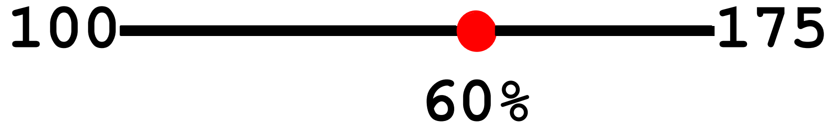
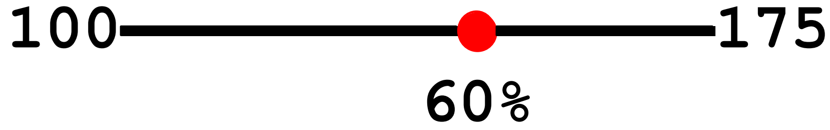
Static Semantics and Rank Polymorphism



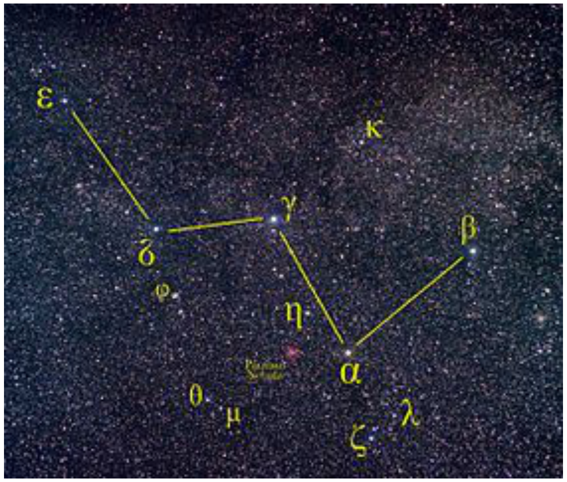

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```




```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 100
 175
 0.6)
```

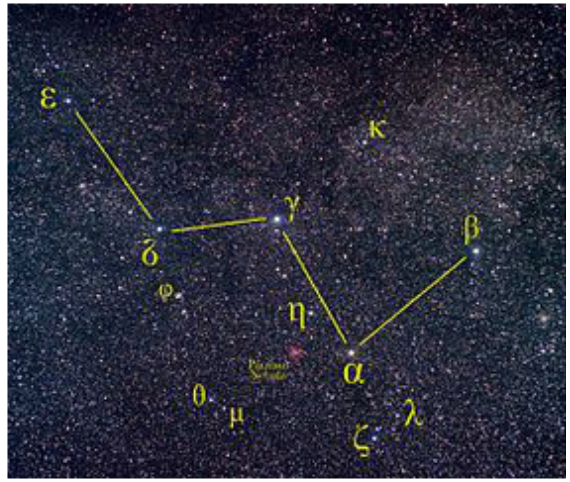

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```


```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 rgb1 ; 3 channels
 rgb2 ; 3 channels
 0.6) ; scalar
```



Credit: Wikimedia user Sadalsuud

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```



Credit: Wikimedia user Sadalsuud

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 sky         ; row × col × chan
 labels      ; row × col × chan
 img-mask)   ; row × col × chan
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 film       ; time × row × col × chan
 audience   ; time × row × col × chan
 vid-mask)  ; time × row × col × chan
```


```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```


```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 scene1     ; time × row × col × chan
 scene2     ; time × row × col × chan
 [0.0 ... 1.0]) ; time
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```

Polymorphic in dimensionality

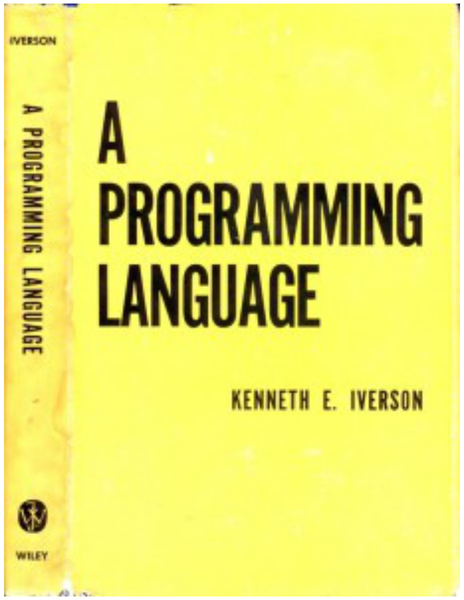
```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```

Polymorphic in dimensionality

Natural fit for data-parallel code

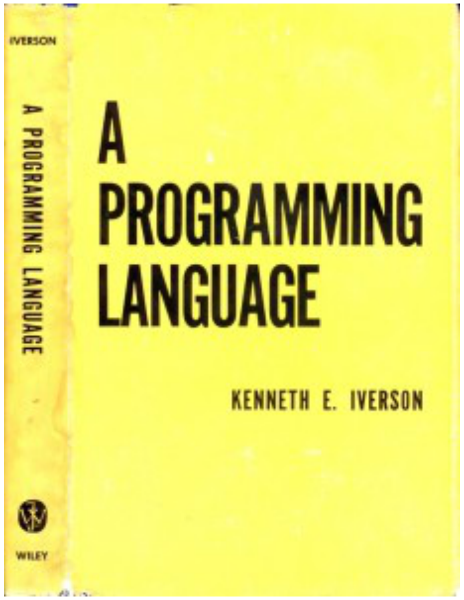
```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 x  ; unknown shape
 y  ; unknown shape
 z) ; unknown shape
```

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 x  ; unknown shape
 y  ; unknown shape
 z) ; unknown shape
```



1962

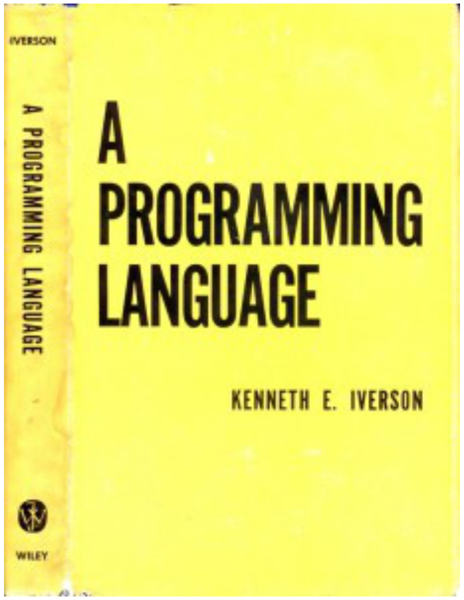
```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 x  ; unknown shape
 y  ; unknown shape
 z) ; unknown shape
```



Compilation?

1962

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 x  ; unknown shape
 y  ; unknown shape
 z) ; unknown shape
```



1962

Compilation?

- formal semantics
- dependent types
- bidirectional type inference
- decision procedures

- formal semantics
- dependent types
- bidirectional type inference
- decision procedures

- formal semantics
- dependent types
- bidirectional type inference
- decision procedures

Remora

- Rank-polymorphic programming model
- Using the programming model
- Type system
- Type inference
- Compiling rank-polymorphic programs

Rank-polymorphic programming model

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2}$$

$$\begin{bmatrix} 0 \end{bmatrix}_{\bullet}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

atoms: non-aggregate elements

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

$$\begin{bmatrix} 0 \end{bmatrix}_{\bullet}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7}$$

atoms: non-aggregate elements

$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2}$$

shape: size in each dimension

$$\begin{bmatrix} 0 \end{bmatrix}_{.}$$

Programming model

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}_{3,7} \qquad 2$$

atoms: non-aggregate elements

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{bmatrix}_{2,2,2} \qquad 3$$

shape: size in each dimension

$$\begin{bmatrix} 0 \end{bmatrix}_{\bullet} \qquad 0$$

rank: length of shape
i.e., number of dimensions

Programming model

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

Programming model

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2}\ \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

cell: basic unit function operates on

Programming model

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

cell: basic unit function operates on

Programming model

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

cell: basic unit function operates on

frame: structure around cells

Programming model

$$\texttt{dot-prod}\ \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\end{bmatrix}_{2}$$

cell: basic unit function operates on

frame: structure around cells

function applied to each cell

Programming model

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

cell: basic unit function operates on

frame: structure around cells

function applied to each cell

results reassembled in frame

Rank n array can split n+1 ways

Rank n array can split n+1 ways

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}_{3,4}$$

Rank n array can split n+1 ways

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}_{3,4} \qquad \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}_{3,4}$$

Rank n array can split n+1 ways

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}_{3,4} \qquad \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}_{3,4} \qquad \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \end{bmatrix}_{3,4}$$

Programming model

Frame-of-cells split chosen by function

Programming model

Frame-of-cells split chosen by function

$$\texttt{dot-prod}\quad \begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}\quad \begin{bmatrix}3 & 5\end{bmatrix}_{2}$$

Programming model

Frame-of-cells split chosen by function

`dot-prod` $\begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2} \quad \begin{bmatrix}3 & 5\end{bmatrix}_{2}$

`+` $\begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2} \quad \begin{bmatrix}3 & 5\end{bmatrix}_{2}$

Programming model

Frame-of-cells split chosen by function

`dot-prod` $\begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

`+` $\begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

`poly-eval` $\begin{bmatrix}1 & 4\\ 2 & 3\end{bmatrix}_{2,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

Programming model

Frame-of-cells split chosen by function

`dot-prod` $\begin{bmatrix}1 & 4\\2 & 3\end{bmatrix}_{{\color{red}2},{\color{blue}2}}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{\color{blue}2}$

`+` $\begin{bmatrix}1 & 4\\2 & 3\end{bmatrix}_{{\color{red}2},{\color{red}2}}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{\color{red}2}$

`poly-eval` $\begin{bmatrix}1 & 4\\2 & 3\end{bmatrix}_{{\color{red}2},{\color{blue}2}}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{\color{red}2}$

Principal frame: maximum under prefix ordering

Programming model

Frame-of-cells split chosen by function

$$\texttt{dot-prod} \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} = \begin{bmatrix} 23 & 21 \end{bmatrix}_{2}$$

$$+ \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} = \begin{bmatrix} 4 & 7 \\ 7 & 8 \end{bmatrix}_{2,2}$$

$$\texttt{poly-eval} \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} = \begin{bmatrix} 13 & 17 \end{bmatrix}_{2}$$

Principal frame: maximum under prefix ordering

Programming model

Trying different shapes

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

`+` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

Programming model

Trying different shapes

dot-prod $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

+ $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

poly-eval $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5\end{bmatrix}_{2}$

Programming model

Trying different shapes

dot-prod $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2} \quad \begin{bmatrix}3 & 5\end{bmatrix}_{2} = \begin{bmatrix}23 & 21 & 61\end{bmatrix}_{3}$

+ $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2} \quad \begin{bmatrix}3 & 5\end{bmatrix}_{2}$

poly-eval $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2} \quad \begin{bmatrix}3 & 5\end{bmatrix}_{2}$

Programming model

Trying different shapes

`dot-prod` $$\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} \quad = \quad \begin{bmatrix} 23 & 21 & 61 \end{bmatrix}_{3}$$

`+` $$\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} \quad \rightarrow$$

Shape error:
no principal frame

`poly-eval` $$\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

Programming model

Trying different shapes

$$\texttt{dot-prod} \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} \quad = \quad \begin{bmatrix} 23 & 21 & 61 \end{bmatrix}_{3}$$

$$+ \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} \quad \rightarrow$$

Shape error:
no principal frame

$$\texttt{poly-eval} \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} \quad \rightarrow$$

Shape error:
no principal frame

Programming model

Trying different shapes

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$

Programming model

Trying different shapes

$$\texttt{dot-prod} \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$$

$$+ \quad \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2} \quad \begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$$

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5 & 1\end{bmatrix}_{3}$

`+` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5 & 1\end{bmatrix}_{3}$

`poly-eval` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5 & 1\end{bmatrix}_{3}$

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5 & 1\end{bmatrix}_{3}$ → Domain error: `dot-prod` requires equal length args

`+` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5 & 1\end{bmatrix}_{3}$

`poly-eval` $\begin{bmatrix}1 & 4\\ 2 & 3\\ 7 & 8\end{bmatrix}_{3,2}$ $\begin{bmatrix}3 & 5 & 1\end{bmatrix}_{3}$

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$ $\rightarrow$ Domain error: **`dot-prod`** requires equal length args

+ $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$ $=$ $\begin{bmatrix} 4 & 7 \\ 7 & 8 \\ 8 & 9 \end{bmatrix}_{3,2}$

`poly-eval` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$

Programming model

Trying different shapes

`dot-prod` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$ $\rightarrow$ Domain error: **`dot-prod`** requires equal length args

`+` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$ $=$ $\begin{bmatrix} 4 & 7 \\ 7 & 8 \\ 8 & 9 \end{bmatrix}_{3,2}$

`poly-eval` $\begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 7 & 8 \end{bmatrix}_{3,2}$ $\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}_{3}$ $\rightarrow$ $\begin{bmatrix} 13 & 17 & 15 \end{bmatrix}_{3}$

Programming model

Rank annotations in user-defined functions

Programming model

Rank annotations in user-defined functions

```
(λ [(xs 1) (ys 1)]
  (reduce + 0 (* xs ys)))
```

Programming model

Rank annotations in user-defined functions

```
(λ [(xs 1) (ys 1)]
  (reduce + 0 (* xs ys)))
```

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```

Evaluation

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

Evaluation

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{\color{red}{2},\color{blue}{2}}\ \begin{bmatrix} 3 & 5 \end{bmatrix}_{\color{blue}{2}}$$

Evaluation

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{{\color{red}2},{\color{blue}2}} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{{\color{blue}2}}$$

$${\color{red}2} \quad {\color{blue}2} \qquad\qquad {\color{red}\bullet} \quad {\color{blue}2}$$

Evaluation

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2}\ \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

$$2 \quad 2 \qquad\qquad \bullet \quad 2$$

$$\bullet \ \sqsubseteq\ 2$$

Evaluation

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

$$2 \quad 2 \qquad\qquad \bullet \quad 2$$

$\bullet \sqsubseteq 2$ principal frame

Evaluation

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2}\ \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

$$\mapsto_{lift}\ \texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2}\ \begin{bmatrix} 3 & 5 \\ 3 & 5 \end{bmatrix}_{2,2}$$

Raise into principal frame by replication

Evaluation

$$\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2}\ \begin{bmatrix} 3 & 5 \\ 3 & 5 \end{bmatrix}_{2,2}$$

$$\mapsto_{map} \begin{bmatrix} \left(\texttt{dot-prod}\ \begin{bmatrix} 1 & 4 \end{bmatrix}_{2}\ \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}\right) \\ \left(\texttt{dot-prod}\ \begin{bmatrix} 2 & 3 \end{bmatrix}_{2}\ \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}\right) \end{bmatrix}_{2}$$

Apply function cell-wise

Evaluation

$$\left[\begin{array}{l}\left(\texttt{dot-prod}\ \left[\boxed{1\quad 4}\right]_{\color{blue}{2}}\ \left[\boxed{3\quad 5}\right]_{\color{blue}{2}}\right)\\ \left(\texttt{dot-prod}\ \left[\boxed{2\quad 3}\right]_{\color{blue}{2}}\ \left[\boxed{3\quad 5}\right]_{\color{blue}{2}}\right)\end{array}\right]_{\color{red}{2}}$$

$$\longmapsto_{\delta}{}^{2} \left[\begin{array}{l}[23]_{\bullet}\\ [21]_{\bullet}\end{array}\right]_{2}$$

Compute each result cell

Evaluation

$$\begin{bmatrix} \begin{bmatrix} 23 \end{bmatrix} \bullet \\ \begin{bmatrix} 21 \end{bmatrix} \bullet \end{bmatrix}_2$$

$$\longmapsto_{collapse} \begin{bmatrix} 23 \\ 21 \end{bmatrix}_2$$

Collapse frame-of-cells to single array

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2} \mapsto_{lift}$$

$$\texttt{dot-prod} \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{\color{red}{2},\color{blue}{2}} \begin{bmatrix} 3 & 5 \\ \color{orange}{3} & \color{orange}{5} \end{bmatrix}_{\color{orange}{2},\color{blue}{2}} \mapsto_{map}$$

$$\begin{bmatrix} \left(\texttt{dot-prod} \begin{bmatrix} 1 & 4 \end{bmatrix}_{\color{blue}{2}} \begin{bmatrix} 3 & 5 \end{bmatrix}_{\color{blue}{2}}\right) \\ \left(\texttt{dot-prod} \begin{bmatrix} 2 & 3 \end{bmatrix}_{\color{blue}{2}} \begin{bmatrix} 3 & 5 \end{bmatrix}_{\color{blue}{2}}\right) \end{bmatrix}_{\color{red}{2}} \mapsto_{\delta}^{2}$$

$$\begin{bmatrix} \begin{bmatrix} 23 \end{bmatrix}_{\bullet} \\ \begin{bmatrix} 21 \end{bmatrix}_{\bullet} \end{bmatrix}_{2} \mapsto_{collapse} \begin{bmatrix} 23 \\ 21 \end{bmatrix}_{2}$$

Using the programming model

Constructing an iteration space

Factorial:

```
(λ [(n 0)]
  (reduce * 1 (add1 (iota [n]))))
```

Constructing an iteration space

Factorial:

```
(λ [(n 0)]
  (reduce * 1 (add1 (iota [n]))))
```

```
(iota [n])                    [0 1 2 3 ... n-1]
```

Constructing an iteration space

Factorial:

```
(λ [(n 0)]
  (reduce * 1 (add1 (iota [n]))))
```

```
(iota [n])               [0 1 2 3 ... n-1]

(add1 (iota [n]))        [1 2 3 4 ... n]
```

Constructing an iteration space

Factorial:

```
(λ [(n 0)]
  (reduce * 1 (add1 (iota [n]))))
```

```
(iota [n])                 [0 1 2 3 ... n-1]

(add1 (iota [n]))          [1 2 3 4 ... n]

(reduce
 * 1                       (* 1 2 3 4 ... n)
 (add1 (iota [n])))        = n!
```

Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))
```

Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))


(rotate [2 3 4 3 2 1 0 1] 0)
```

Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))
```

(rotate [] 0)

Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))
```

(rotate [] 0)   []

Convolution:

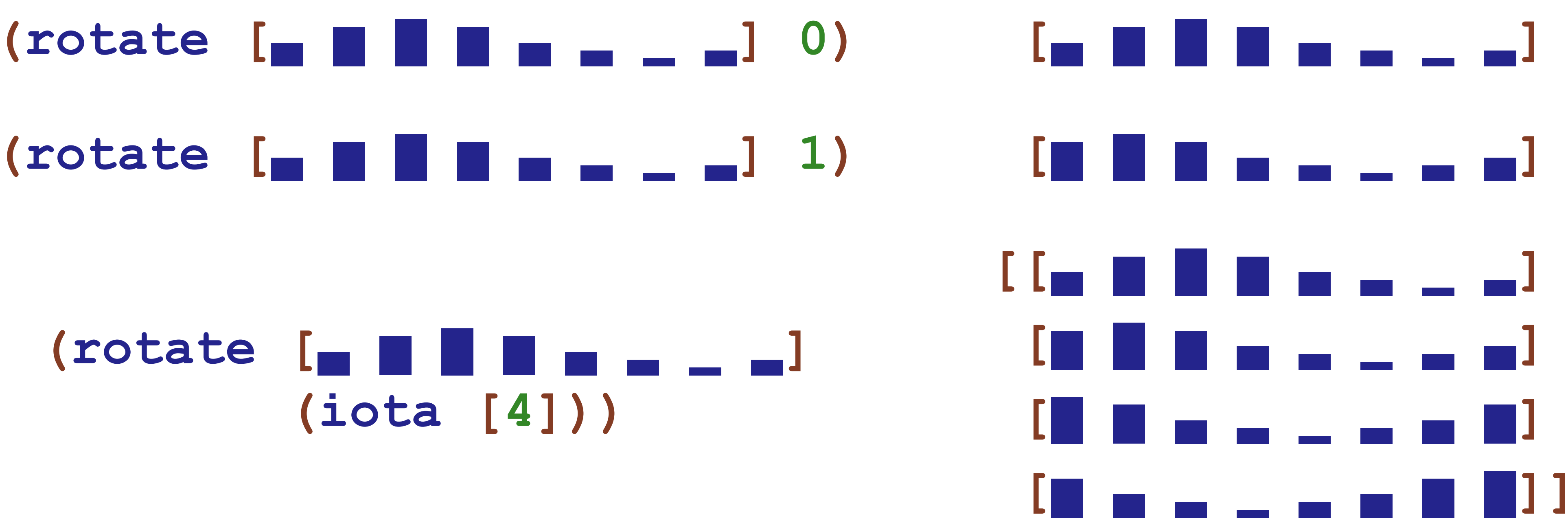
```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))
```

(rotate [] 0) []

(rotate [] 1) []

Convolution:

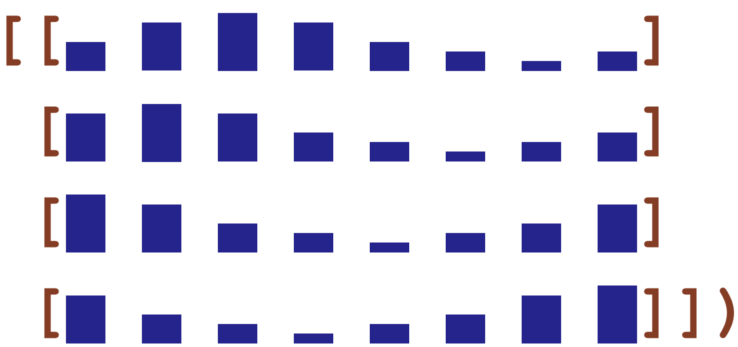
```
(λ [(filter 1 signal 1)]
   (reduce
    + 0
    (* filter
       (rotate signal
               (iota [(length filter)])))))
```



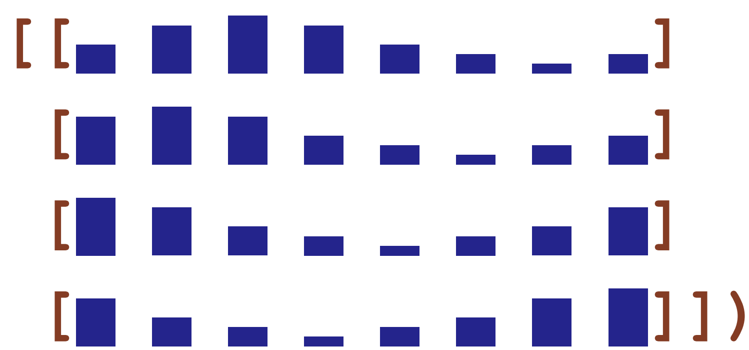
Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
    (* filter
       (rotate signal
               (iota [(length filter)])))))
```

```
(* [f0 f1 f2 f3]
```

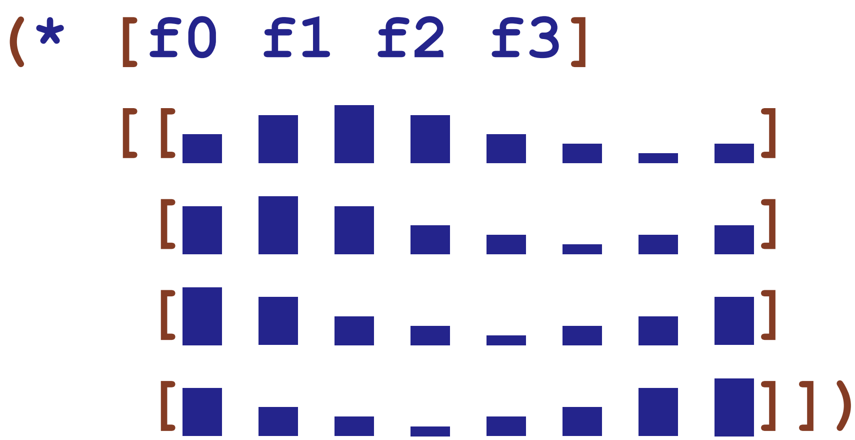


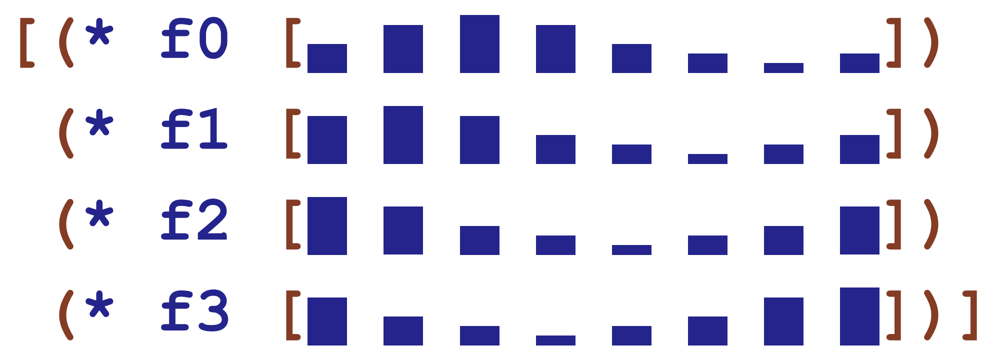
```
(* [[f0 f0 ... f0]
    [f1 f1 ... f1]
    [f2 f2 ... f2]
    [f3 f3 ... f3]]
```



Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
    (* filter
       (rotate signal
               (iota [(length filter)])))))
```

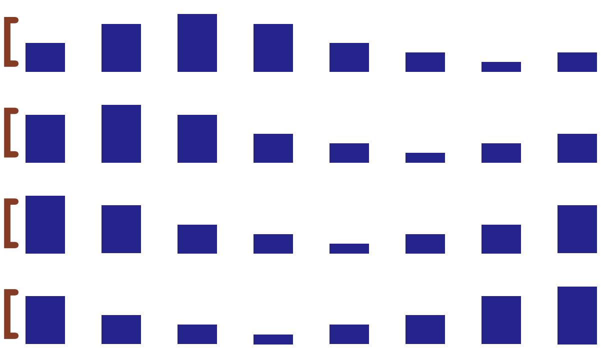




Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))
```

```
(reduce + 0
        [(* f0 [])
         (* f1 [])
         (* f2 [])
         (* f3 [])])
```



Convolution:

```
(λ [(filter 1 signal 1)]
  (reduce
   + 0
   (* filter
      (rotate signal
              (iota [(length filter)])))))
```

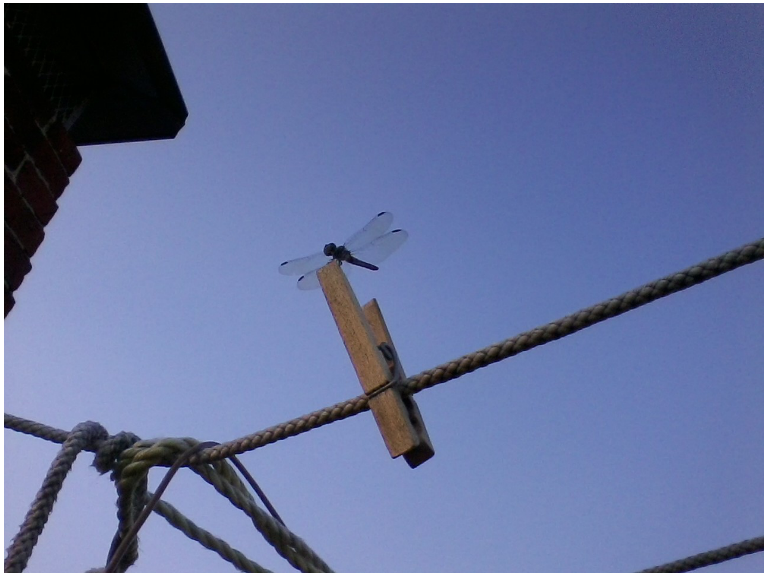
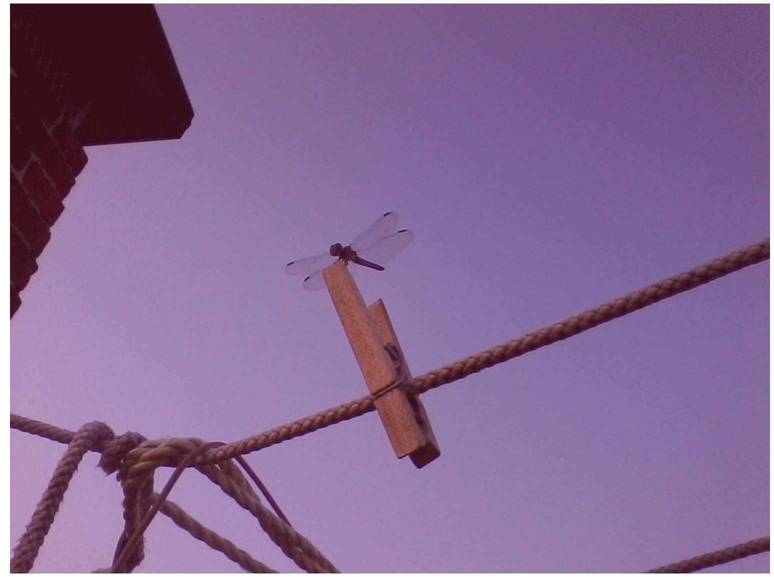
(+ (* f0 [])

(* f1 [])

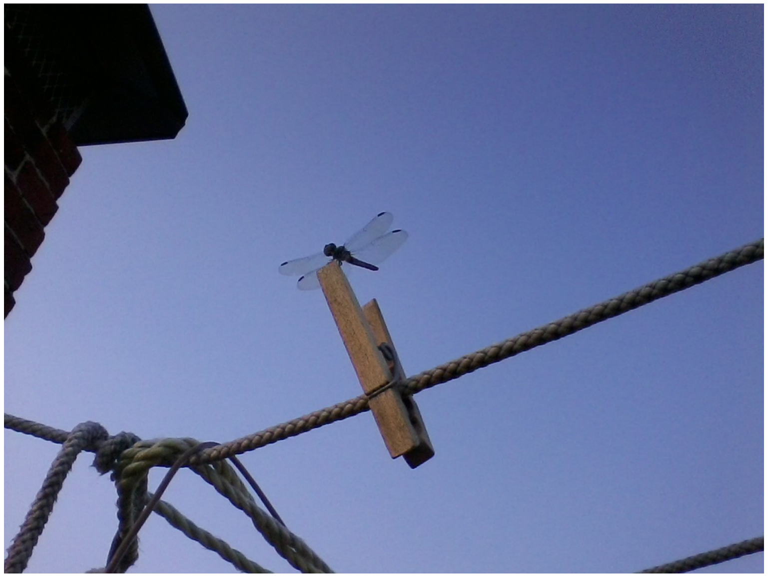
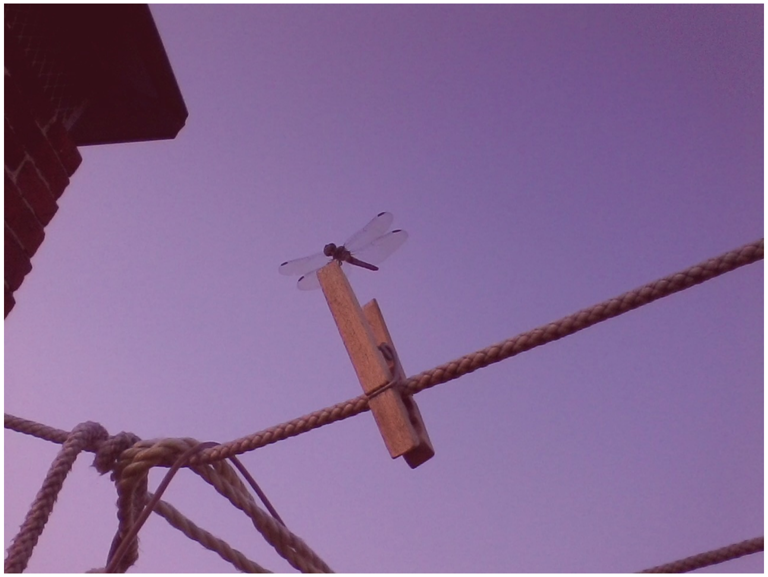
(* f2 [])

(* f3 []))

Tinting an image

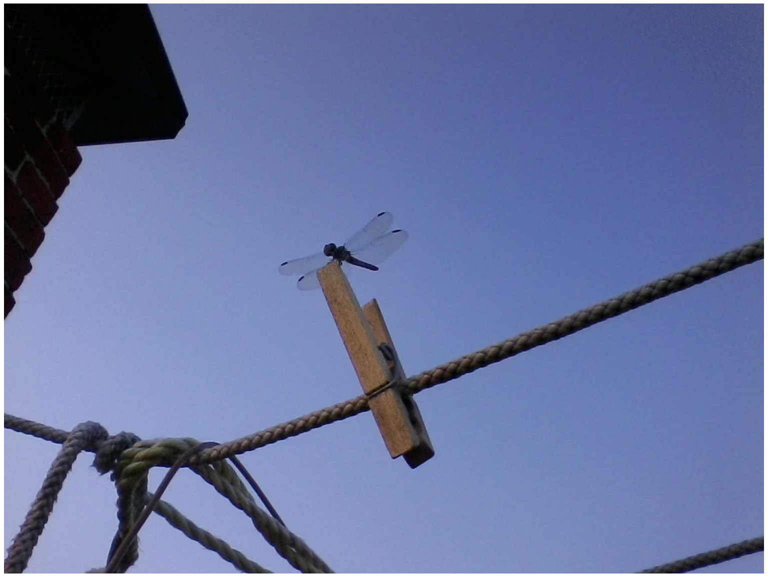
 

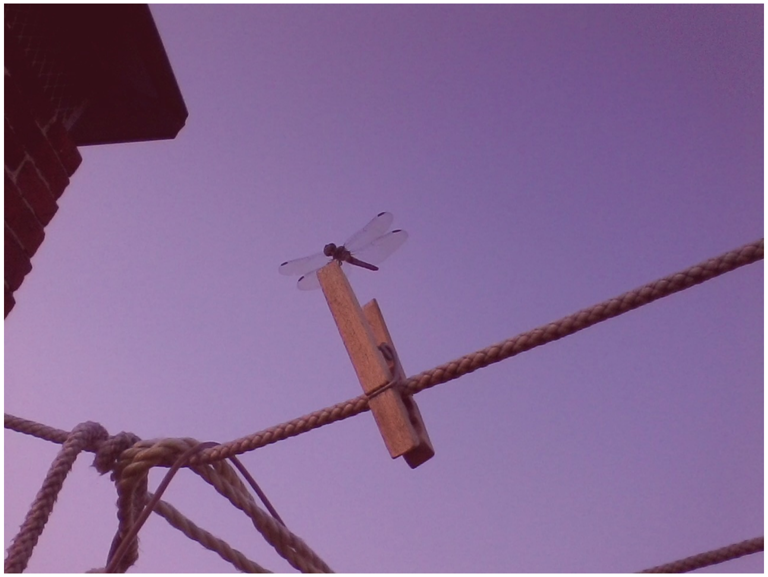
Tinting an image

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* hi α) (* lo (- 1 α))))
```

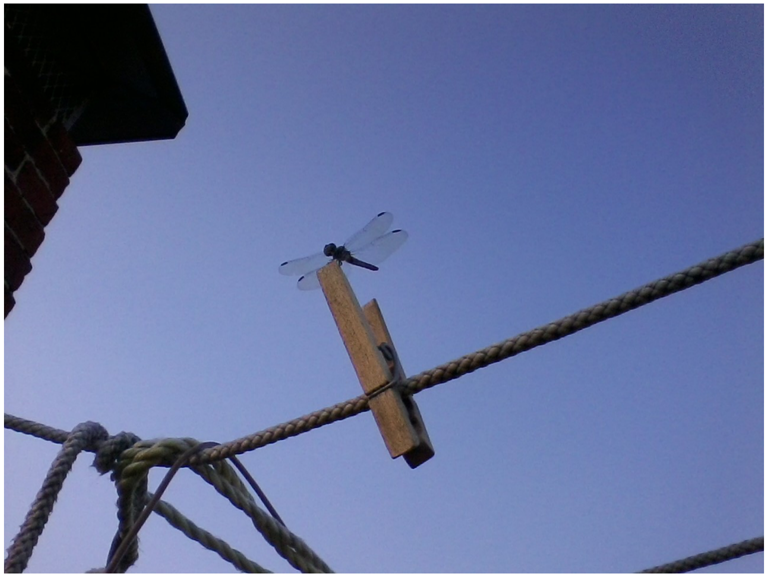
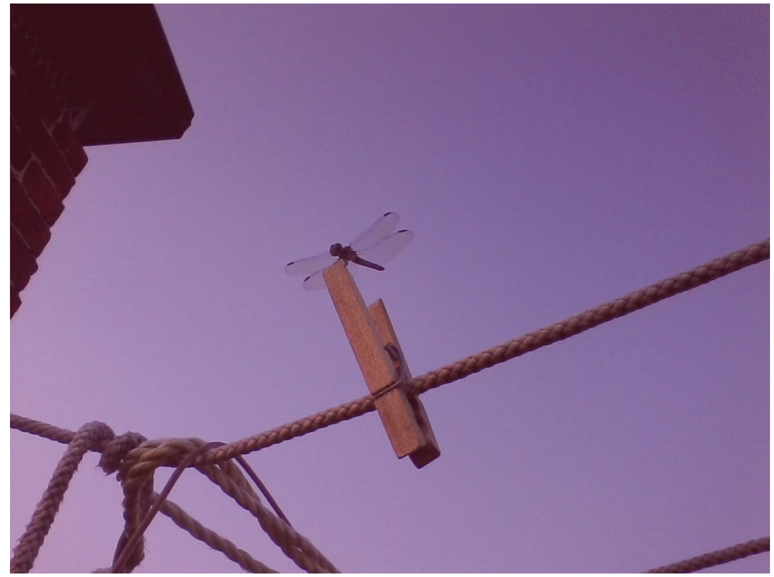
Tinting an image





```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

Tinting an image

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* hi α) (* lo (- 1 α))))
 photo   ; row × col × chan
 rgb     ; chan
 0.6)
```

Frame mismatch

Reranking

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* hi α) (* lo (- 1 α))))
```

Reranking

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
         (+ (* hi α) (* lo (- 1 α))))


      ((λ [(lo ?) (hi ?) (α ?)]
         (lerp lo hi α))
       photo  ; row × col × chan
       rgb    ; chan
       0.6)
```

Reranking

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
         (+ (* hi α) (* lo (- 1 α))))


      ((λ [(lo 1) (hi ?) (α ?)]
          (lerp lo hi α))
       photo  ; row × col × chan
       rgb    ; chan
       0.6)
```

Reranking

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
          (+ (* hi α) (* lo (- 1 α))))


      ((λ [(lo 1) (hi 1) (α ?)]
          (lerp lo hi α))
        photo   ; row × col × chan
        rgb     ; chan
        0.6)
```

Reranking

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
         (+ (* hi α) (* lo (- 1 α))))


      ((λ [(lo 1) (hi 1) (α 0)]
         (lerp lo hi α))
       photo  ; row × col × chan
       rgb    ; chan
       0.6)
```

Reranking

```
lerp = (λ [(lo 0) (hi 0) (α 0)]
         (+ (* hi α) (* lo (- 1 α))))
```

```
((λ [(lo 1) (hi 1) (α 0)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
((λ [(lo 1) (hi 1) (α 0)]
   (lerp lo hi α))
 photo          ; row × col × chan
 rgb-expanded ; row × col × chan
 [0.6 … 0.6]) ; row × col
```

Reranking

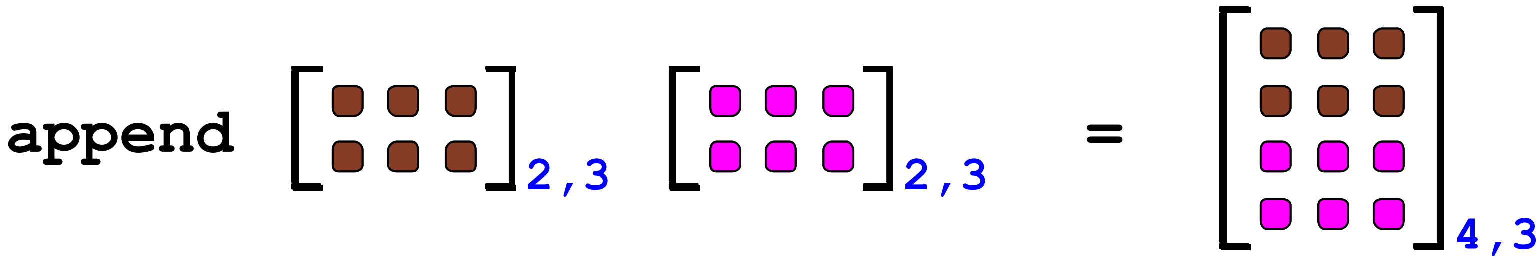
```
(λ [(lo 1) (hi 1) (α 0)]
  (lerp lo hi α))
```

Reranking

```
(λ [(lo 1) (hi 1) (α 0)]
  (lerp lo hi α))
```

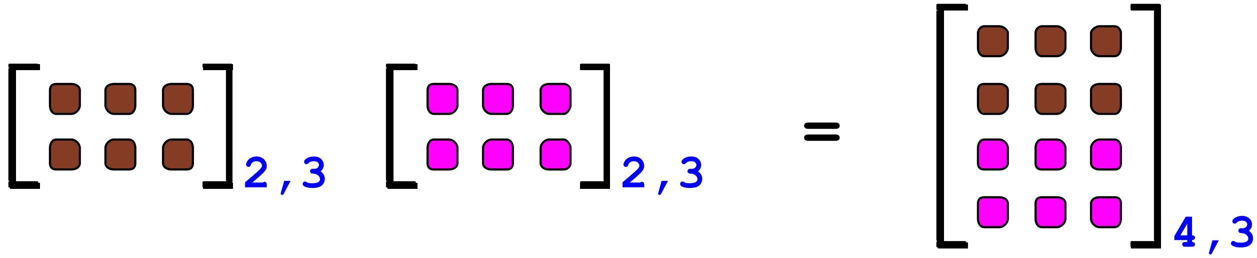
$\texttt{lerp}^{1,1,0}$

Axis alignment

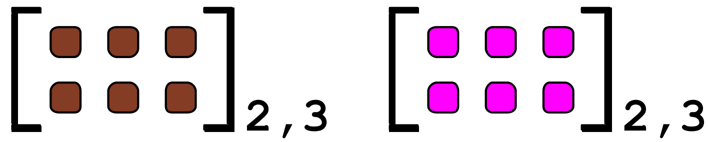


Axis alignment

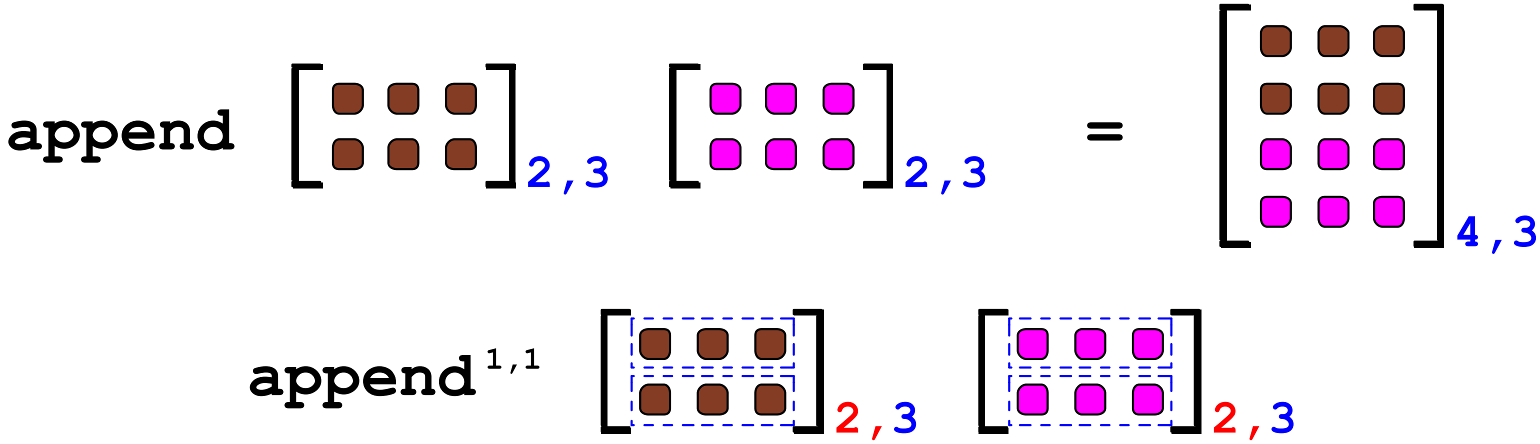
append



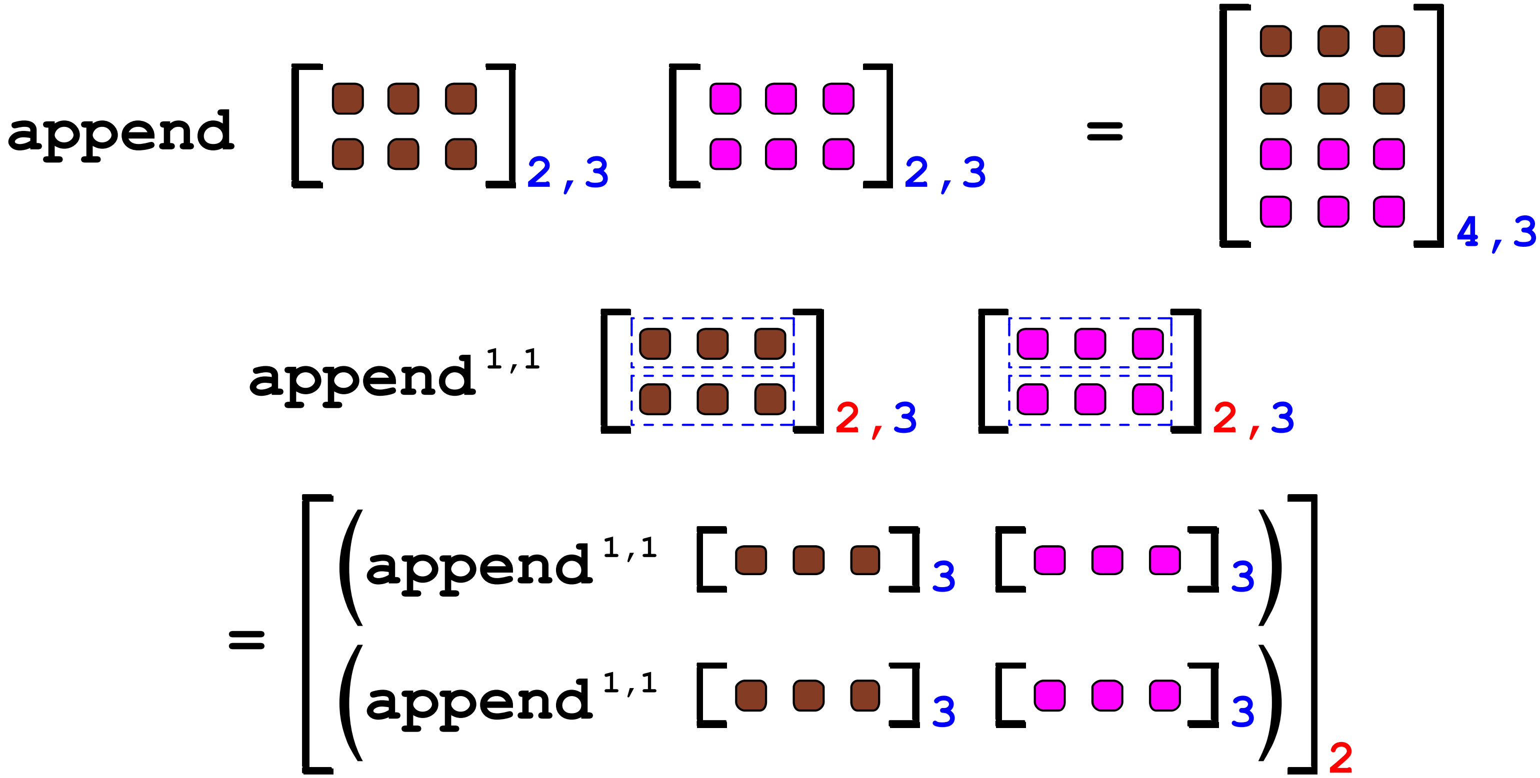
append^{1,1}



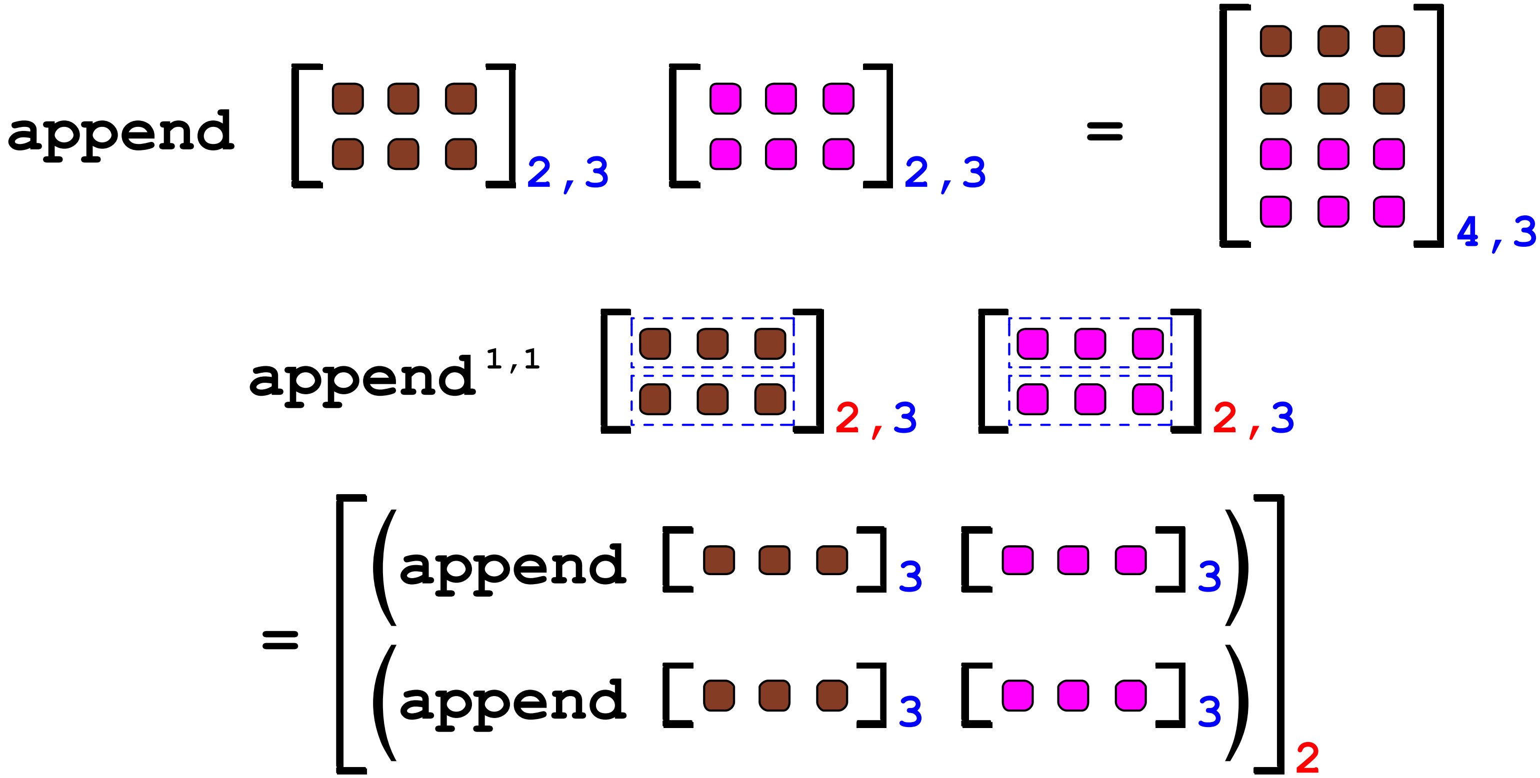
Axis alignment



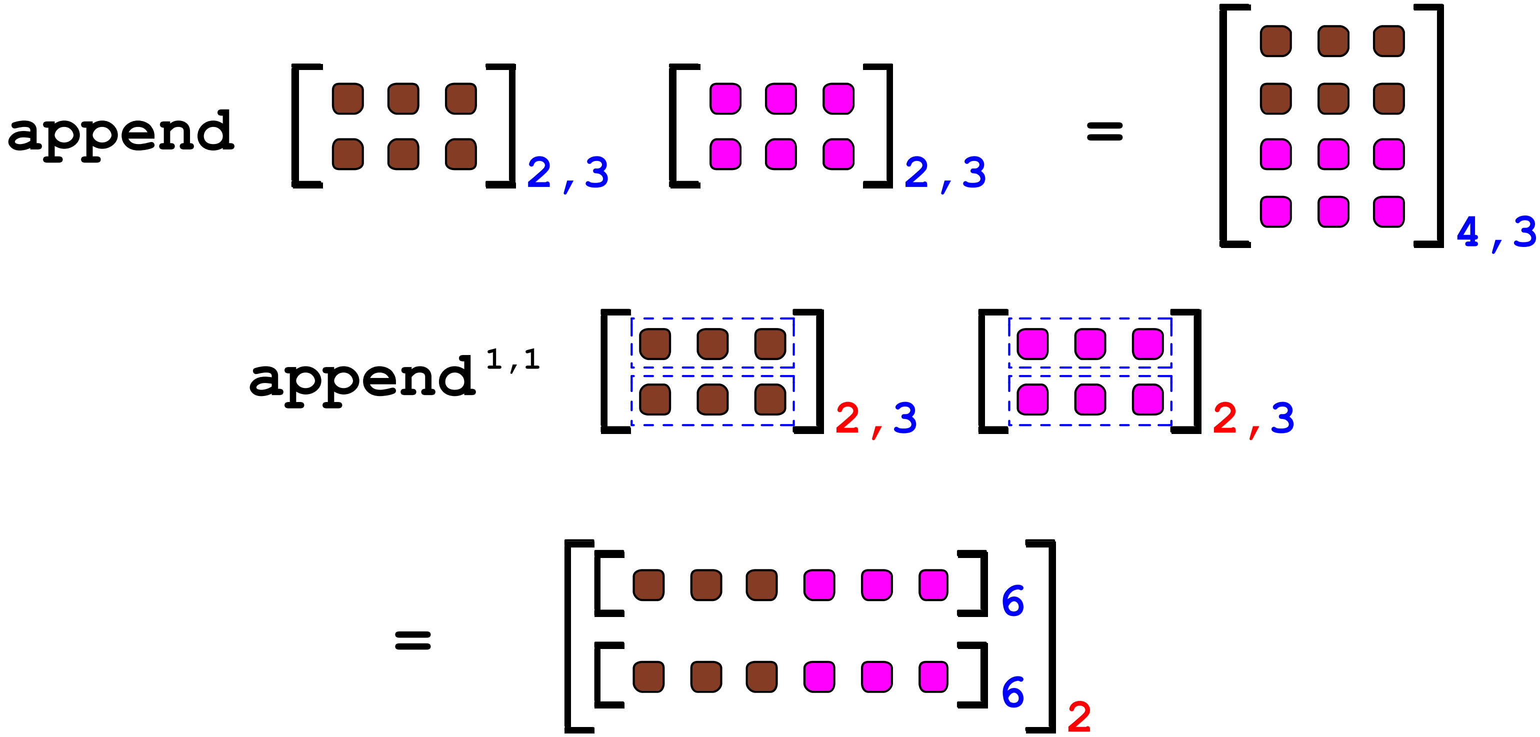
Axis alignment



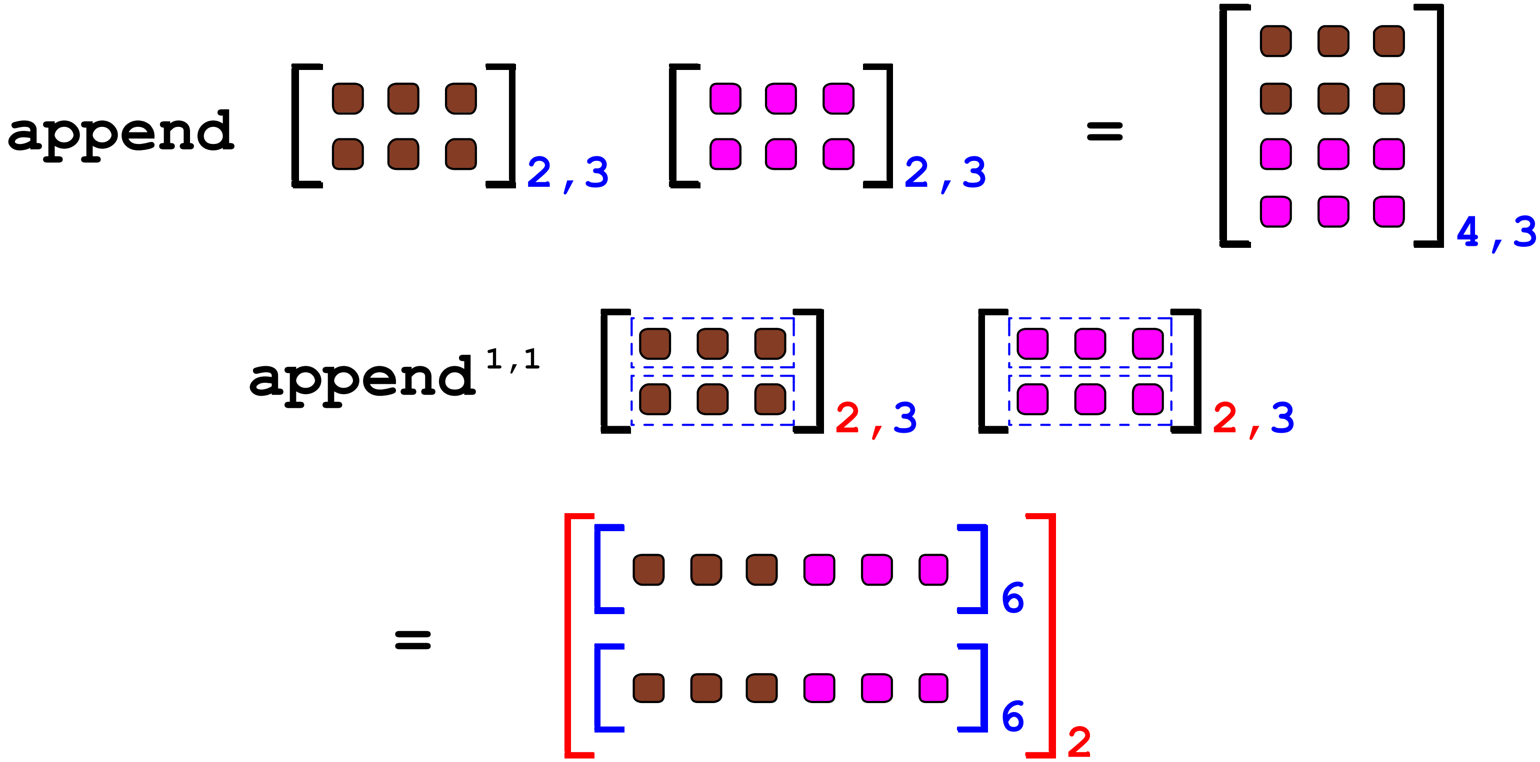
Axis alignment



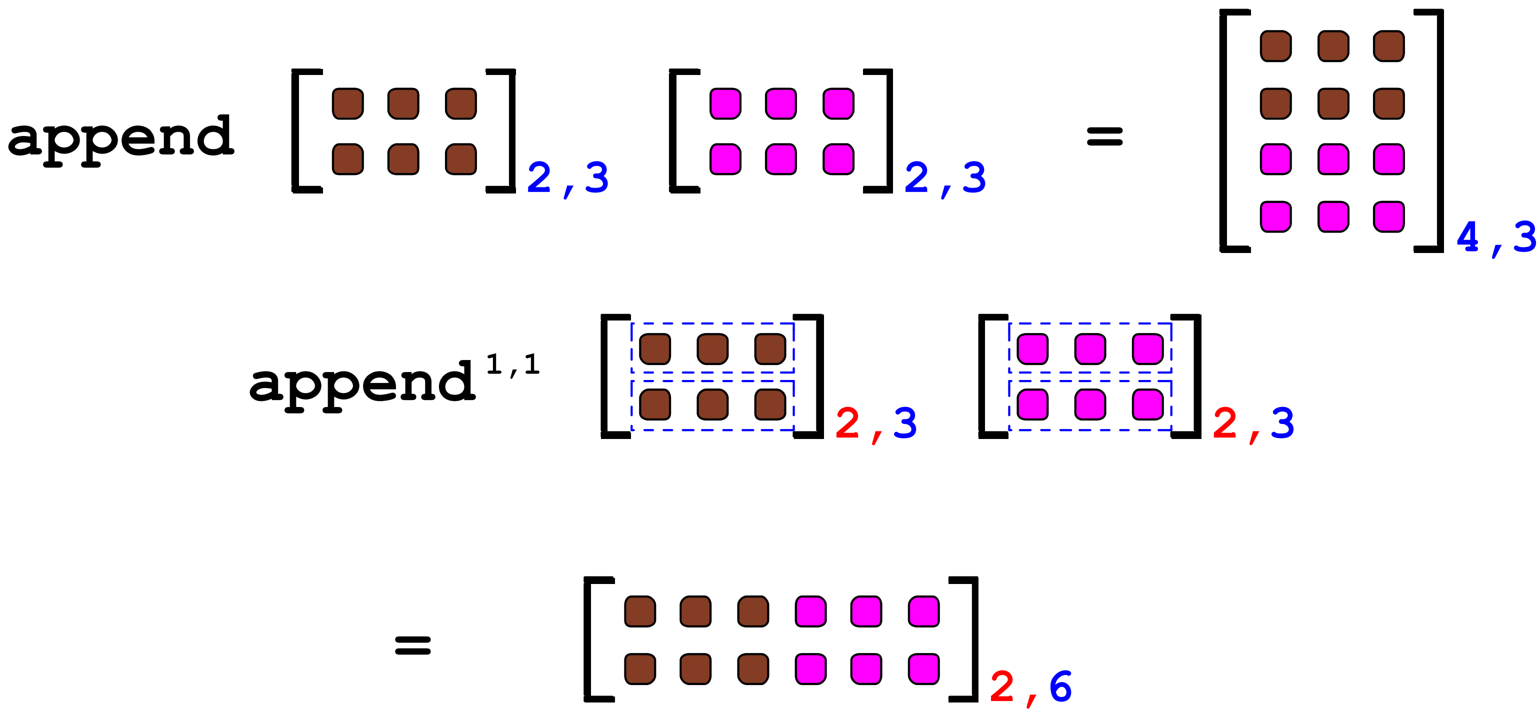
Axis alignment



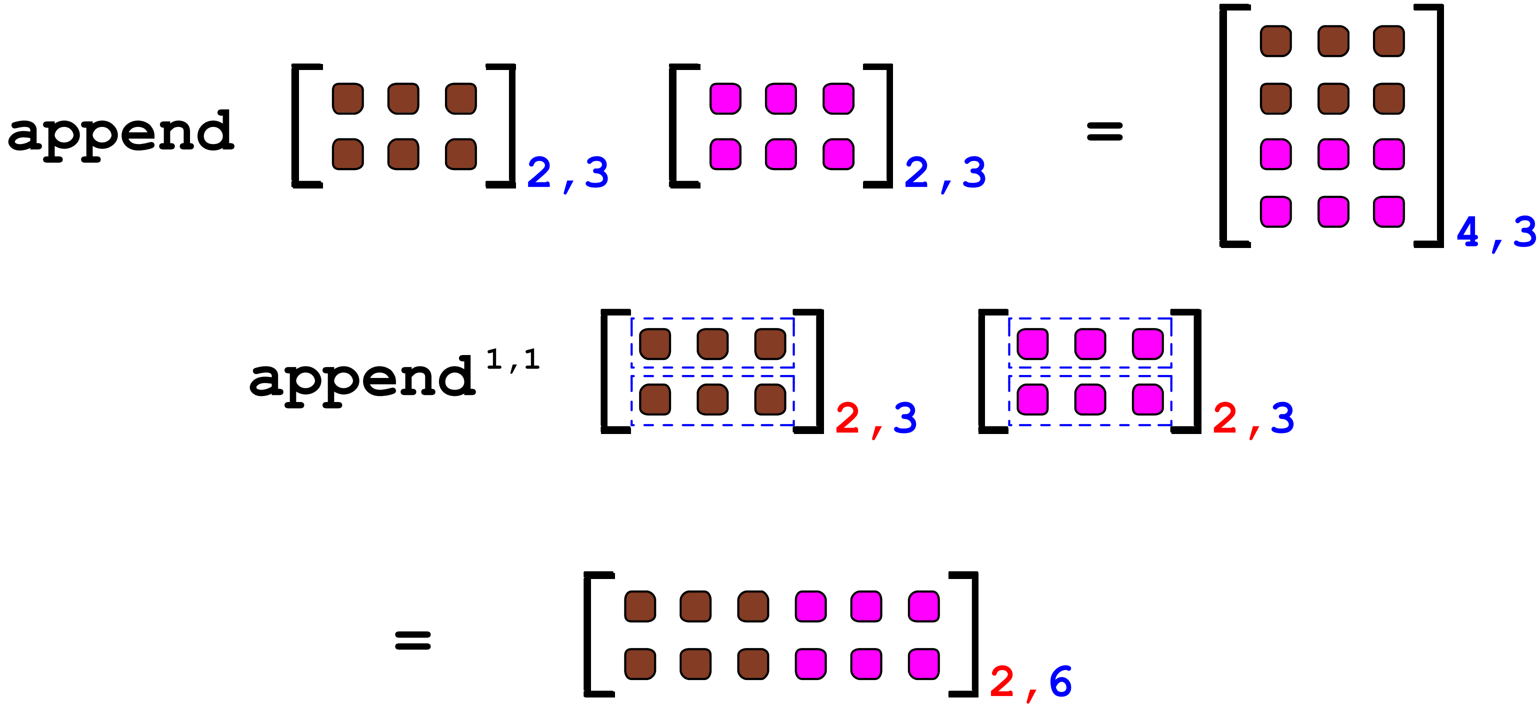
Axis alignment



Axis alignment



Axis alignment



Reranked version operates on different axis

Axis alignment

$$\texttt{*}^{\,0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_3\ \begin{bmatrix}1 & 10\end{bmatrix}_2$$

Axis alignment

$$*^{0,1} \quad \begin{bmatrix} \boxed{1} & \boxed{2} & \boxed{3} \end{bmatrix}_{3} \quad \begin{bmatrix} \boxed{1 \quad 10} \end{bmatrix}_{2}$$

Axis alignment

$$*^{0,1} \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_3 \quad \begin{bmatrix} 1 & 10 \end{bmatrix}_2$$

$$*^{0,1} \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_3 \quad \begin{bmatrix} 1 & 10 \\ 1 & 10 \\ 1 & 10 \end{bmatrix}_{3,2}$$

Axis alignment

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_3\ \begin{bmatrix}1 & 10\end{bmatrix}_2$$

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_3\ \begin{bmatrix}1 & 10\\ 1 & 10\\ 1 & 10\end{bmatrix}_{3,2}$$

$$\begin{bmatrix}\left(*^{0,1}\ \begin{bmatrix}1\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_2\right)\\ \left(*^{0,1}\ \begin{bmatrix}2\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_2\right)\\ \left(*^{0,1}\ \begin{bmatrix}3\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_2\right)\end{bmatrix}_3$$

Axis alignment

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_{3}\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}$$

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_{3}\ \begin{bmatrix}1 & 10\\ 1 & 10\\ 1 & 10\end{bmatrix}_{3,2}$$

$$\begin{bmatrix}\left(*^{0,1}\ \begin{bmatrix}1\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}\right)\\ \left(*^{0,1}\ \begin{bmatrix}2\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}\right)\\ \left(*^{0,1}\ \begin{bmatrix}3\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}\right)\end{bmatrix}_{3}$$

Axis alignment

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_{3}\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}$$

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_{3}\ \begin{bmatrix}1 & 10\\ 1 & 10\\ 1 & 10\end{bmatrix}_{3,2}$$

$$\begin{bmatrix}\left(*\ \begin{bmatrix}1\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}\right)\\ \left(*\ \begin{bmatrix}2\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}\right)\\ \left(*\ \begin{bmatrix}3\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_{2}\right)\end{bmatrix}_{3}$$

Axis alignment

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_3\ \begin{bmatrix}1 & 10\end{bmatrix}_2$$

$$*^{0,1}\ \begin{bmatrix}1 & 2 & 3\end{bmatrix}_3\ \begin{bmatrix}1 & 10\\ 1 & 10\\ 1 & 10\end{bmatrix}_{3,2}$$

$$\begin{bmatrix}\left(*\ \begin{bmatrix}1\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_2\right)\\ \left(*\ \begin{bmatrix}2\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_2\right)\\ \left(*\ \begin{bmatrix}3\end{bmatrix}.\ \begin{bmatrix}1 & 10\end{bmatrix}_2\right)\end{bmatrix}_3$$

Axis alignment

$$*^{0,1}\ \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{3}\ \begin{bmatrix} 1 & 10 \end{bmatrix}_{2}$$

$$*^{0,1}\ \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{3}\ \begin{bmatrix} 1 & 10 \\ 1 & 10 \\ 1 & 10 \end{bmatrix}_{3,2}$$

$$\begin{bmatrix} \left(*\ \begin{bmatrix} 1 & 1 \end{bmatrix}_{2} \begin{bmatrix} 1 & 10 \end{bmatrix}_{2}\right) \\ \left(*\ \begin{bmatrix} 2 & 2 \end{bmatrix}_{2} \begin{bmatrix} 1 & 10 \end{bmatrix}_{2}\right) \\ \left(*\ \begin{bmatrix} 3 & 3 \end{bmatrix}_{2} \begin{bmatrix} 1 & 10 \end{bmatrix}_{2}\right) \end{bmatrix}_{3}$$

Axis alignment

$$\ast^{0,1} \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{3} \quad \begin{bmatrix} 1 & 10 \end{bmatrix}_{2}$$

$$\ast^{0,1} \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{3} \quad \begin{bmatrix} 1 & 10 \\ 1 & 10 \\ 1 & 10 \end{bmatrix}_{3,2}$$

$$\begin{bmatrix} \begin{bmatrix} 1 & 10 \end{bmatrix}_{2} \\ \begin{bmatrix} 2 & 20 \end{bmatrix}_{2} \\ \begin{bmatrix} 3 & 30 \end{bmatrix}_{2} \end{bmatrix}_{3}$$

Axis alignment

$$*^{0,1} \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{3} \quad \begin{bmatrix} 1 & 10 \end{bmatrix}_{2}$$

$$*^{0,1} \quad \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}_{3} \quad \begin{bmatrix} 1 & 10 \\ 1 & 10 \\ 1 & 10 \end{bmatrix}_{3,2}$$

$$\begin{bmatrix} 1 & 10 \\ 2 & 20 \\ 3 & 30 \end{bmatrix}_{3,2}$$

Types for rank polymorphism

Type indices

$$\iota, \kappa \quad ::= \quad x$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\iota, \kappa \quad ::= \quad x$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\begin{array}{lcl} \iota, \kappa & ::= & x \\ & | & n \end{array}$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\begin{array}{rcl} \iota, \kappa & ::= & x \\ & | & n \end{array}$$

$$\frac{n \in \mathbb{N}}{\Theta \vdash n :: \texttt{Nat}}$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\begin{array}{rcl} \iota, \kappa & ::= & x \\ & | & n \\ & | & (\mathtt{S}\ \iota\ \ldots) \end{array}$$

$$\frac{n \in \mathbb{N}}{\Theta \vdash n :: \mathtt{Nat}}$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\frac{n \in \mathbb{N}}{\Theta \vdash n :: \texttt{Nat}}$$

$$\begin{array}{lcl} \iota, \kappa & ::= & x \\ & \mid & n \\ & \mid & (\texttt{S}\ \iota\ \ldots) \end{array}$$

$$\frac{\Theta \vdash \iota_j :: \texttt{Nat} \text{ for each } j}{\Theta \vdash (\texttt{S}\ \iota\ \ldots) :: \texttt{Shape}}$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\frac{n \in \mathbb{N}}{\Theta \vdash n :: \texttt{Nat}}$$

$$\begin{array}{lcl} \iota, \kappa & ::= & x \\ & \mid & n \\ & \mid & (\texttt{S}\ \iota\ \ldots) \\ & \mid & (+\ \iota\ \kappa) \end{array}$$

$$\frac{\Theta \vdash \iota_j :: \texttt{Nat} \text{ for each } j}{\Theta \vdash (\texttt{S}\ \iota\ \ldots) :: \texttt{Shape}}$$

Type indices

$$\frac{(x :: \gamma) \in \Theta}{\Theta \vdash x :: \gamma}$$

$$\begin{array}{rcl} \iota, \kappa & ::= & x \\ & | & n \\ & | & (\mathtt{S}\ \iota\ \ldots) \\ & | & (+\ \iota\ \kappa) \end{array}$$

$$\frac{n \in \mathbb{N}}{\Theta \vdash n :: \mathtt{Nat}}$$

$$\frac{\Theta \vdash \iota_j :: \mathtt{Nat} \text{ for each } j}{\Theta \vdash (\mathtt{S}\ \iota\ \ldots) :: \mathtt{Shape}}$$

$$\frac{\Theta \vdash \iota :: \mathtt{Nat} \quad \Theta \vdash \kappa :: \mathtt{Nat}}{\Theta \vdash (+\ \iota\ \kappa) :: \mathtt{Nat}}$$

Types

$$\begin{array}{rcl} \tau, \sigma & ::= & x \\ & \mid & B \\ & \mid & (\tau \rightarrow \sigma) \end{array}$$

$$\begin{array}{lcl} \Gamma & :: & \text{Type environment} \\ \Delta & :: & \text{Kind environment} \\ \Theta & :: & \text{Sort environment} \end{array}$$

Types

$$
\begin{array}{rcl}
\tau, \sigma & ::= & x \\
& | & B \\
& | & (\tau \to \sigma) \\
& | & \mathtt{A}_\iota \tau
\end{array}
$$

$$
\begin{array}{rcl}
\Gamma & :: & \text{Type environment} \\
\Delta & :: & \text{Kind environment} \\
\Theta & :: & \text{Sort environment}
\end{array}
$$

Types

$$\begin{array}{rcl} \tau, \sigma & ::= & x \\ & | & B \\ & | & (\tau \to \sigma) \\ & | & \mathtt{A}_\iota \tau \end{array}$$

$$\frac{\begin{array}{c} \Gamma; \Delta; \Theta \vdash e : \mathtt{A}_\iota (\sigma \ \ldots \ \to \tau) \\ \Gamma; \Delta; \Theta \vdash e'_j : \mathtt{A}_{\kappa_j} \sigma_j \quad \text{for each } j \\ \iota' = Max\, [\![\iota, \kappa \ \ldots\ ]\!] \end{array}}{\Gamma; \Delta; \Theta \vdash (e \ e' \ \ldots\ ) : \mathtt{A}_{\iota'} \tau}$$

$$\begin{array}{rcl} \Gamma & :: & \text{Type environment} \\ \Delta & :: & \text{Kind environment} \\ \Theta & :: & \text{Sort environment} \end{array}$$

Types

$$\frac{\begin{array}{c}\Gamma ; \Delta ; \Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \rightarrow \tau) \\ \Gamma ; \Delta ; \Theta \vdash e'_j : \mathtt{A}_{\kappa_j} \sigma_j \quad \text{for each } j \\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma ; \Delta ; \Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'} \tau}$$

Types

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \rightarrow \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\frac{}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\,\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\,\mathtt{Int}})\ :\ ?}$$

Types

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \to \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\mathtt{+} : \mathtt{A}_{(\mathtt{s})}\left(\mathtt{A}_{(\mathtt{s})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{s})}\mathtt{Int} \to \mathtt{A}_{(\mathtt{s})}\mathtt{Int}\right)$$

$$\frac{}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{s}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{s}\ 3\ 2)}\mathtt{Int}})\ :\ ?}$$

Types

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \rightarrow \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\mathtt{+} : \mathtt{A}_{(\mathtt{S})}\left(\mathtt{A}_{(\mathtt{S})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{S})}\mathtt{Int} \rightarrow \color{blue}{\mathtt{A}_{(\mathtt{S})}\mathtt{Int}}\right)$$

$$\mathtt{vec} : \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int} \qquad \mathtt{mat} : \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}$$

$$\frac{}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}})\ :\ ?}$$

Types

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \to \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ \ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\frac{\begin{array}{c}\mathtt{+} : \mathtt{A}_{(\mathtt{S})}\left(\mathtt{A}_{(\mathtt{S})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{S})}\mathtt{Int} \to {\color{blue}\mathtt{A}_{(\mathtt{S})}\mathtt{Int}}\right) \\ \mathtt{vec} : \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int} \qquad \mathtt{mat} : \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int} \\ (\mathtt{S}) \qquad (\mathtt{S}\ 3) \qquad (\mathtt{S}\ 3\ 2)\end{array}}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}})\ :\ ?}$$

Types

$$\frac{\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \rightarrow \tau) \qquad \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \qquad \iota' = \mathit{Max}\,[\![\iota, \kappa\ \ldots]\!]}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\frac{\begin{array}{c} \mathtt{+} : \mathtt{A}_{(\mathtt{S})}\left(\mathtt{A}_{(\mathtt{S})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{S})}\mathtt{Int} \rightarrow \color{blue}{\mathtt{A}_{(\mathtt{S})}\mathtt{Int}}\right) \\ \mathtt{vec} : \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int} \qquad \mathtt{mat} : \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int} \\ (\mathtt{S}) \qquad (\mathtt{S}\ 3) \qquad \color{red}{(\mathtt{S}\ 3\ 2)} \end{array}}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}}) : ?}$$

Types

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \to \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\,\sigma_j \quad \text{for each } j \\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\,\tau}$$

$$\frac{\begin{array}{c}\mathtt{+} : \mathtt{A}_{(\mathtt{S})}\left(\mathtt{A}_{(\mathtt{S})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{S})}\mathtt{Int} \to \color{blue}{\mathtt{A}_{(\mathtt{S})}\mathtt{Int}}\right) \\ \mathtt{vec} : \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int} \qquad \mathtt{mat} : \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int} \\ (\mathtt{S}) \quad (\mathtt{S}\ 3) \quad \color{red}{(\mathtt{S}\ 3\ 2)} \qquad \mathtt{A}_{\color{red}{(\mathtt{S}\ 3\ 2)}}\color{blue}{\mathtt{A}_{(\mathtt{S})}\mathtt{Int}}\end{array}}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}})\ :\ ?}$$

Types

$$\frac{\begin{array}{c}\Gamma; \Delta; \Theta \vdash e : \mathtt{A}_{\iota}\, (\sigma\ \ldots \to \tau) \\ \Gamma; \Delta; \Theta \vdash e'_j : \mathtt{A}_{\kappa_j} \sigma_j \quad \text{for each } j \\ \iota' = Max\, [\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma; \Delta; \Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\, \tau}$$

$$\frac{\begin{array}{c}+ : \mathtt{A}_{(\mathtt{S})}\left(\mathtt{A}_{(\mathtt{S})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{S})}\mathtt{Int} \to {\color{blue}\mathtt{A}_{(\mathtt{S})}\mathtt{Int}}\right) \\ \mathtt{vec} : \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int} \qquad \mathtt{mat} : \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int} \\ (\mathtt{S}) \qquad (\mathtt{S}\ 3) \qquad {\color{red}(\mathtt{S}\ 3\ 2)} \qquad \mathtt{A}_{\color{red}(\mathtt{S}\ 3\ 2)}{\color{blue}\mathtt{Int}}\end{array}}{(+\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}})\ :\ ?}$$

Types

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots \to \tau)\\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j\\ \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\frac{\begin{array}{c}\mathtt{+} : \mathtt{A}_{(\mathtt{S})}\left(\mathtt{A}_{(\mathtt{S})}\mathtt{Int}\ \ \mathtt{A}_{(\mathtt{S})}\mathtt{Int} \to \mathtt{A}_{(\mathtt{S})}\mathtt{Int}\right)\\ \mathtt{vec} : \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int} \qquad \mathtt{mat} : \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}\\ (\mathtt{S}) \quad (\mathtt{S}\ 3) \quad (\mathtt{S}\ 3\ 2) \qquad \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}\end{array}}{(\mathtt{+}\ \ \mathtt{vec}^{\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Int}}\ \ \mathtt{mat}^{\mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}}) \ :\ \mathtt{A}_{(\mathtt{S}\ 3\ 2)}\mathtt{Int}}$$

Types

$$\begin{array}{rcl} \tau, \sigma & ::= & x \\ & \mid & B \\ & \mid & (\tau \to \sigma) \\ & \mid & \mathtt{A}_\iota \tau \\ & \mid & (\forall\, [x\ \ldots]\ \tau) \end{array}$$

$$\frac{\Gamma; \Delta; \Theta \vdash e : \mathtt{A}_\iota\, (\sigma\ \ldots\ \to \tau) \qquad \Gamma; \Delta; \Theta \vdash e'_j : \mathtt{A}_{\kappa_j} \sigma_j \quad \text{for each } j \qquad \iota' = Max\, [\![\iota, \kappa\ \ldots]\!]}{\Gamma; \Delta; \Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'} \tau}$$

$$\begin{array}{rcl} \Gamma & :: & \text{Type environment} \\ \Delta & :: & \text{Kind environment} \\ \Theta & :: & \text{Sort environment} \end{array}$$

Types

$$
\begin{array}{rcl}
\tau, \sigma & ::= & x \\
& | & B \\
& | & (\tau \rightarrow \sigma) \\
& | & \mathtt{A}_\iota \tau \\
& | & (\forall\, [x\ \ldots]\ \tau)
\end{array}
$$

$$
\frac{\Gamma; \Delta; \Theta \vdash e : \mathtt{A}_\iota\,(\sigma\ \ldots\ \rightarrow \tau) \qquad \Gamma; \Delta; \Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \qquad \iota' = Max\,[\![\iota, \kappa\ \ldots]\!]}{\Gamma; \Delta; \Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}
$$

$$
\begin{array}{rcl}
\Gamma & :: & \text{Type environment} \\
\Delta & :: & \text{Kind environment} \\
\Theta & :: & \text{Sort environment}
\end{array}
$$

$$
\frac{\Gamma; \Delta; \Theta \vdash l : (\forall\, [x\ \ldots]\ \sigma) \qquad \Delta; \Theta \vdash \tau_j \quad \text{for each } j \qquad \text{no } \tau_j \text{ is an array type}}{\Gamma; \Delta; \Theta \vdash (\mathtt{T\text{-}APP}\ l\ \tau\ \ldots) : \sigma[(x \leftarrow_t \tau)\ \ldots]}
$$

Types

$$\tau, \sigma \quad ::= \quad x \mid B \mid (\tau \to \sigma) \mid \mathtt{A}_\iota\,\tau \mid (\forall\,[x\ \ldots]\ \tau) \mid (\Pi\,[(x\ \gamma)\ \ldots]\ \tau)$$

$$\Gamma \quad :: \quad \text{Type environment}$$
$$\Delta \quad :: \quad \text{Kind environment}$$
$$\Theta \quad :: \quad \text{Sort environment}$$

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_\iota\,(\sigma\ \ldots\ \to \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \\ \iota' = \mathit{Max}\,[\![\iota,\kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}$$

$$\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash l : (\forall\,[x\ \ldots]\ \sigma) \\ \Delta;\Theta \vdash \tau_j \quad \text{for each } j \\ \text{no } \tau_j \text{ is an array type}\end{array}}{\Gamma;\Delta;\Theta \vdash (\mathtt{T\text{-}APP}\ l\ \tau\ \ldots) : \sigma[(x \leftarrow_t \tau)\ \ldots]}$$

Types

$$
\begin{array}{rcl}
\tau, \sigma & ::= & x \\
& | & B \\
& | & (\tau \rightarrow \sigma) \\
& | & \mathtt{A}_{\iota}\, \tau \\
& | & (\forall\, [x\ \ldots]\ \tau) \\
& | & (\Pi\, [(x\ \gamma)\ \ldots]\ \tau) \\
& | & (\Sigma\, [(x\ \gamma)\ \ldots]\ \tau)
\end{array}
$$

$$
\begin{array}{rcl}
\Gamma & :: & \text{Type environment} \\
\Delta & :: & \text{Kind environment} \\
\Theta & :: & \text{Sort environment}
\end{array}
$$

$$
\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash e : \mathtt{A}_{\iota}\,(\sigma\ \ldots\ \rightarrow \tau) \\ \Gamma;\Delta;\Theta \vdash e'_j : \mathtt{A}_{\kappa_j}\sigma_j \quad \text{for each } j \\ \iota' = Max\, [\![\iota, \kappa\ \ldots]\!]\end{array}}{\Gamma;\Delta;\Theta \vdash (e\ e'\ \ldots) : \mathtt{A}_{\iota'}\tau}
$$

$$
\frac{\begin{array}{c}\Gamma;\Delta;\Theta \vdash l : (\forall\, [x\ \ldots]\ \sigma) \\ \Delta;\Theta \vdash \tau_j \quad \text{for each } j \\ \color{red}{\text{no } \tau_j \text{ is an array type}}\end{array}}{\begin{array}{c}\Gamma;\Delta;\Theta \vdash (\mathtt{T\text{-}APP}\ l\ \tau\ \ldots) : \\ \sigma[(x \leftarrow_t \tau)\ \ldots]\end{array}}
$$

Types

$$
\begin{array}{rcl}
\tau, \sigma & ::= & x \\
& | & B \\
& | & (\tau \rightarrow \sigma) \\
& | & \mathtt{A}_\iota \, \tau \\
& | & (\forall \, [x \; \ldots] \; \tau) \\
& | & (\Pi \, [(x \; \gamma) \; \ldots] \; \tau) \\
& | & (\Sigma \, [(x \; \gamma) \; \ldots] \; \tau)
\end{array}
$$

$$
\begin{array}{lcl}
\Gamma & :: & \text{Type environment} \\
\Delta & :: & \text{Kind environment} \\
\Theta & :: & \text{Sort environment}
\end{array}
$$

$$
\frac{\begin{array}{c}
\Gamma; \Delta; \Theta \vdash e : \mathtt{A}_\iota \, (\sigma \; \ldots \; \rightarrow \tau) \\
\Gamma; \Delta; \Theta \vdash e'_j : \mathtt{A}_{\kappa_j} \sigma_j \quad \text{for each } j \\
\iota' = \mathit{Max} \, [\![\iota, \kappa \; \ldots]\!]
\end{array}}{\Gamma; \Delta; \Theta \vdash (e \; e' \; \ldots) : \mathtt{A}_{\iota'} \tau}
$$

$$
\frac{\begin{array}{c}
\Gamma; \Delta; \Theta \vdash l : (\forall \, [x \; \ldots] \; \sigma) \\
\Delta; \Theta \vdash \tau_j \quad \text{for each } j \\
\text{no } \tau_j \text{ is an array type}
\end{array}}{\begin{array}{c}\Gamma; \Delta; \Theta \vdash (\mathtt{T\text{-}APP} \; l \; \tau \; \ldots) : \\ \sigma[(x \leftarrow_t \tau) \; \ldots]\end{array}}
$$

See ESOP'14 paper for full type system

Type inference

Heavy annotation burden

Heavy annotation burden

Untyped:

```
(A [] [(λ ([f-l 0] [f-r 0] [f-j 0])
        (A [] [(λ [(x all) (y all)]
                 (f-j (f-l x) (f-r y)))]))])
```

Heavy annotation burden

Untyped:

```
(A [] [(λ ([f-l 0] [f-r 0] [f-j 0])
        (A [] [(λ [(x all) (y all)]
                 (f-j (f-l x) (f-r y)))]))])
```

Typed:

```
(I-λ [(s-li Shape) (s-lo Shape) (s-ri Shape) (s-ro Shape) (s-jo Shape)]
  (T-λ [t-li t-lo t-ri t-ro t-jo]
    (A [] [(λ ([f-l (Array (S) ((Array s-li t-li)
                                -> (Array s-lo t-lo)))]
               [f-r (Array (S) ((Array s-ri t-ri)
                                -> (Array s-ro t-ro)))]
               [f-j (Array (S) ((Array s-lo t-lo)
                                 (Array s-ro t-ro)
                                -> (Array s-jo t-jo)))])
             (A [] [(λ [(x (Array s-li t-li))
                        (y (Array s-ri t-ri))]
                      (f-j (f-l x) (f-r y)))]))])))
```

Index arguments matter

```
(λ ([M (Array s Int)]
    [N (Array s Int)])
  (append M N))
```

Index arguments matter

```
(... ((λ ([M (Array s Int)]
          [N (Array s Int)])
         (append M N))
       [[1 2]
        [3 4]]
       [[5 6]
        [7 8]]) ...)
```

Index arguments matter

```
(... ((λ ([M (Array s Int)]
          [N (Array s Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

```
(... ((λ ([M (Array (S 2) Int)]
          [N (Array (S 2) Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

Index arguments matter

```
(... ((λ ([M (Array s Int)]
          [N (Array s Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

```
(... ((λ ([M (Array (S 2) Int)]
          [N (Array (S 2) Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

```
(... ((λ ([M (Array (S 2 2) Int)]
          [N (Array (S 2 2) Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

Index arguments matter

```
(... ((λ ([M (Array s Int)]
          [N (Array s Int)])
        (append M N))
     [[1 2]
      [3 4]]
     [[5 6]
      [7 8]]) ...)
```

```
(... ((λ ([M (Array (S 2) Int)]
          [N (Array (S 2) Int)])
        (append M N))
     [[1 2]
      [3 4]]
     [[5 6]
      [7 8]]) ...)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

```
(... ((λ ([M (Array (S 2 2) Int)]
          [N (Array (S 2 2) Int)])
        (append M N))
     [[1 2]
      [3 4]]
     [[5 6]
      [7 8]]) ...)
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Index arguments matter

```
(... ((λ ([M (Array s Int)]
          [N (Array s Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

```
(... ((λ ([M (Array (S 2) Int)]
          [N (Array (S 2) Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

```
(... ((λ ([M (Array (S 2 2) Int)]
          [N (Array (S 2 2) Int)])
        (append M N))
      [[1 2]
       [3 4]]
      [[5 6]
       [7 8]]) ...)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Need type inference unlike Dependent ML:
elaboration, not just typability

Bidirectional typechecking

Distinguish "checking" types from "synthesizing" types

Bidirectional typechecking

Distinguish "checking" types from "synthesizing" types

$$\frac{\Gamma \vdash e_f {\uparrow} \sigma \to \tau \qquad \Gamma \vdash e_a {\downarrow} \sigma}{\Gamma \vdash (e_f\ e_a) {\uparrow} \tau} \quad (\textsc{Syn-App})$$

Bidirectional typechecking

Distinguish "checking" types from "synthesizing" types

$$\frac{\Gamma \vdash e_f {\uparrow} \sigma \to \tau \qquad \Gamma \vdash e_a {\downarrow} \sigma}{\Gamma \vdash (e_f\ e_a) {\uparrow} \tau} \quad (\textsc{Syn-App})$$

$$\frac{\Gamma \vdash e {\downarrow} \tau}{\Gamma \vdash (e : \tau) {\uparrow} \tau} \quad (\textsc{Syn-Ann})$$

Bidirectional typechecking

Distinguish "checking" types from "synthesizing" types

$$\frac{\Gamma \vdash e_f \uparrow \sigma \to \tau \qquad \Gamma \vdash e_a \downarrow \sigma}{\Gamma \vdash (e_f\ e_a) \uparrow \tau} \quad (\textsc{Syn-App})$$

$$\frac{\Gamma \vdash e \downarrow \tau}{\Gamma \vdash (e : \tau) \uparrow \tau} \quad (\textsc{Syn-Ann})$$

$$\frac{\Gamma \vdash e \uparrow \sigma \qquad \Gamma \vdash \sigma <: \tau}{\Gamma \vdash e \downarrow \tau} \quad (\textsc{Chk-Sub})$$

Bidirectional typechecking

Dunfield & Krishnaswami, ICFP'13

$\Gamma \vdash e \uparrow \tau$ $\Gamma \vdash e \downarrow \tau$

Bidirectional typechecking

Dunfield & Krishnaswami, ICFP'13

$\Gamma \vdash e \uparrow \tau$ $\quad\quad\quad\quad$ $\Gamma \vdash e \downarrow \tau$

$$\Gamma \vdash \sigma \bullet e \Rrightarrow \tau$$

Bidirectional typechecking

Dunfield & Krishnaswami, ICFP'13

$$\Gamma \vdash e \uparrow \tau \qquad\qquad \Gamma \vdash e \downarrow \tau$$

$$\Gamma \vdash \sigma \bullet e \Rrightarrow \tau$$

$$\ldots . \grave{t} \ldots . \vdash e \downarrow \tau$$

Bidirectional typechecking

Dunfield & Krishnaswami, ICFP'13

$$\Gamma \vdash e \uparrow \tau \qquad\qquad \Gamma \vdash e \downarrow \tau$$

$$\Gamma \vdash \sigma \bullet e \Rrightarrow \tau$$

$$\ldots.\grave{t}\ldots. \vdash e \downarrow \tau \dashv \ldots, \hat{t} = \texttt{Int}, \ldots$$

Bidirectional typechecking

Dunfield & Krishnaswami, ICFP'13

$$\Gamma \vdash e \uparrow \tau \dashv \Delta \qquad \Gamma \vdash e \downarrow \tau \dashv \Delta$$

$$\Gamma \vdash \sigma \bullet e \Rightarrow\!\!\!\!\Rightarrow \tau \dashv \Delta$$

$$\ldots.\hat{t}\ldots. \vdash e \downarrow \tau \dashv \ldots, \hat{t} = \texttt{Int}, \ldots$$

Theory of shapes

Remora needs solver for theory of array shapes

Theory of shapes

Remora needs solver for theory of array shapes

$$\iota, \kappa ::= (+\ \iota\ \kappa) \mid (\mathrm{S}\ \iota\ \ldots) \mid x \mid n$$

A shape is a list of naturals

Theory of shapes

Remora needs solver for theory of array shapes

$$\iota, \kappa ::= (+\ \iota\ \kappa)\ \mid\ (\mathrm{S}\ \iota\ \ldots)\ \mid x \mid n$$

A shape is a list of naturals

The theory of shapes is the theory of the free monoid on $\mathbb{N}$

Theory of shapes

Remora needs solver for theory of array shapes

$$\iota, \kappa ::= (+ \ \iota \ \kappa) \ \mid \ (\mathrm{S} \ \iota \ \ldots) \ \mid x \mid n$$

A shape is a list of naturals

The theory of shapes is the theory of the free monoid on $\mathbb{N}$

$$a +\!\!+ (b +\!\!+ c) = (a +\!\!+ b) +\!\!+ c$$

Theory of shapes

Remora needs solver for theory of array shapes

$$\iota, \kappa ::= (+\ \iota\ \kappa)\ \mid\ (\mathrm{S}\ \iota\ \ldots)\ \mid x \mid n$$

A shape is a list of naturals

The theory of shapes is the theory of the free monoid on $\mathbb{N}$

$$a \mathbin{+\!\!\!+} (b \mathbin{+\!\!\!+} c) = (a \mathbin{+\!\!\!+} b) \mathbin{+\!\!\!+} c$$

$$\square \mathbin{+\!\!\!+} a = a \mathbin{+\!\!\!+} \square = a$$

Theory of shapes

Remora needs solver for theory of array shapes

$$\iota, \kappa ::= (+\ \iota\ \kappa)\ \mid\ (\mathrm{S}\ \iota\ \ldots)\ \mid x \mid n$$

A shape is a list of naturals

The theory of shapes is the theory of the free monoid on $\mathbb{N}$

$$a +\!\!+ (b +\!\!+ c) = (a +\!\!+ b) +\!\!+ c$$

$$\square +\!\!+ a = a +\!\!+ \square = a$$

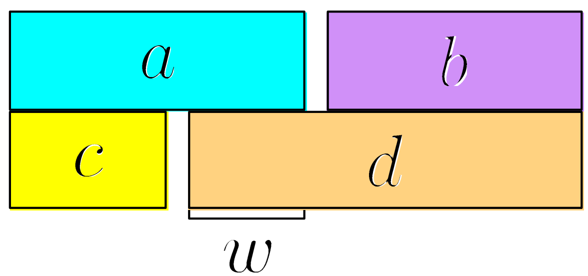
$$\begin{aligned} a +\!\!+ b = c +\!\!+ d \implies \exists w.\ & (a +\!\!+ w = c \wedge w +\!\!+ d = b) \\ \vee\ & (c +\!\!+ w = a \wedge w +\!\!+ b = d) \end{aligned}$$

Theory of shapes

$$a +\!\!\!+ (b +\!\!\!+ c) = (a +\!\!\!+ b) +\!\!\!+ c$$

$$\square +\!\!\!+ a = a +\!\!\!+ \square = a$$

$$\begin{aligned} a +\!\!\!+ b = c +\!\!\!+ d \implies & \exists w. (a +\!\!\!+ w = c \wedge w +\!\!\!+ d = b) \\ & \vee (c +\!\!\!+ w = a \wedge w +\!\!\!+ b = d) \end{aligned}$$



Work to do

Type inference

- local type inference for Dependent ML
- (semi)decision procedure for free monoid
- extend DML type inference to Remora

Compiling rank-polymorphic programs

Compilation

```
(+ [10 20]
   [[1 2 3]
    [4 5 6]])
```

Compilation

```
(+ [10 20]
   [[1 2 3]
    [4 5 6]])
```

```
(+ [[10 10 10]
    [20 20 20]]
   [[ 1  2  3]
    [ 4  5  6]])
```

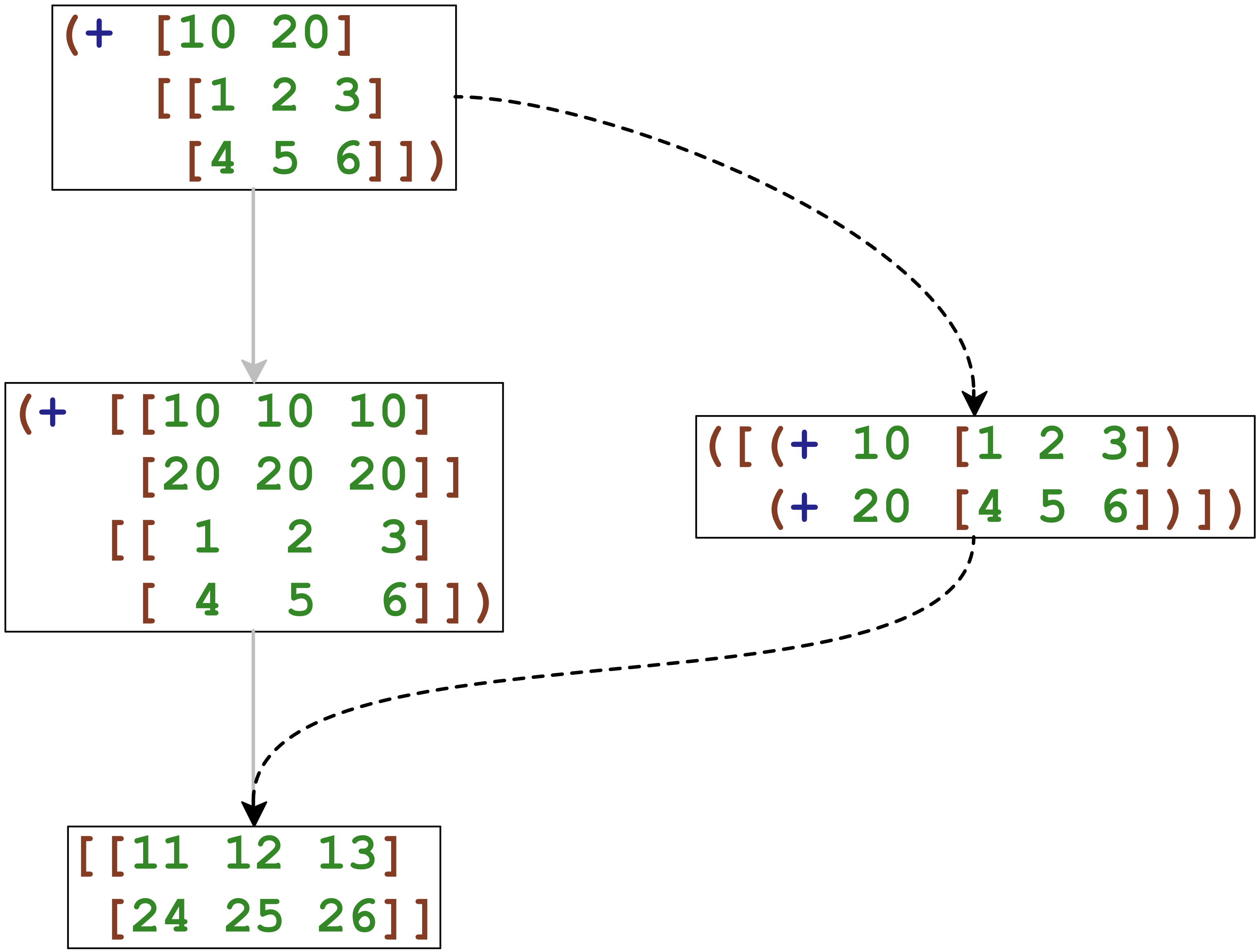
Compilation

```
(+ [10 20]
   [[1 2 3]
    [4 5 6]])
```

```
(+ [[10 10 10]
    [20 20 20]]
   [[ 1  2  3]
    [ 4  5  6]])
```

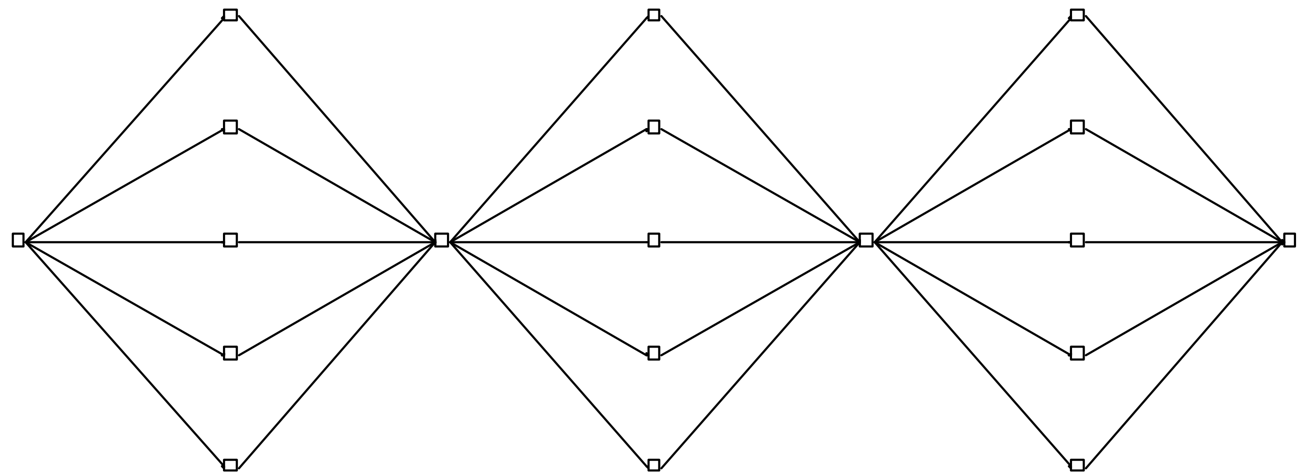
```
[[11 12 13]
 [24 25 26]]
```

Compilation



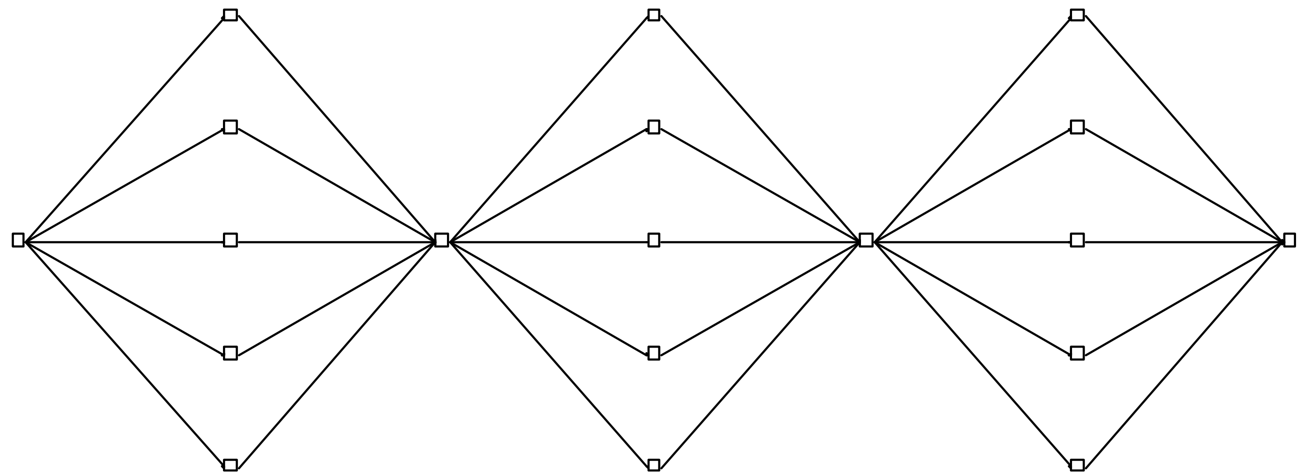
Compilation

Naive compilation

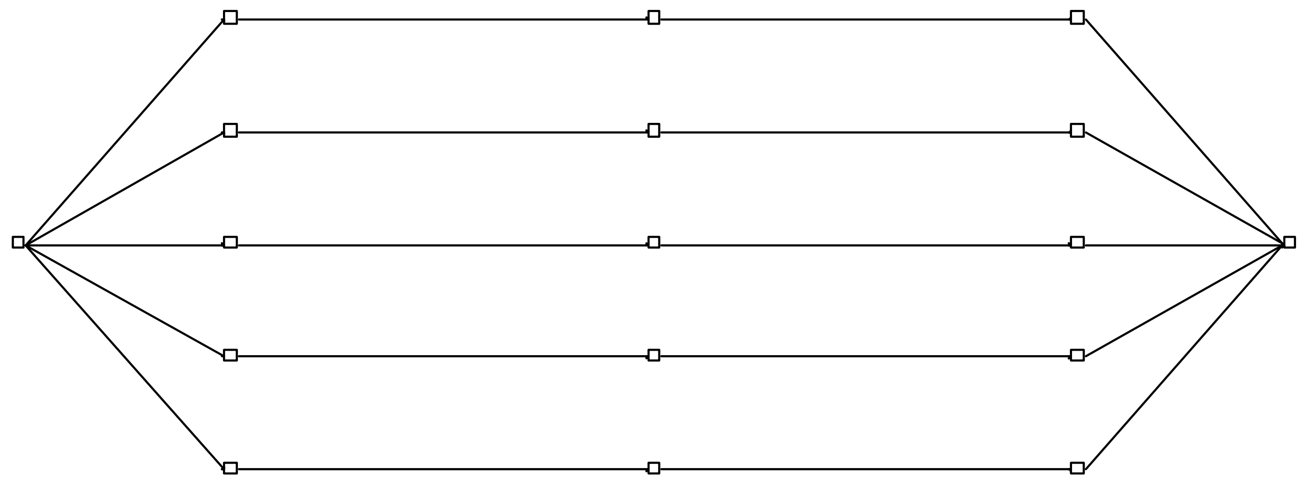


Compilation

Naive compilation

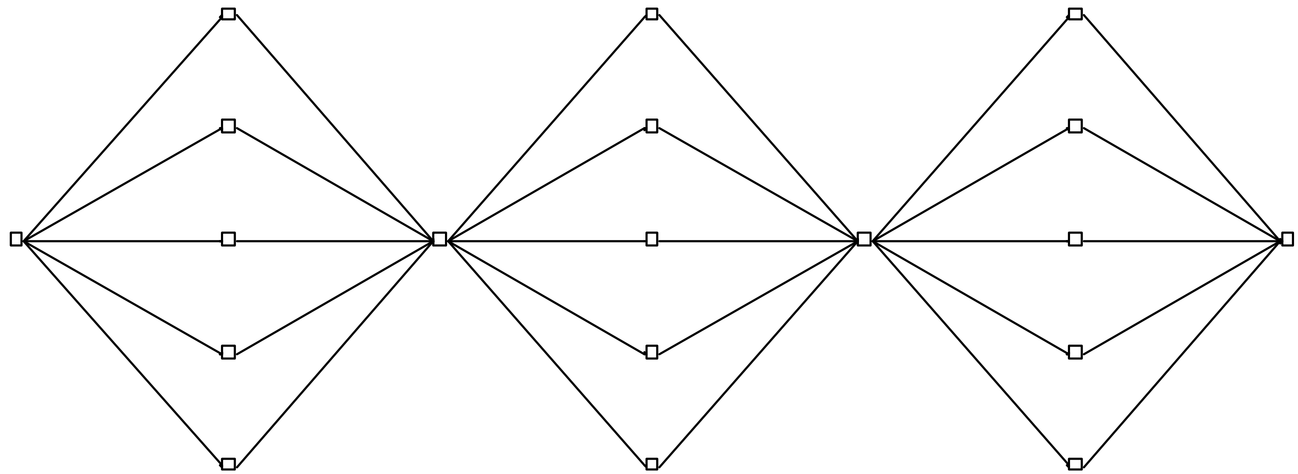


Preferable

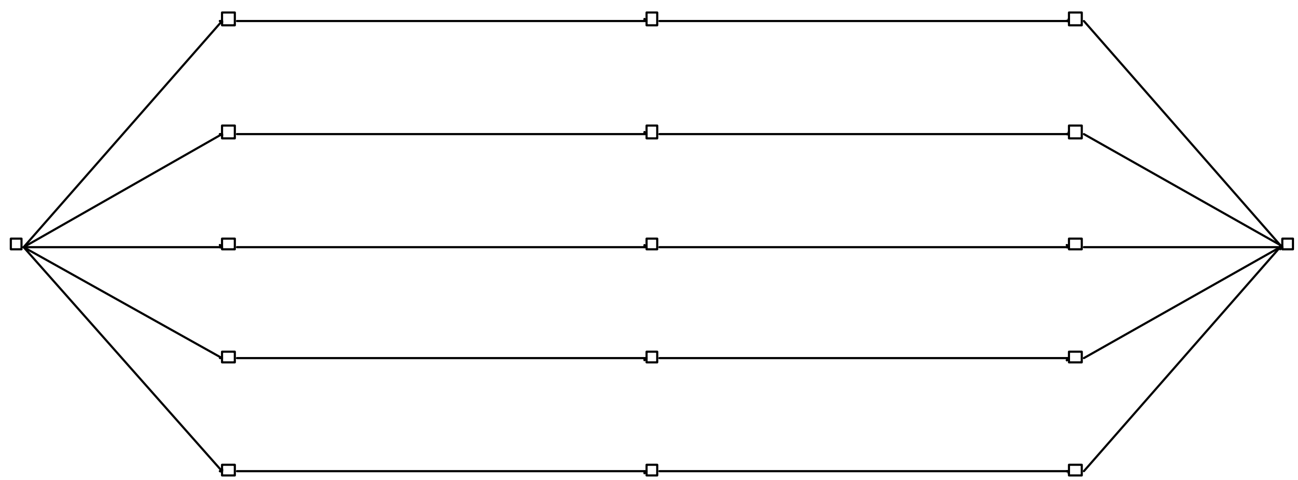


Compilation

Naive compilation



Preferable



Scatter early, gather late

Work to do

Type inference

- local type inference for Dependent ML
- (semi)decision procedure for free monoid
- extend DML type inference to Remora

Term calculus

- general β → early scattering
- delayed collapse → late gathering
- revised metatheory → confluence, type soundness

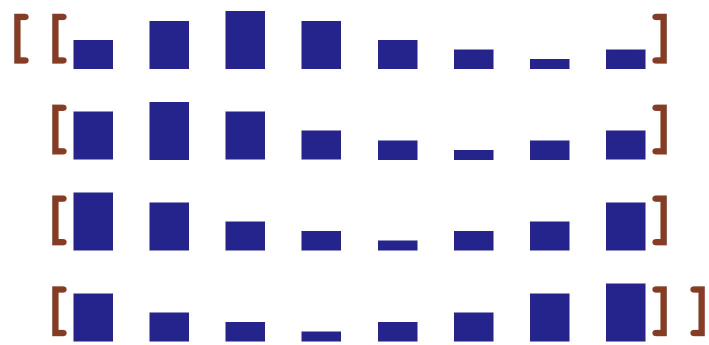
Lower-level concerns

```
(rotate
 signal
 (iota [(length filter)]))
```

Lower-level concerns

```
(rotate
 signal
 (iota [(length filter)]))
```

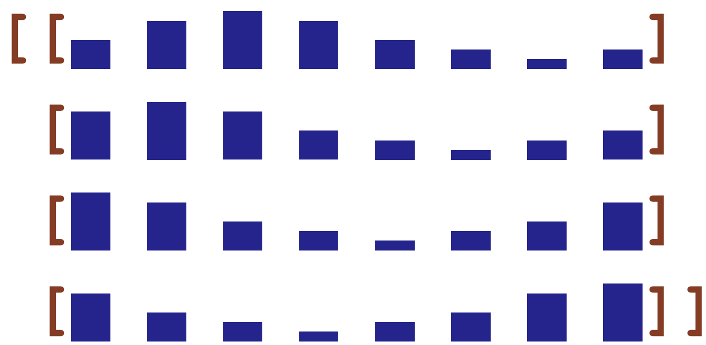
↓



Lower-level concerns

```
(rotate
 signal
 (iota [(length filter)]))
```

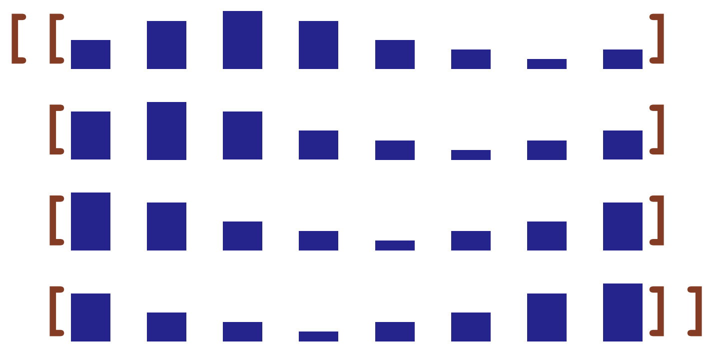
↓



Lower-level concerns

```
(rotate
 signal
 (iota [(length filter)]))
```

↓



Outside the scope of β

```
[0 1 2 3 4 5]
```

[0 1 2 3 4 5]

[2 3 4 5 0 1]

rotate

`[0 1 2 3 4 5]`

`[2 3 4 5 0 1]` rotate

`[1 2 3]` slice

`[0 1 2 3 4 5]`

`[2 3 4 5 0 1]`	rotate
`[1 2 3]`	slice
`[5 4 3 2 1 0]`	reverse

```
[0 1 2 3 4 5]
```

`[2 3 4 5 0 1]`	rotate
`[1 2 3]`	slice
`[5 4 3 2 1 0]`	reverse
`[[0 1 2]` ` [3 4 5]]`	reshape

[0 1 2 3 4 5]

[2 3 4 5 0 1]	rotate
[1 2 3]	slice
[5 4 3 2 1 0]	reverse
[[0 1 2] [3 4 5]]	reshape
[[0 1 2 3 4 5] [0 1 2 3 4 5]]	replicate

```
[0 1 2 3 4 5]
```

```
[2 3 4 5 0 1]
```
rotate

```
[1 2 3]
```
slice

```
[5 4 3 2 1 0]
```
reverse

```
[[0 1 2]
 [3 4 5]]
```
reshape

```
[[0 1 2 3 4 5]
 [0 1 2 3 4 5]]
```
replicate

Don't copy data, just change index calculation

```
[0 1 2 3 4 5]

[2 3 4 5 0 1]      (λ (x) (mod (+ x 2) 6))

[1 2 3]            (λ (x) (+ x 1))

[5 4 3 2 1 0]      (λ (x) (- 6 x))

[[0 1 2]           (λ (x0 x1)
 [3 4 5]]            (+ (* 3 x0) x1))

[[0 1 2 3 4 5]
 [0 1 2 3 4 5]]    (λ (x0 x1) x1)
```

Don't copy data, just change index calculation

"Starter" compiler

Serial (targetting LLVM or C)

"Starter" compiler

Serial (targetting LLVM or C)

- eventual parallel compiler still needs serial "fallback" code generator
- keeps hardware options open (CUDA, NVPTX, LambdaJIT, etc.)

"Starter" compiler

Serial (targetting LLVM or C)

- eventual parallel compiler still needs serial "fallback" code generator
- keeps hardware options open (CUDA, NVPTX, LambdaJIT, etc.)

GPU (targetting Futhark)

"Starter" compiler

Serial (targetting LLVM or C)

- eventual parallel compiler still needs serial "fallback" code generator
- keeps hardware options open (CUDA, NVPTX, LambdaJIT, etc.)

GPU (targetting Futhark)

- intended as compilation target
- actual future suitability not guaranteed

Work to do

Type inference

- local type inference for Dependent ML
- (semi)decision procedure for free monoid
- extend DML type inference to Remora

Term calculus

- general β → early scattering
- delayed collapse → late gathering
- revised metatheory → confluence, type soundness

Compiler (LLVM/C/Futhark)

- IR for allocation/sharing
- code transformation to avoid large intermediates

Work to do

Type inference

- local type inference for Dependent ML
- (semi)decision procedure for free monoid
- extend DML type inference to Remora

Evaluation: can we infer types for existing Remora code?

Term calculus

- general β → early scattering
- delayed collapse → late gathering
- revised metatheory → confluence, type soundness

Compiler (LLVM/C/Futhark)

- IR for allocation/sharing
- code transformation to avoid large intermediates

Evaluation: time/memory vs "von Neumann-style" code

Work to do

Type inference

- local type inference for Dependent ML
- (semi)decision procedure for free monoid
- extend DML type inference to Remora

Evaluation: can we infer types for existing Remora code?

Term calculus

- general β → early scattering
- delayed collapse → late gathering
- revised metatheory → confluence, type soundness

Compiler (LLVM/C/Futhark)

- IR for allocation/sharing
- code transformation to avoid large intermediates

Evaluation: time/memory vs "von Neumann-style" code

Writing

Schedule

12 months

Type inference

- local type inference for Dependent ML
- (semi)decision procedure for free monoid
- extend DML type inference to Remora

Evaluation: can we infer types for existing Remora code?

6 months

Term calculus

- general β → early scattering
- delayed collapse → late gathering
- revised metatheory → confluence, type soundness

Compiler (LLVM/C/Futhark)

- IR for allocation/sharing
- code transformation to avoid large intermediates

Evaluation: time/memory vs "von Neumann-style" code

6 months

Writing