lang.org/publications/ppopp19.pdf>

$$\frac{e = \mathbf{loop} \ \bar{x} = e_0 \ \mathbf{for} \ y < e_n \ \mathbf{do} \ e_{body} \quad x_{s_0} \text{ fresh} \quad x_{s_0} = \mathbf{replicate} \ e_n \ 0}{e \Rightarrow_F \mathbf{loop} \ (\bar{x}, x_s) = (e_0, x_{s_0}) \ \mathbf{for} \ y < e_n \ \mathbf{do} \ (e_{body}, x_s[y] = \bar{x}) \parallel (\bar{x} \mapsto x_s)} \text{FWDLOOP}$$

Reverse pass (\Uparrow)

$$\begin{array}{l} e_{body} = \mathbf{let} \ \bar{rs} = e'_{body} \ \mathbf{in} \ \bar{rs} \quad e_{loop} = \mathbf{let} \ \bar{lres} = \mathbf{loop} \ \bar{x} = e_0 \ \mathbf{for} \ y < e_n \ \mathbf{do} \ e_{body} \ \mathbf{in} \ \bar{lres} \\ e_{loop} \Rightarrow_F e'_{loop} \parallel \Omega \quad \bar{fv} = FV(e_{body}) \setminus \bar{x} \quad \bar{x}, \bar{fv}, \bar{fv}', \bar{fv}'', \bar{rs}, \bar{rs}' \text{ fresh} \\ \bar{reset} = \mathbf{map} \ (\lambda _ . 0) \ \bar{x} \quad \Lambda'_1 = \Lambda_1, \ \bar{x} \mapsto \bar{x}, \ \bar{fv} \mapsto \bar{fv}, \ \bar{rs} \mapsto \bar{rs} \\ \hat{e}_{body} = \mathbf{let} \ \bar{z} = e'_{body} \ \mathbf{in} \ \bar{z} \quad (\Lambda'_1 \vdash e_{body}) \Uparrow (\Lambda_2 \vdash \hat{e}_{body}) \\ \Lambda_{2,\Delta fv} = \{v \mapsto \hat{v} \mid v \in \bar{fv}, (v \mapsto \hat{v}) \in \Lambda_2 \setminus \Lambda'_1\} \quad \Lambda_{2,rs} = \{r \mapsto \hat{r} \mid r \in \bar{rs}, (r \mapsto \hat{r}) \in \Lambda_2\} \\ \hat{e}''_{body} = \mathbf{let} \ \bar{rs} = \Omega[y] \ \mathbf{in} \ (\mathbf{let} \ \bar{z}' = \hat{e}'_{body} \ \mathbf{in} \ (\bar{reset}, im(\Lambda_{2,rs}), im(\Lambda_{2,\Delta fv}))) \\ \hat{init} = (\bar{reset}, \Lambda_1[\bar{lres}], \Lambda_1[dom(\Lambda_{2,\Delta fv})]) \\ \hat{e}_{loop} = \mathbf{loop} \ (\bar{x}, \bar{rs}, \bar{fv}) = \hat{init} \ \mathbf{for} \ y = e_n - 1 \ \mathbf{to} \ 0 \ \mathbf{do} \ \hat{e}''_{body} \\ \Lambda_3 = \Lambda_1 \cup \left(dom(\Lambda_{2,\Delta fv}) \mapsto \bar{fv}'' \right) \\ \hline \Lambda_1 \vdash e_{loop} \Uparrow \left(\Lambda_3 \vdash \mathbf{let} \ (_, \bar{rs}', \bar{fv}') = \hat{e}_{loop} \ \mathbf{in} \ (\mathbf{let} \ \bar{fv}'' = (\mathbf{map} \ (+) \ \bar{fv}' \ \bar{rs}') \ \mathbf{in} \ \bar{fv}'') \right) \quad \text{REVLOOP} \\ \\ \Lambda \vdash e_t \Uparrow \Lambda_t \vdash \mathbf{let} \ \bar{fv}_t = \hat{e}_t \ \mathbf{in} \ \bar{fv}_t \quad \Lambda \vdash e_f \Uparrow \Lambda_f \vdash \mathbf{let} \ \bar{fv}_f = \hat{e}_f \ \mathbf{in} \ \bar{fv}_f \\ \Lambda_{\Delta_t} = \Lambda \setminus \Lambda_t \quad \Lambda_{\Delta_f} = \Lambda \setminus \Lambda_f \quad \hat{e}'_t = \mathbf{let} \ \bar{fv}_t = \hat{e}_t \ \mathbf{in} \ sort(\bar{fv}_t \uparrow\uparrow im(\Lambda_{\Delta_f} - \Lambda_{\Delta_t})) \\ \hat{e}'_f = \mathbf{let} \ \bar{fv}_f = \hat{e}_f \ \mathbf{in} \ sort(\bar{fv}_f \uparrow\uparrow im(\Lambda_{\Delta_t} - \Lambda_{\Delta_f})) \\ \bar{res} \text{ fresh} \quad \Lambda' = \Lambda, \ (dom(\Lambda_{\Delta_t} \cup \Lambda_{\Delta_f}) \mapsto \bar{res}) \\ \hline \Lambda \vdash \mathbf{let} \ \bar{res} = \mathbf{if} \ e_p \ \mathbf{then} \ e_t \ \mathbf{else} \ e_f \ \mathbf{in} \ \bar{res} \Uparrow \Lambda' \vdash \mathbf{let} \ \bar{res} = \mathbf{if} \ e_p \ \mathbf{then} \ \hat{e}'_t \ \mathbf{else} \ \hat{e}'_f \ \mathbf{in} \ \bar{res} \quad \text{REVIF} \end{array}$$