Terms

(Stolen from the Incremental Flattening paper[1] :))

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

The **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

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

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:

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

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

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

im returns all elements:

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

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

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

are to be understood as

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

or

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

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 ($\Rightarrow_F$)

$$\frac{e = \textbf{loop}\ \overline{x} = e_0\ \textbf{for}\ y < e_n\ \textbf{do}\ e_{body} \qquad x_{s_0}\ \text{fresh} \qquad x_{s_0} = \textbf{replicate}\ e_n\ \textbf{0}}{e \Rightarrow_F \textbf{loop}\ (\overline{x}, x_s) = (e_0, x_{s_0})\ \textbf{for}\ y < e_n\ \textbf{do}\ (e_{body}, x_s[y] = \overline{x}) \parallel (\overline{x} \mapsto x_s)}\ \textsc{FwdLoop}$$

[1] https://futhark-lang.org/publications/ppopp19.pdf

Reverse pass ($\Rightarrow_R$)

$$\frac{\begin{array}{c}
e_{body} = \textbf{let}\ \overline{rs} = e'_{body}\ \textbf{in}\ \overline{rs} \\
e_{loop} = \textbf{let}\ \overline{lres} = \textbf{loop}\ \overline{x} = e_0\ \textbf{for}\ y < e_n\ \textbf{do}\ e_{body}\ \textbf{in}\ \overline{lres} \qquad e_{loop} \Rightarrow_F e'_{loop} \parallel \Omega \\
\overline{fv} = FV(e_{body}) \setminus \overline{x} \qquad \overline{\hat{x}},\ \overline{\hat{fv}},\ \overline{\hat{fv}'},\ \overline{\hat{fv}''},\ \overline{\hat{rs}},\ \overline{\hat{rs}'}\ fresh \qquad \overline{reset} = \textbf{map}\ (\lambda_.\ \mathbf{0})\ \overline{\hat{x}} \\
\Lambda'_1 = \Lambda_1,\ \overline{x} \mapsto \overline{\hat{x}},\ \overline{fv} \mapsto \overline{\hat{fv}},\ \overline{rs} \mapsto \overline{\hat{rs}} \qquad \hat{e}_{body} = \textbf{let}\ \overline{\hat{z}} = \hat{e}'_{body}\ \textbf{in}\ \overline{\hat{z}} \qquad (\Lambda'_1 \vdash e_{body}) \Rightarrow_R (\Lambda_2 \vdash \hat{e}_{body}) \\
\Lambda_{2,\Delta fv} = \{v \mapsto \hat{v} \mid v \in \overline{fv}, (v \mapsto \hat{v}) \in \Lambda_2 \setminus \Lambda'_1\} \qquad \Lambda_{2,rs} = \{r \mapsto \hat{r} \mid r \in \overline{rs}, (r \mapsto \hat{r}) \in \Lambda_2\} \\
\hat{e}''_{body} = \textbf{let}\ \overline{rs} = \Omega[y]\ \textbf{in}\ (\textbf{let}\ \overline{\hat{z}'} = \hat{e}'_{body}\ \textbf{in}\ (\overline{reset}, im(\Lambda_{2,rs}), im(\Lambda_{2,\Delta fv}))) \\
\widehat{init} = (\overline{reset}, \Lambda_1[\overline{lres}], \Lambda_1[dom(\Lambda_{2,\Delta fv})]) \\
\hat{e}_{loop} = \textbf{loop}\ (\overline{\hat{x}}, \overline{\hat{rs}}, \overline{\hat{fv}}) = \widehat{init}\ \textbf{for}\ y = e_n - 1\ \textbf{to}\ 0\ \textbf{do}\ \hat{e}''_{body} \qquad \Lambda_3 = \Lambda_1 \cup \left(dom(\Lambda_{2,\Delta fv}) \mapsto \overline{\hat{fv}''} \right)
\end{array}}{\Lambda_1 \vdash e_{loop} \Rightarrow_R \left(\Lambda_3 \vdash \textbf{let}\ (_, \overline{\hat{rs}'}, \overline{\hat{fv}'}) = \hat{e}_{loop}\ \textbf{in}\ (\textbf{let}\ \overline{\hat{fv}''} = (\textbf{map}\ (+)\ \overline{\hat{fv}'}\ \overline{\hat{rs}'})\ \textbf{in}\ \overline{\hat{fv}''}) \right)}\ \textsc{RevForLoop}$$

$$\frac{\begin{array}{c}
\Lambda \vdash e_t \Rightarrow_R \Lambda_t \vdash \textbf{let}\ \overline{\hat{fv}_t} = \hat{e}_t\ \textbf{in}\ \overline{\hat{fv}_t} \qquad \Lambda \vdash e_f \Rightarrow_R \Lambda_f \vdash \textbf{let}\ \overline{\hat{fv}_f} = \hat{e}_f\ \textbf{in}\ \overline{\hat{fv}_f} \\
\Lambda_{\Delta_t} = \Lambda \setminus \Lambda_t \qquad \Lambda_{\Delta_f} = \Lambda \setminus \Lambda_f \qquad \hat{e}'_t = \textbf{let}\ \overline{\hat{fv}_t} = \hat{e}_t\ \textbf{in}\ sort(\overline{\hat{fv}_t} \mathbin{+\!\!+} im(\Lambda_{\Delta_f} - \Lambda_{\Delta_t})) \\
\hat{e}'_f = \textbf{let}\ \overline{\hat{fv}_f} = \hat{e}_f\ \textbf{in}\ sort(\overline{\hat{fv}_f} \mathbin{+\!\!+} im(\Lambda_{\Delta_t} - \Lambda_{\Delta_f})) \qquad \overline{\hat{res}}\ fresh \qquad \Lambda' = \Lambda,\ \left(dom\left(\Lambda_{\Delta_t} \cup \Lambda_{\Delta_f} \right) \mapsto \overline{\hat{res}} \right)
\end{array}}{\Lambda \vdash \textbf{let}\ \overline{res} = \textbf{if}\ e_p\ \textbf{then}\ e_t\ \textbf{else}\ e_f\ \textbf{in}\ \overline{res} \Rightarrow_R \Lambda' \vdash \textbf{let}\ \overline{\hat{res}} = \textbf{if}\ e_p\ \textbf{then}\ \hat{e}'_t\ \textbf{else}\ \hat{e}'_f\ \textbf{in}\ \overline{\hat{res}}}\ \textsc{RevIF}$$

$$\frac{\begin{array}{c}
f = \lambda\overline{x}\ \to \textbf{let}\ \overline{res} = e\ \textbf{in}\ \overline{rs} \\
\textbf{let}\ \overline{res} = e\ \textbf{in}\ \overline{res} \Rightarrow_F \textbf{let}\ \overline{res}\ \overline{x_s} = e'\ \textbf{in}\ \overline{res}\ \overline{x_s} \parallel \Omega \qquad \Lambda \vdash \textbf{let}\ \overline{res} = e\ \textbf{in}\ \overline{res} \Rightarrow_R \Lambda' \vdash \textbf{let}\ \overline{\hat{zs}} = \hat{e}\ \textbf{in}\ \overline{\hat{zs}} \\
\overline{\hat{bs}} = sort(\{\hat{x} \mid x \in \overline{x}, \Lambda[x] = \hat{x}, \hat{x} \in \overline{\hat{zs}}\}) \qquad \overline{\hat{fv}} = sort(\overline{\hat{zs}} \setminus \overline{\hat{bs}})
\end{array}}{\Lambda \vdash (\lambda\overline{x}\ \to \textbf{let}\ \overline{res} = e\ \textbf{in}\ \overline{rs}) \Rightarrow_R \Lambda' \vdash \left(\lambda\overline{x}\ (\Lambda'[\overline{res}])\ \to \textbf{let}\ \overline{rs}\ \overline{x_s} = e'\ \textbf{in}\ \textbf{let}\ \overline{\hat{bs}}\ \overline{\hat{fv}} = \hat{e}\ \textbf{in}\ (\overline{\hat{bs}}, \overline{\hat{fv}}) \right)}\ \textsc{RevLambda}$$

$$\frac{\Lambda \vdash (\lambda\overline{x}\ \to e) \Rightarrow_R \Lambda' \vdash \left(\lambda\overline{\hat{x}}\ \overline{\hat{res}}\ \to \textbf{let}\ \overline{x}\ \overline{\hat{bs}}\ \overline{\hat{fv}} = \hat{e}\ \textbf{in}\ (\overline{\hat{bs}}, \overline{\hat{fv}}) \right)}{\begin{array}{l}
\Lambda \vdash (\textbf{let}\ \overline{ys} = \textbf{map}\ (\lambda\overline{x}\ \to e)\ \overline{xs}\ \textbf{in}\ \overline{ys}) \\
\quad \Rightarrow_R \left(\Lambda', \overline{xs} \mapsto \overline{\hat{xs}}, sort(FV(\lambda\overline{x}\ \to e)) \mapsto \overline{\hat{fv}} \right) \vdash \\
\qquad \textbf{let}\ (\Delta\overline{\hat{xs}}, \Delta\overline{\hat{fvs}}) = \textbf{map}\ \left(\lambda\overline{\hat{x}}\ \overline{\hat{res}}\ \to \textbf{let}\ \overline{x}\ \overline{\hat{bs}}\ \overline{\hat{fv}} = \hat{e}\ \textbf{in}\ (\overline{\hat{bs}}, \overline{\hat{fv}}) \right)\ \overline{xs}\ (\Lambda_1[\overline{ys}])\ \textbf{in} \\
\qquad \textbf{let}\ \Delta\overline{\hat{fv}} = \textbf{reduce}\ (\textbf{zipwith}(+))\ \mathbf{0}\ \Delta\overline{\hat{fvs}}\ \textbf{in} \\
\qquad \textbf{let}\ \overline{\hat{xs}} = \textbf{updateArray}\ \overline{xs}\ \Delta\overline{\hat{xs}}\ \textbf{in} \\
\qquad \textbf{let}\ \overline{\hat{fv}'} = \textbf{zipwith}\ (+)\ \Lambda[sort(FV(\lambda\overline{x}\ \to e))]\ \Delta\overline{\hat{fv}}\ \textbf{in}\ (\overline{\hat{xs}}, \overline{\hat{fv}})
\end{array}}\ (\textsc{RevMap})$$