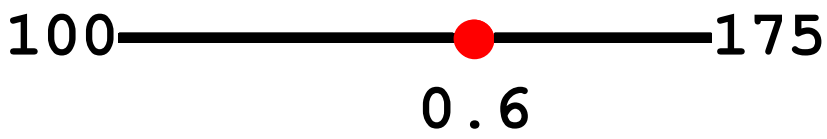
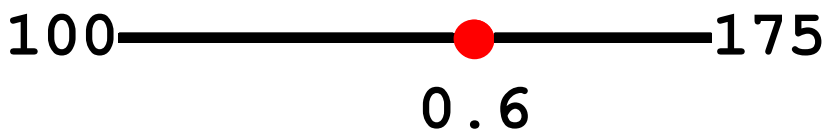
An Array-Oriented Language with Static Rank Polymorphism

Justin Slepak, Olin Shivers, Panagiotis Manolios

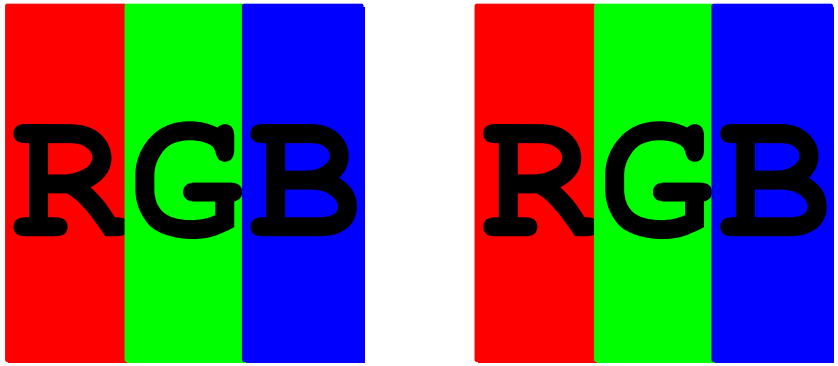
Northeastern University



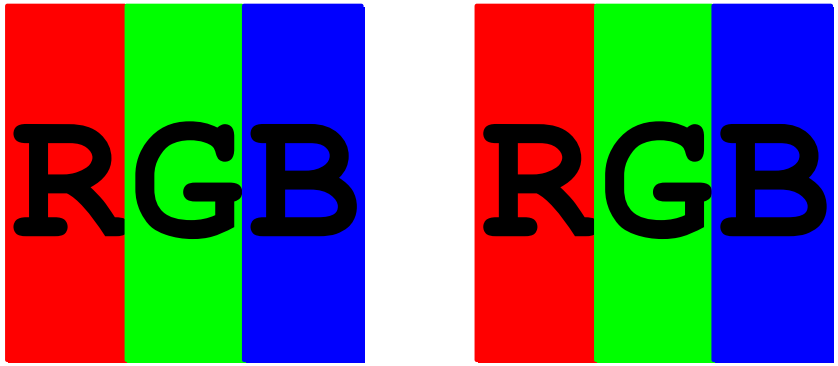

```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```



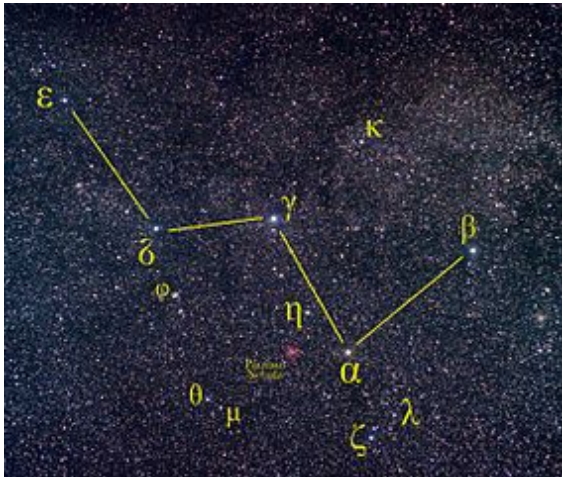

```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 100
 175
 0.6)
```




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

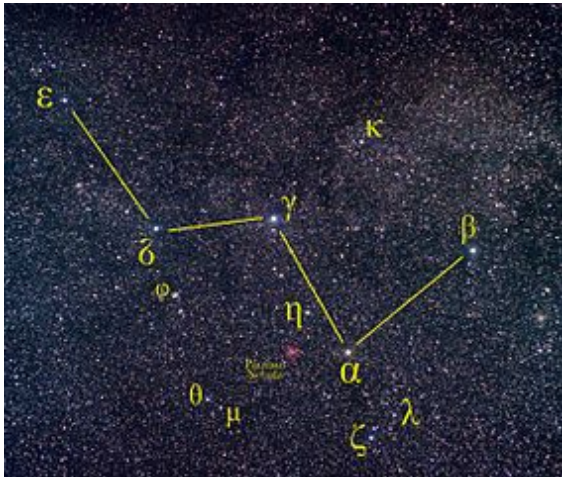



```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 rgb1 ; 3 channels
 rgb2 ; 3 channels
 0.6) ; scalar
```



Credit: Wikimedia user Sadalsuud

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



Credit: Wikimedia user Sadalsuud

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

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

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

Credit: Wikimedia user Thetawave

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

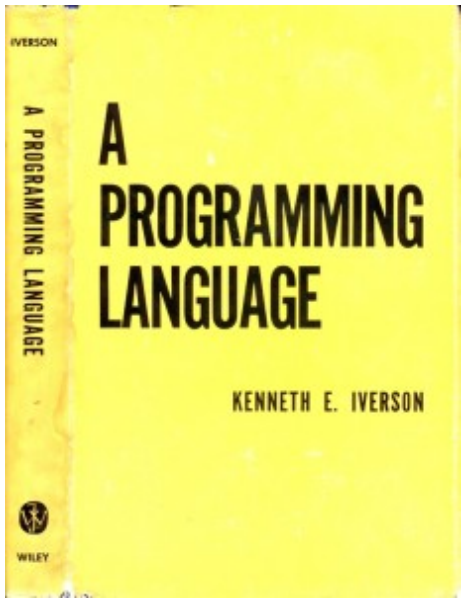
Credit: Wikimedia user Thetawave

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

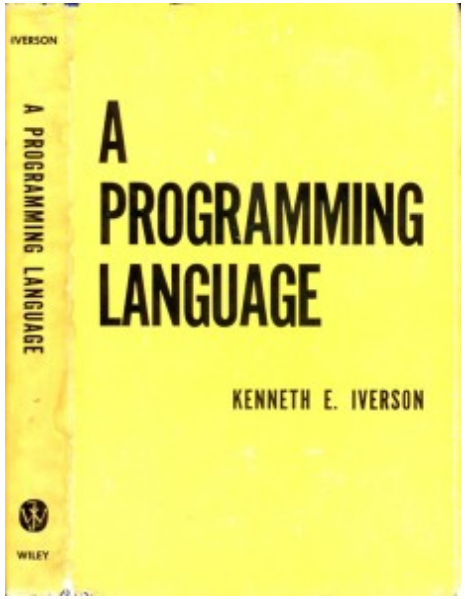
```
(λ [(lo 0) (hi 0) (α 0)]
  (+ (* lo α) (* hi (- 1 α))))
```

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

Polymorphic in dimensionality



All operations are aggregate operations



All operations are aggregate operations

Control structure generated by polymorphism

Problem: Compilation depends on control structure

Problem: Compilation depends on control structure

Solution: Static semantics

Overview of rank-polymorphic programming model

Overview of rank-polymorphic programming model

Dynamic semantics

Programming technique

Overview of rank-polymorphic programming model

Dynamic semantics

Programming technique

Remora: typed rank-polymorphic core language

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

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

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

$$[0].$$

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

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

$$[0].$$

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

$$[0].$$

Programming model

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

atoms: non-aggregate elements

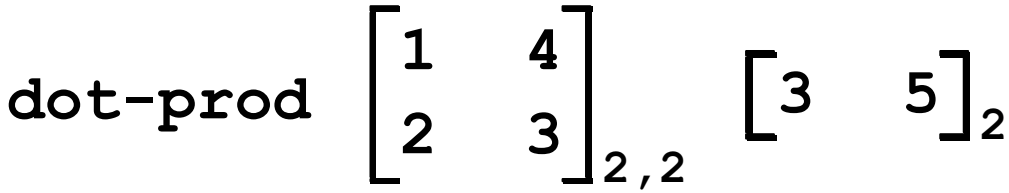
$$\left[\begin{array}{cc} 0 & 1 \\ 2 & 3 \\ \hline 4 & 5 \\ 6 & 7 \end{array}\right]_{2,2,2} \quad 3$$

shape: size in each dimension

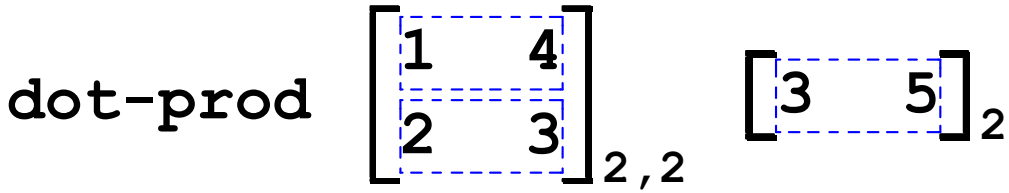
$$[0]. \quad 0$$

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

Programming model

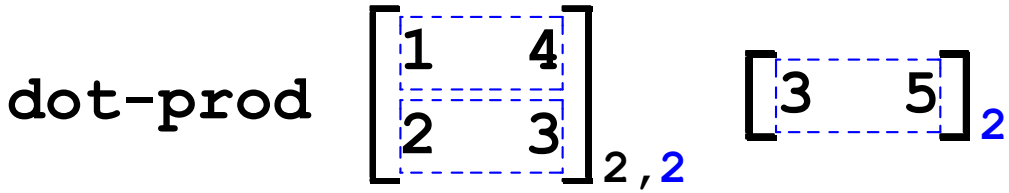


Programming model



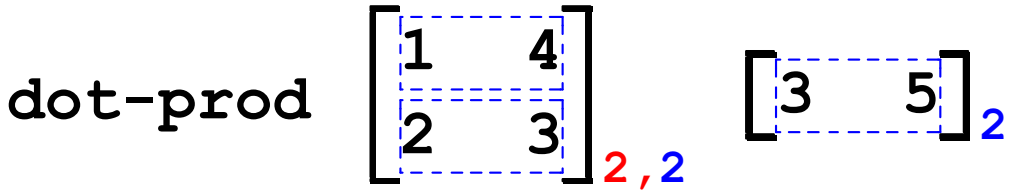
cell: basic unit function operates on

Programming model



cell: basic unit function operates on

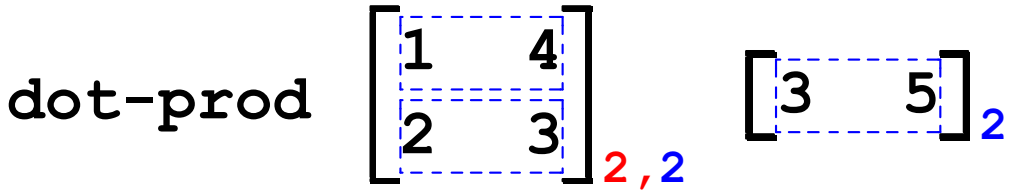
Programming model



cell: basic unit function operates on

frame: structure around cells

Programming model

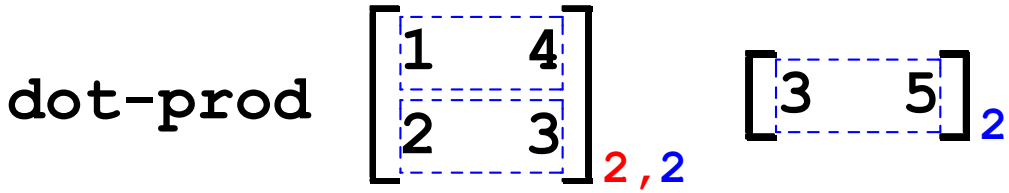


cell: basic unit function operates on

frame: structure around cells

function applied to each cell

Programming model



cell: basic unit function operates on

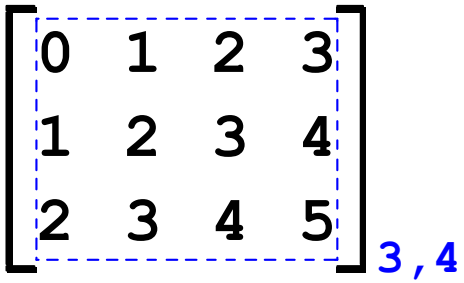
frame: structure around cells

function applied to each cell

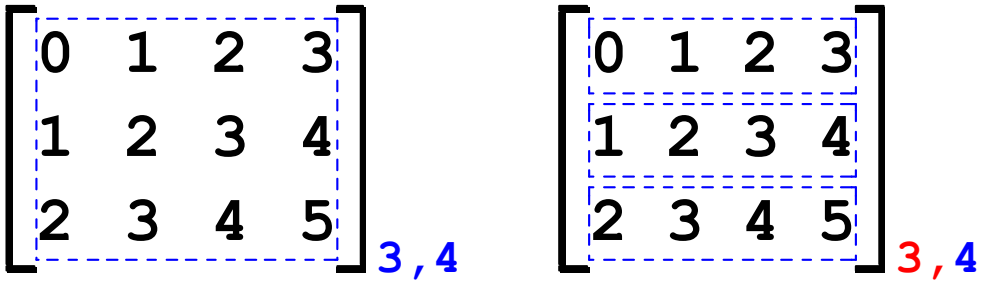
results reassembled in frame

Rank n array can split n+1 ways

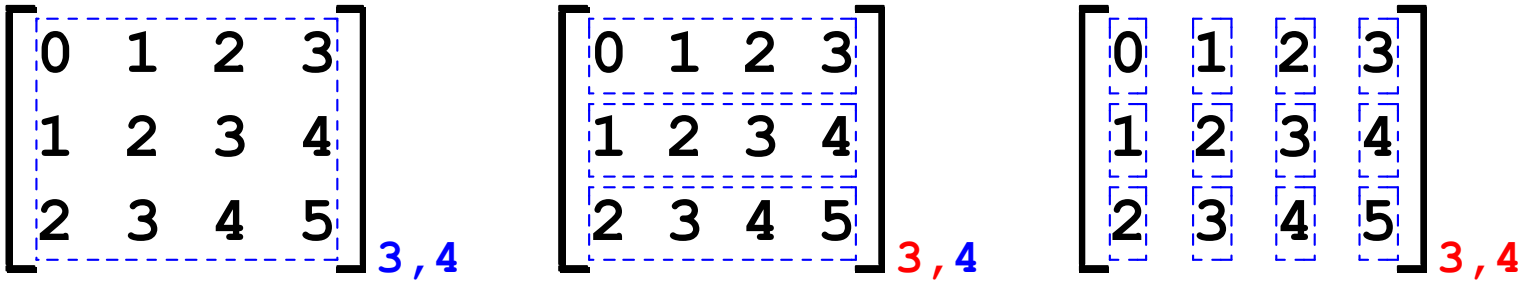
Rank n array can split n+1 ways



Rank n array can split n+1 ways



Rank n array can split n+1 ways

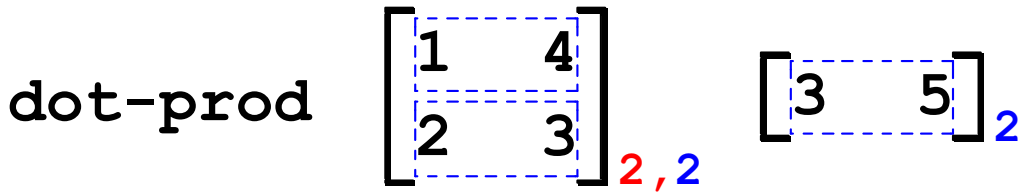


Programming model

Frame-of-cells split chosen by function

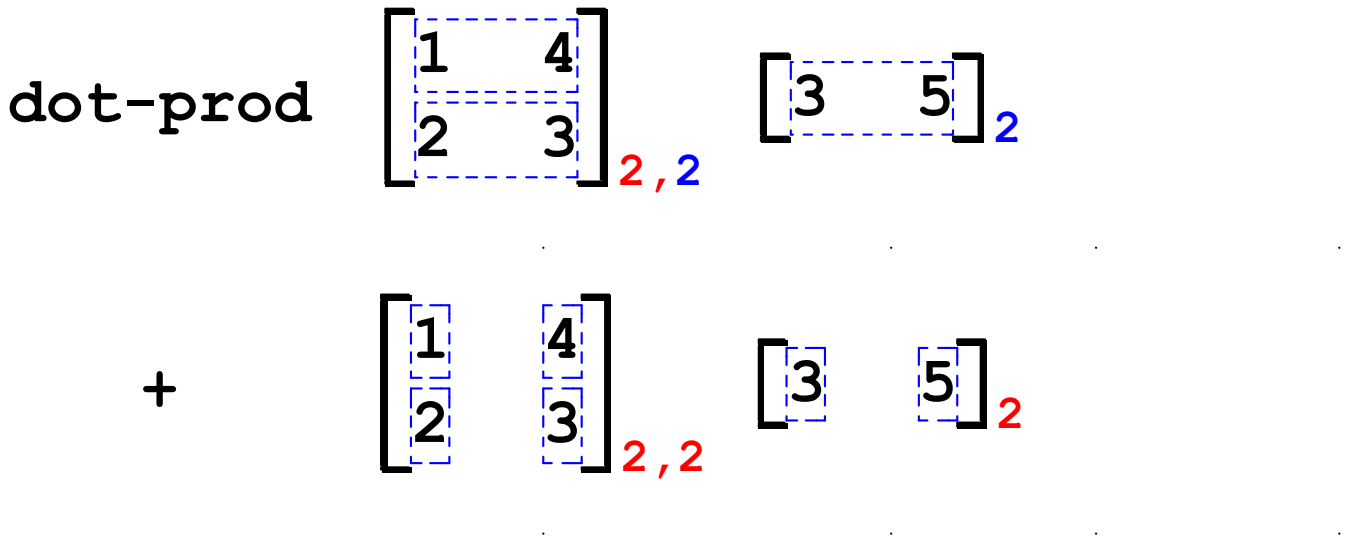
Programming model

Frame-of-cells split chosen by function



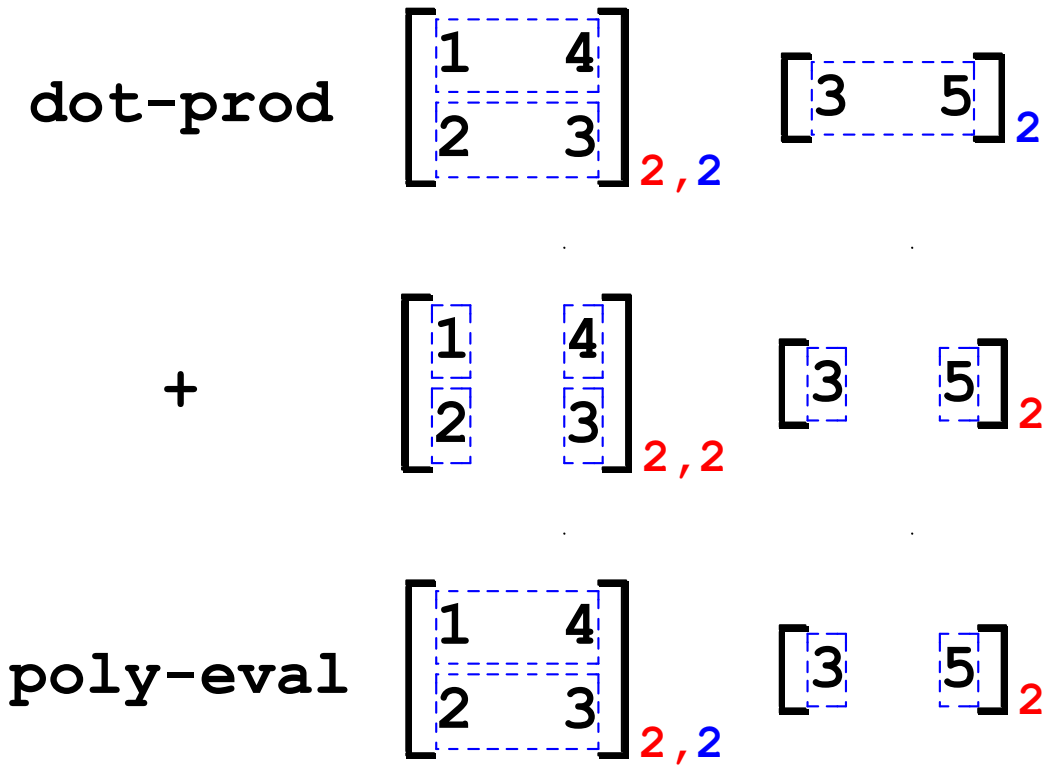
Programming model

Frame-of-cells split chosen by function



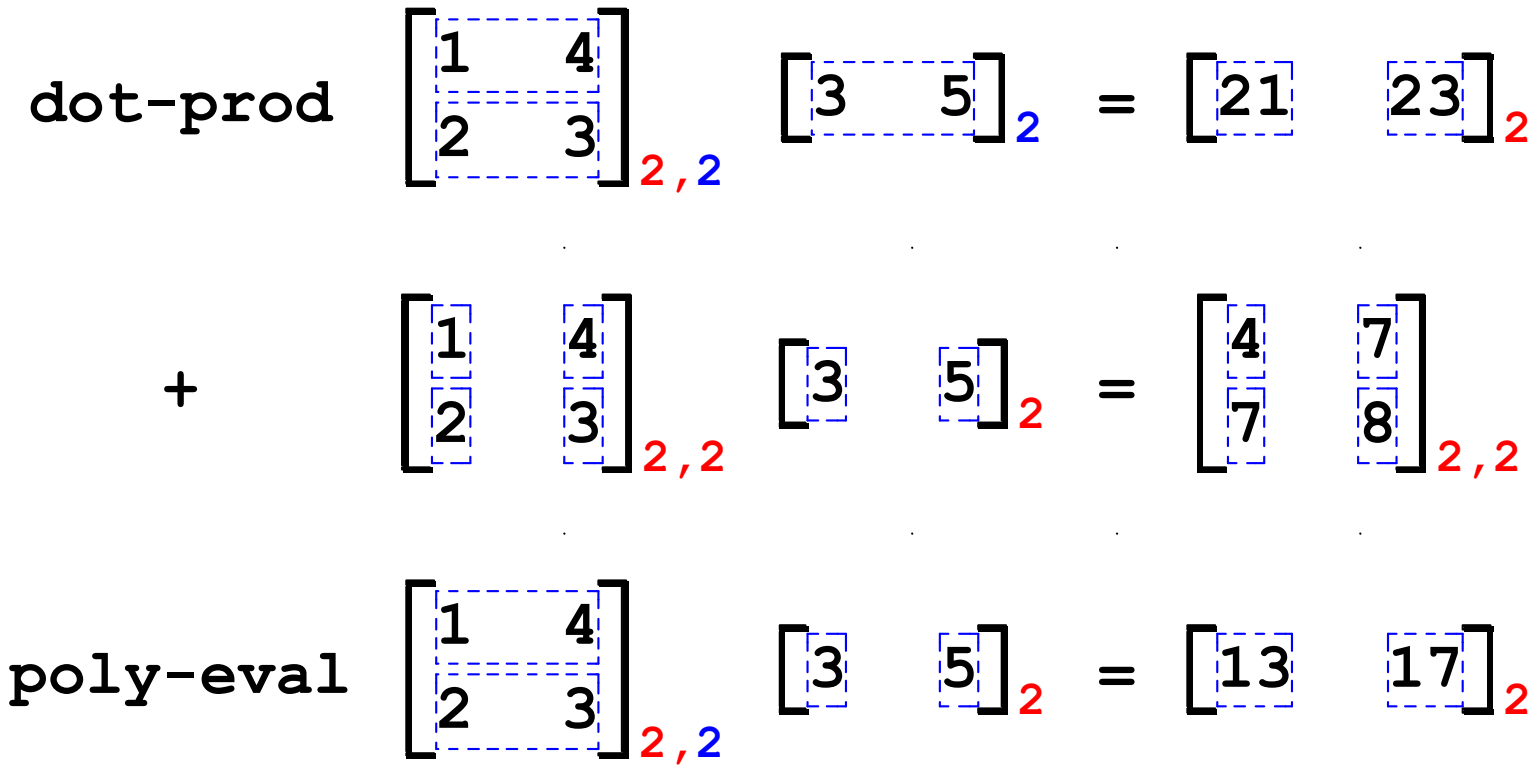
Programming model

Frame-of-cells split chosen by function

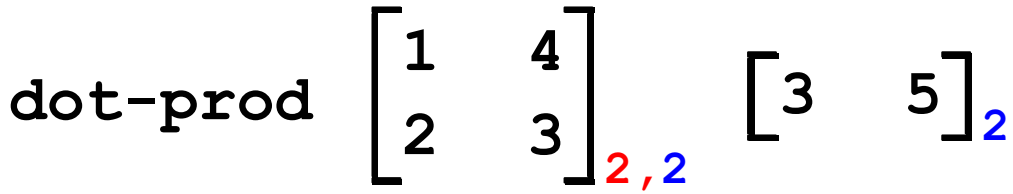


Programming model

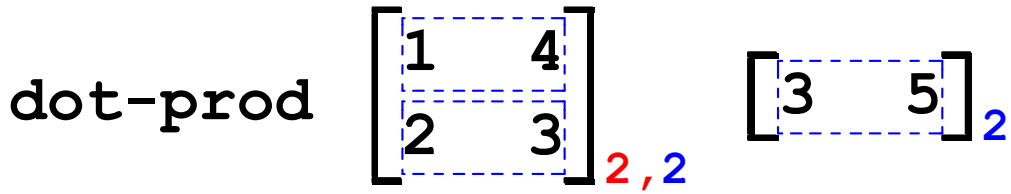
Frame-of-cells split chosen by function



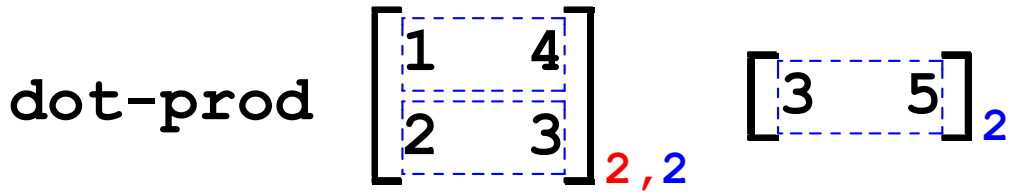
Evaluation



Evaluation

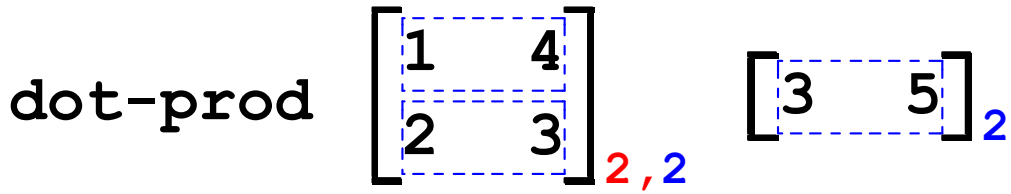


Evaluation



2 2 • 2

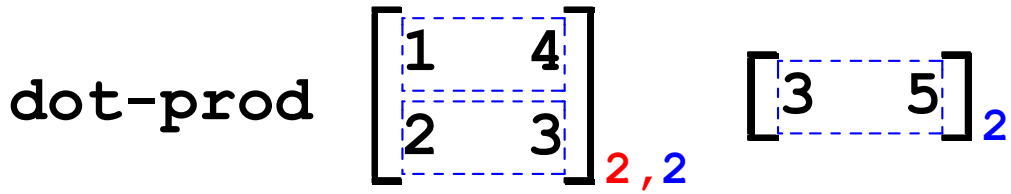
Evaluation



2 2 • 2

• ⊑ 2

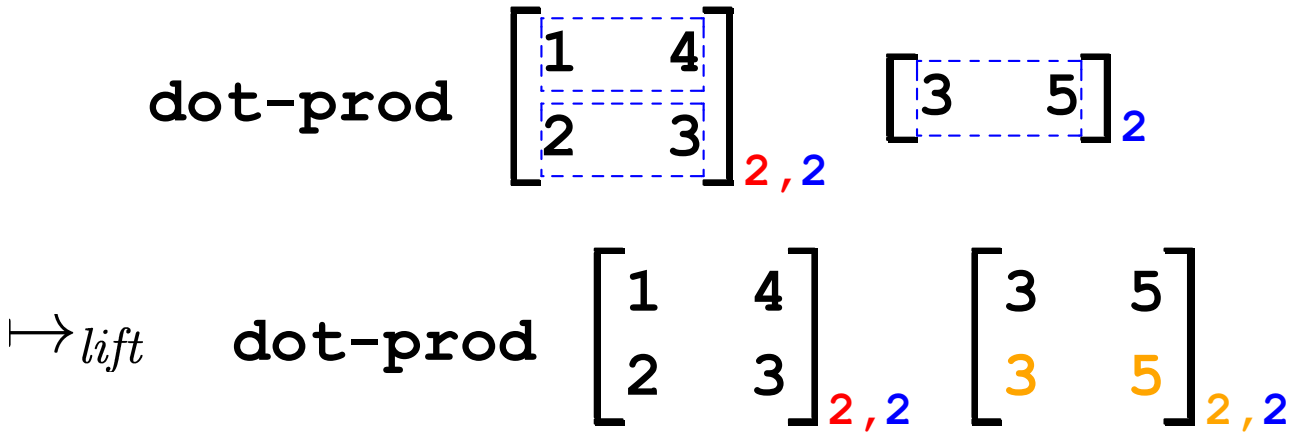
Evaluation



2 2 • 2

• ⊑ 2 principal frame

Evaluation



Raise into principal frame by replication

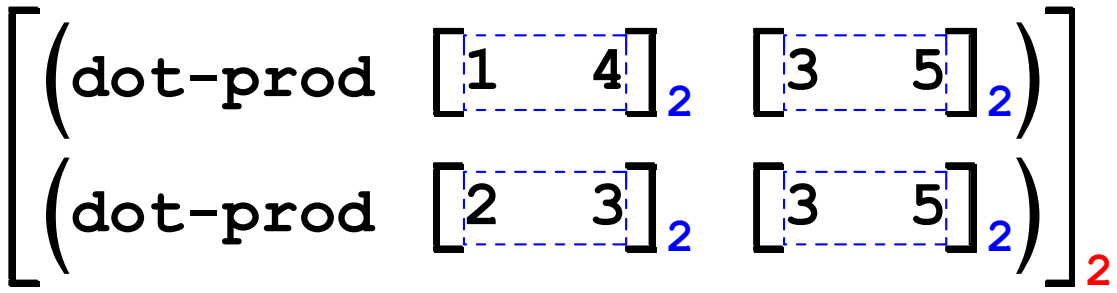
Evaluation

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

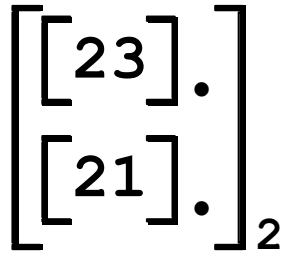
$$\mapsto_{map}\ \begin{bmatrix}(\texttt{dot-prod}\ \texttt{[1 4]}_{\texttt{2}}\ \texttt{[3 5]}_{\texttt{2}})\\ (\texttt{dot-prod}\ \texttt{[2 3]}_{\texttt{2}}\ \texttt{[3 5]}_{\texttt{2}})\end{bmatrix}_{\texttt{2}}$$

Apply function cell-wise

Evaluation



$$\mapsto_{\delta^2}$$



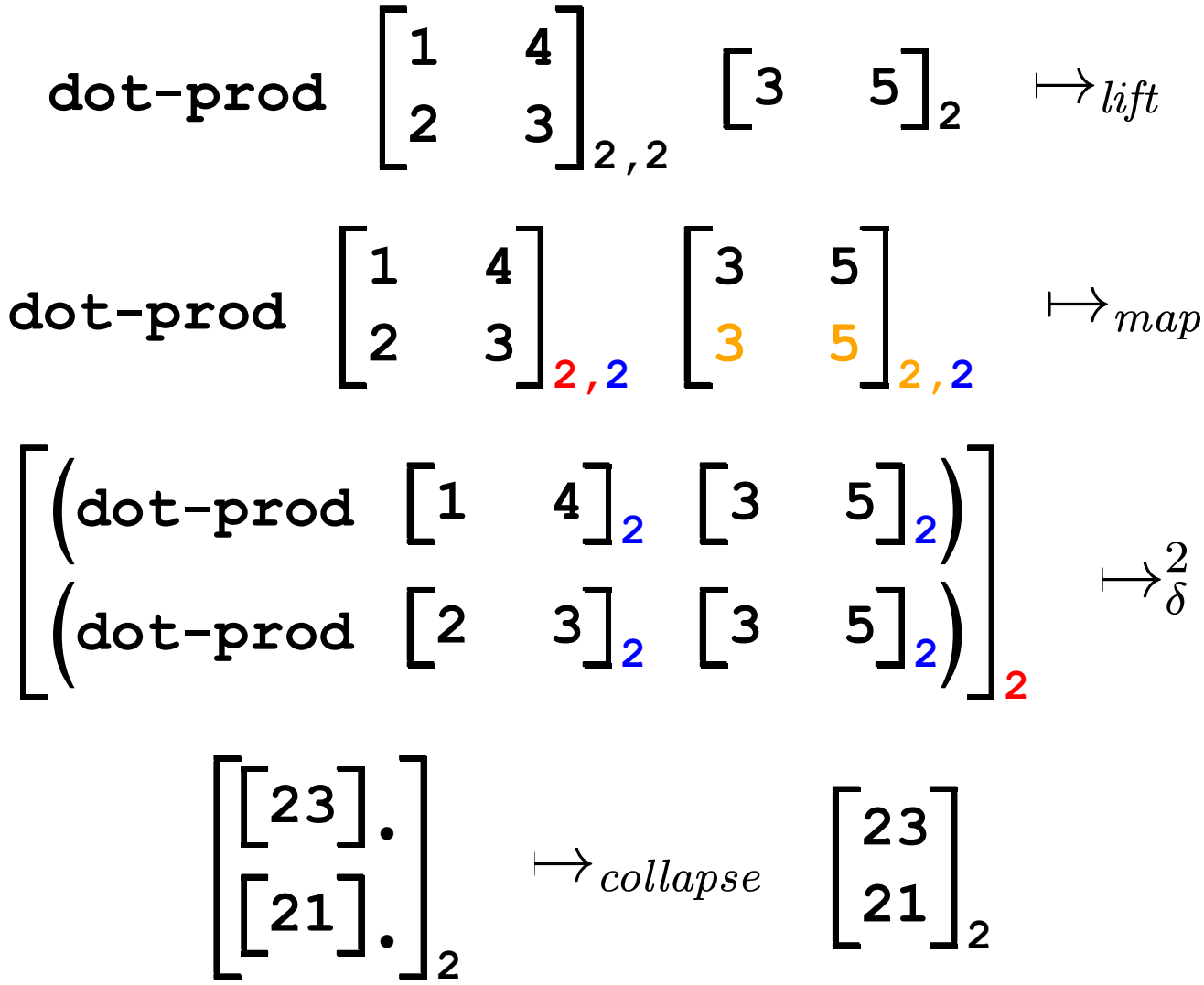
Compute each result cell

Evaluation

$$\left[\left[23\right].\left[21\right].\right]_2$$

$$\mapsto_{collapse} \left[\begin{matrix}23\\21\end{matrix}\right]_2$$

Collapse frame-of-cells to single array

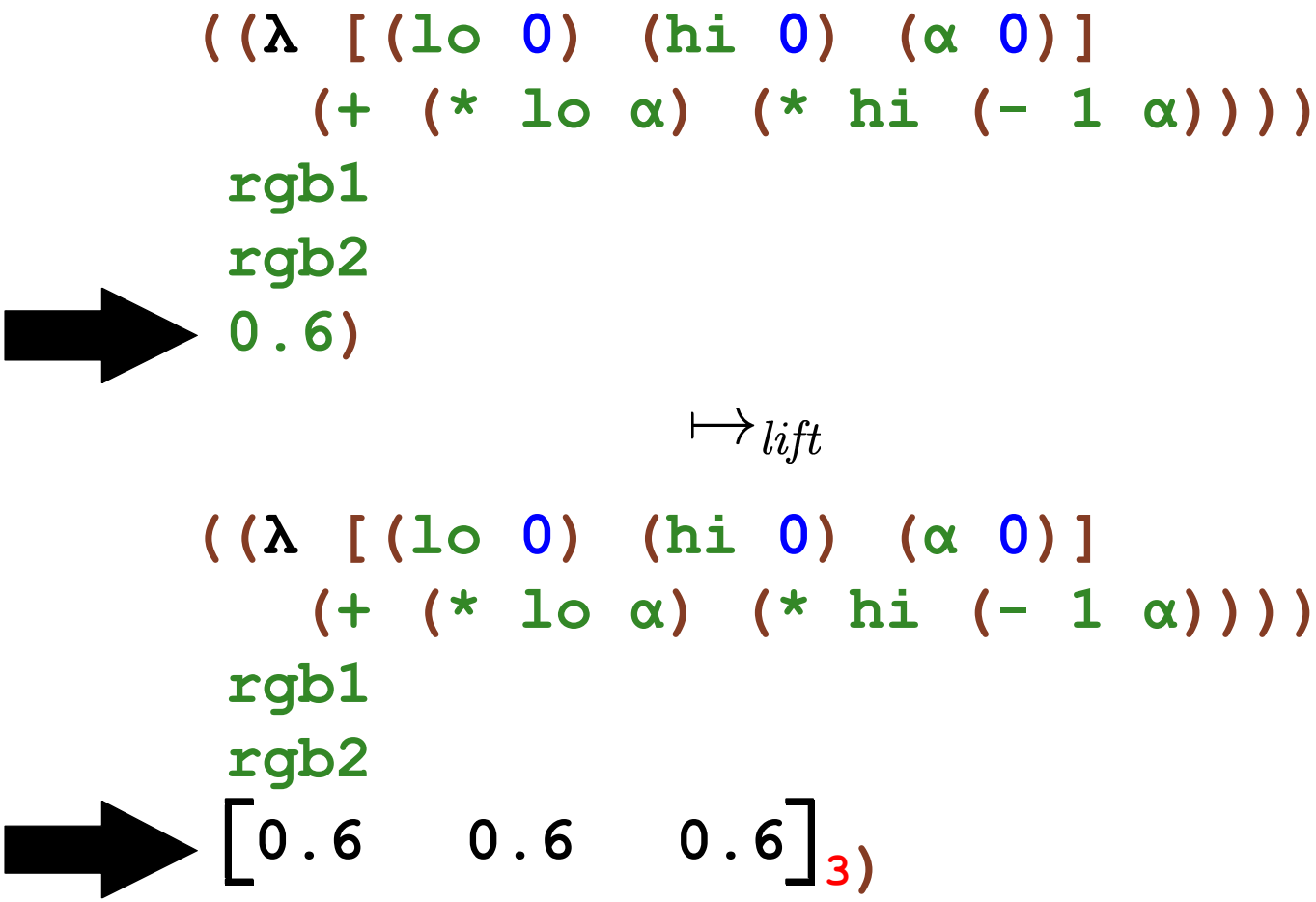

dot-prod [1 4; 2 3]2,2 [3 5]2 ↦lift
dot-prod [1 4; 2 3]2,2 [3 5; 3 5]2,2 ↦map
[(dot-prod [1 4]2 [3 5]2); (dot-prod [2 3]2 [3 5]2)]2 ↦δ2
[[23].; [21].]2 ↦collapse [23; 21]2

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

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

$\mapsto_{lift}$

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


((λ [(lo 0) (hi 0) (α 0)]
(+ (* lo α) (* hi (- 1 α))))
rgb1
rgb2
0.6)
↦lift
((λ [(lo 0) (hi 0) (α 0)]
(+ (* lo α) (* hi (- 1 α))))
rgb1
rgb2
[0.6 0.6 0.6]3)

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

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

$\longmapsto_{lift}$

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

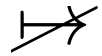
```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1     ; time × row × col × chan
 scene2     ; time × row × col × chan
 ; time
 [0.0 … 1.0])
```

$\longmapsto_{lift}$

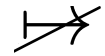
```
((λ [(lo 0) (hi 0) (α 0)]
   (+ (* lo α) (* hi (- 1 α))))
 scene1
 scene2
 ; time × row × col × chan
 [0.0 … 0.0 … 1.0 … 1.0])
```

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

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



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



Frame mismatch: need to align arguments differently

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$+^{1,1}$

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

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

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$+^{1,1}$

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

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$+^{1,1}$

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

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

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

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

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

Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

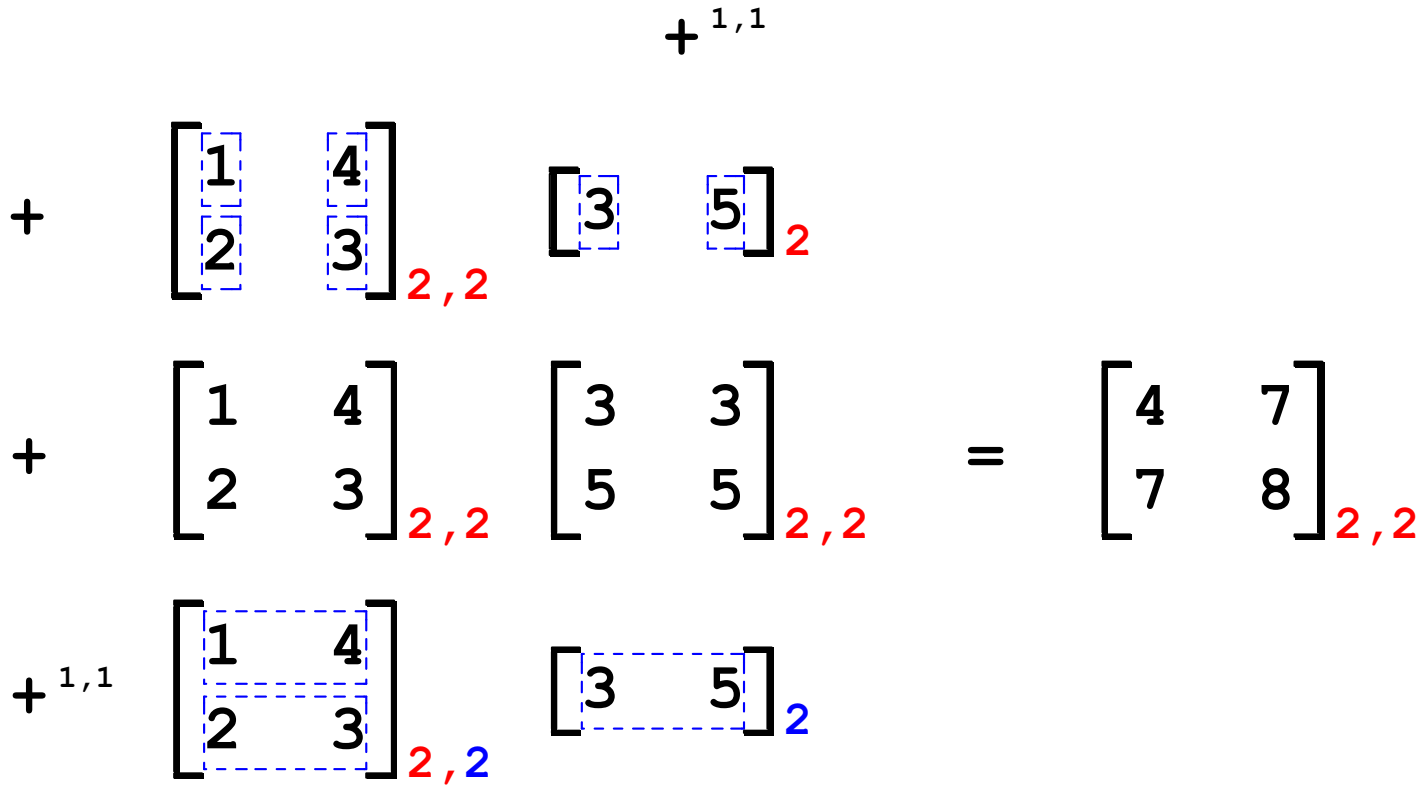
$$+ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_{2}$$

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

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

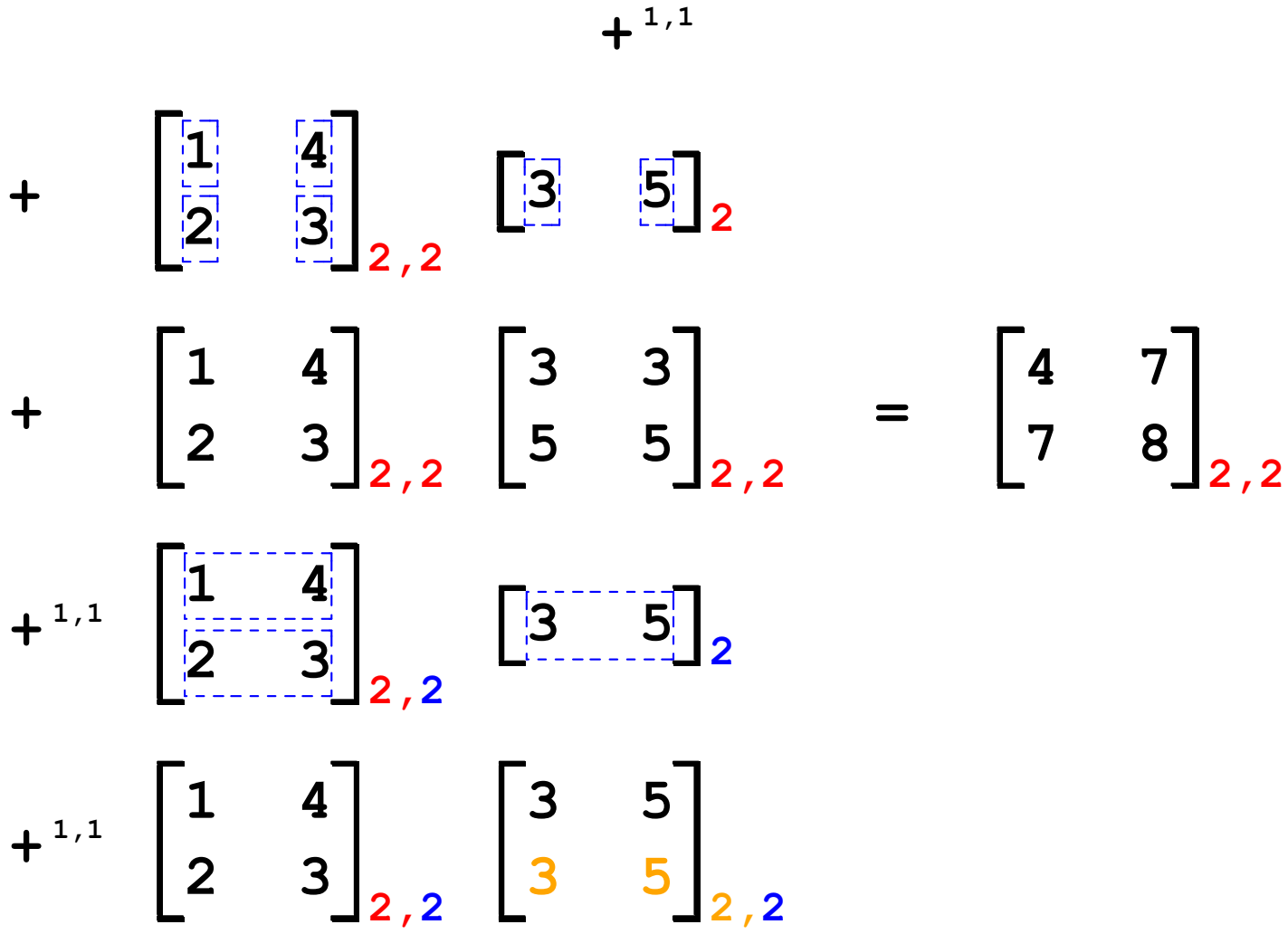
Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```



Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```



Reranking

```
(λ [(x 1) (y 1)] (+ x y))
```

$$+^{1,1}$$

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

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

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

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

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

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

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

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

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

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

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


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

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

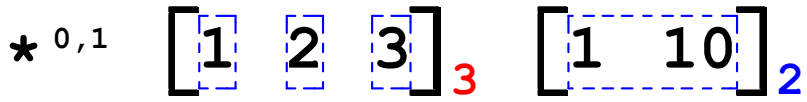
```
((λ [(lo 1) (hi 1) (α 0)]
   (lerp lo hi α))
 photo  ; row × col × chan
 rgb    ; chan
 0.6)
```

```
((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
```

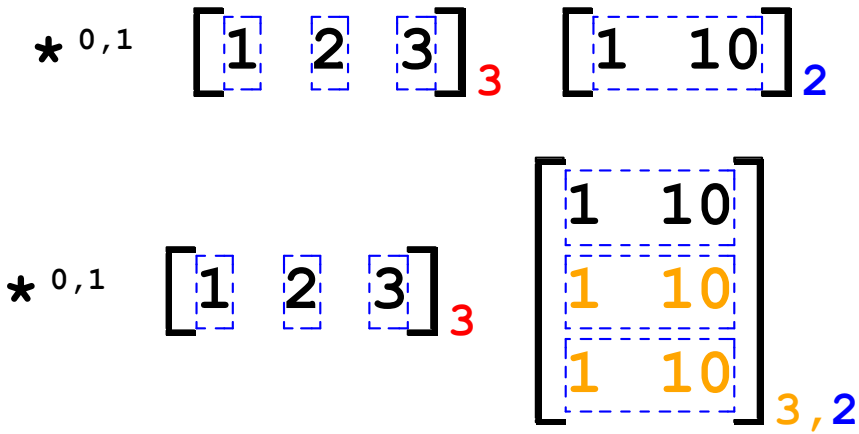
Axis alignment

$$*^{0,1}\ [1\ \ 2\ \ 3]_3\ \ [1\ \ 10]_2$$

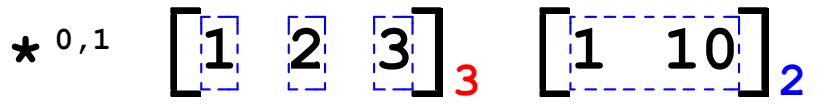
Axis alignment

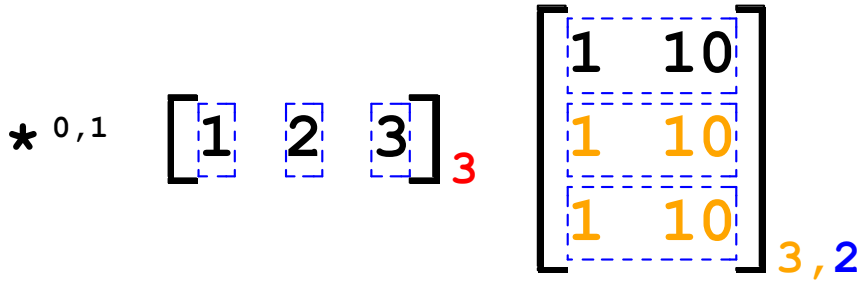


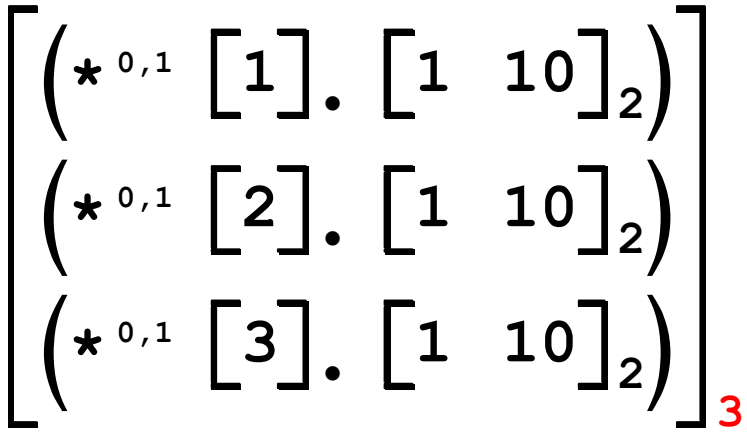
Axis alignment



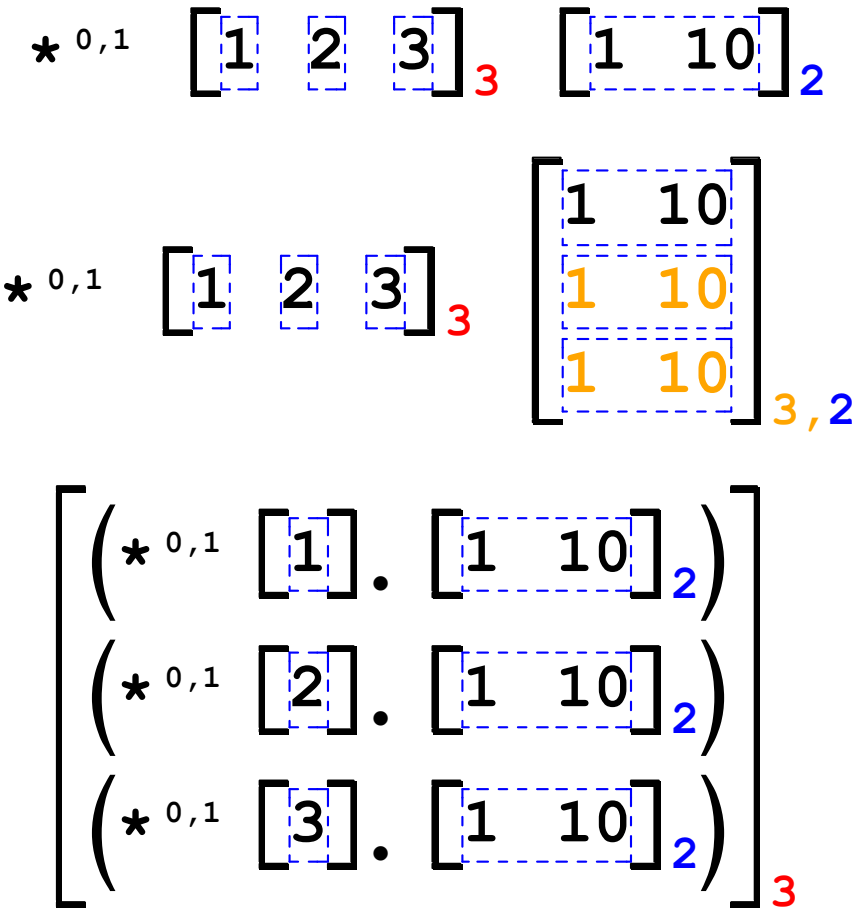
Axis alignment



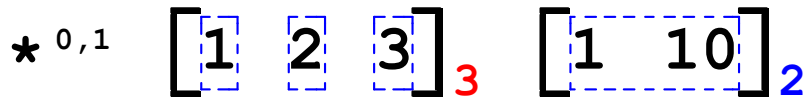


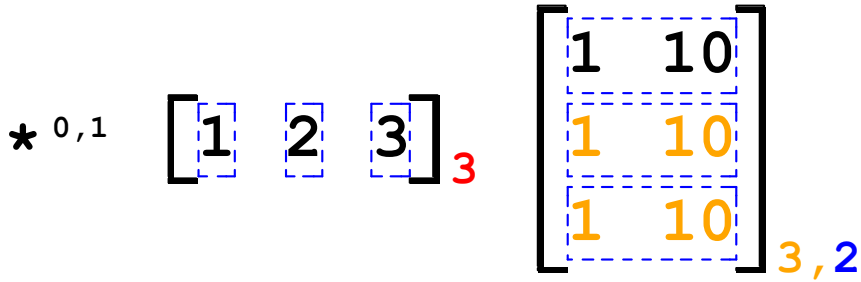


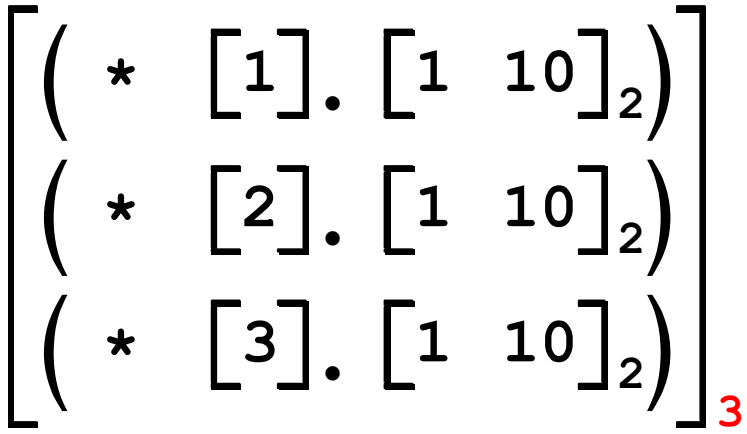
Axis alignment



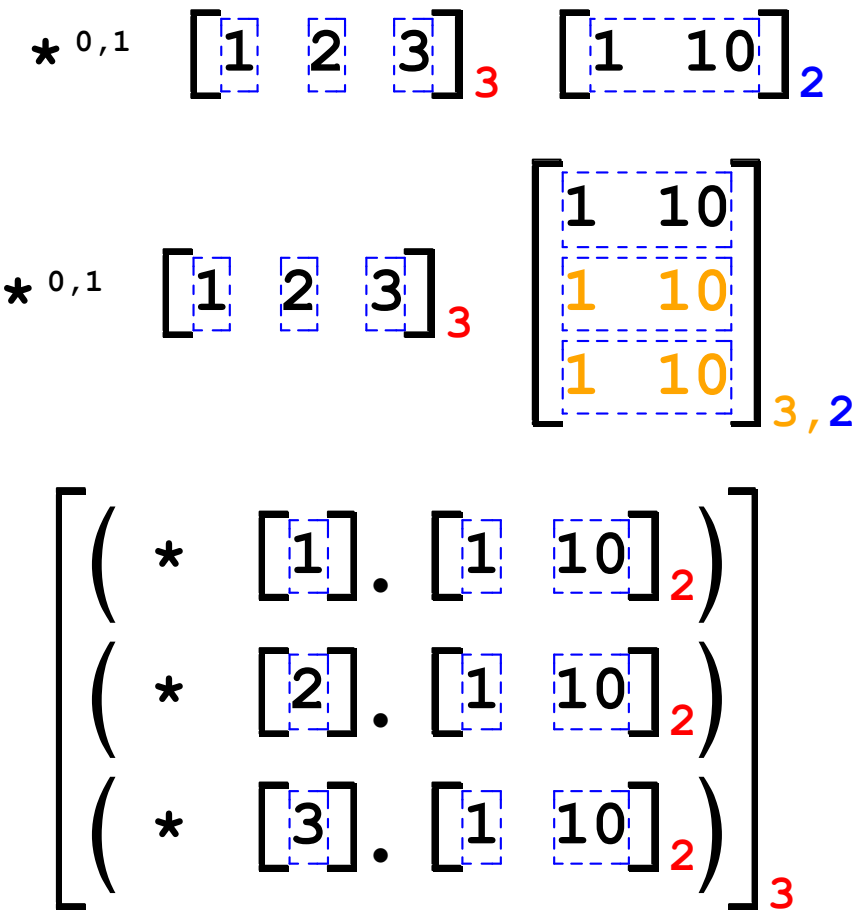
Axis alignment



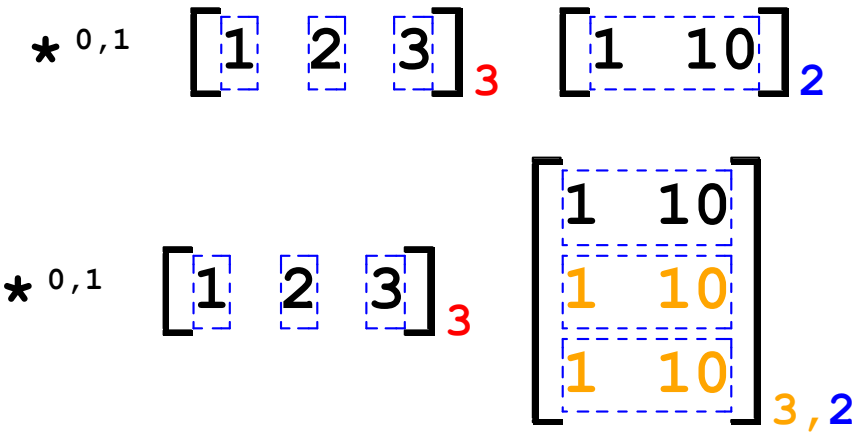




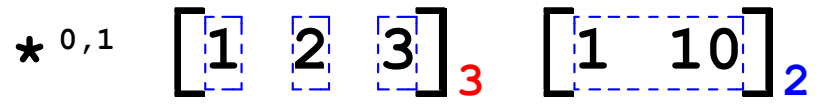
Axis alignment

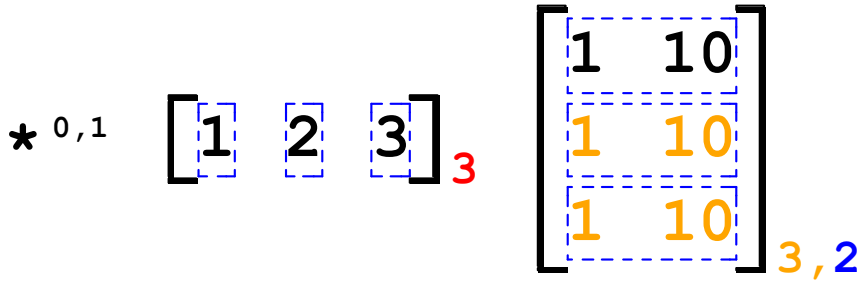


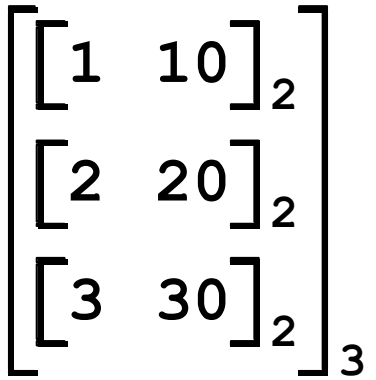
Axis alignment



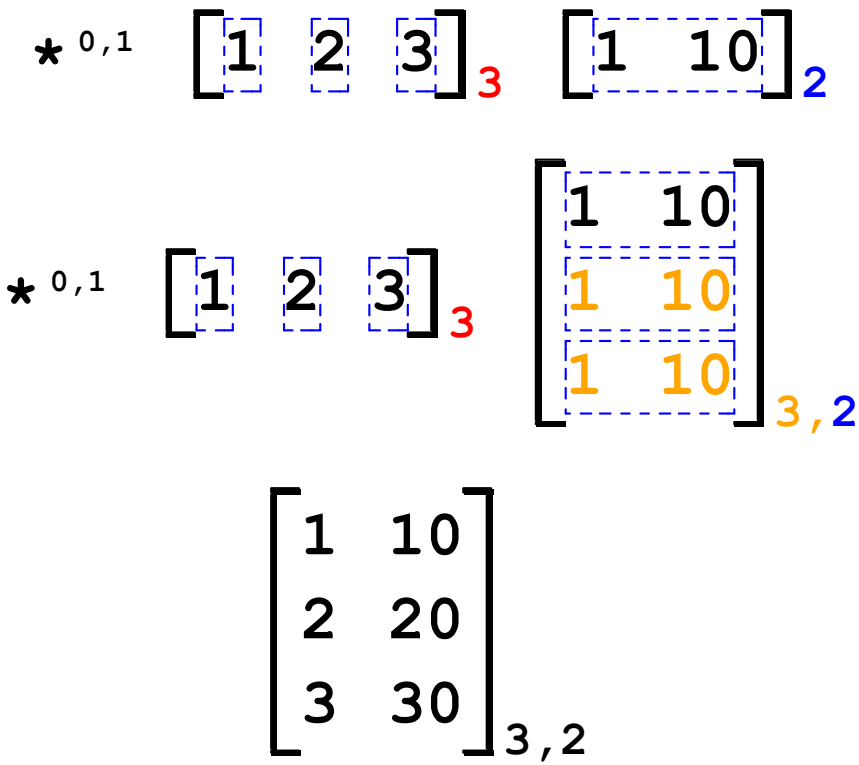
Axis alignment

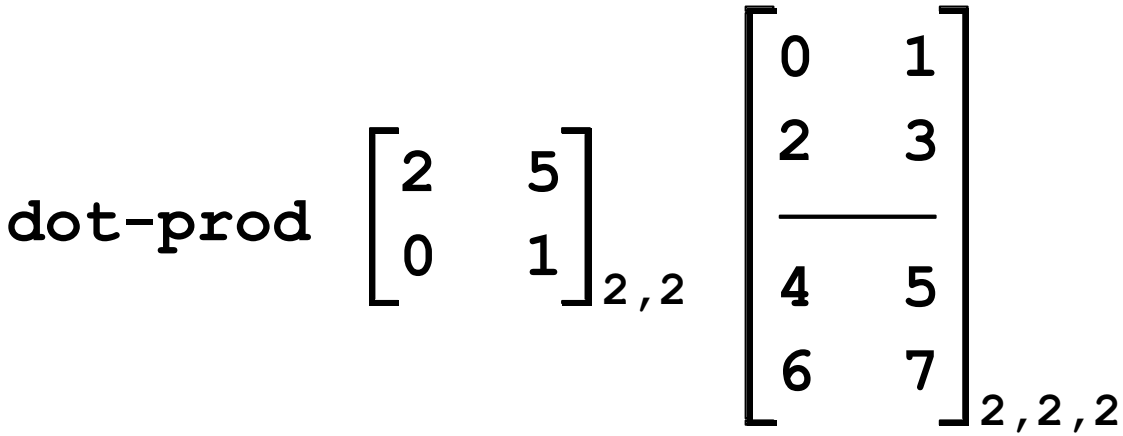


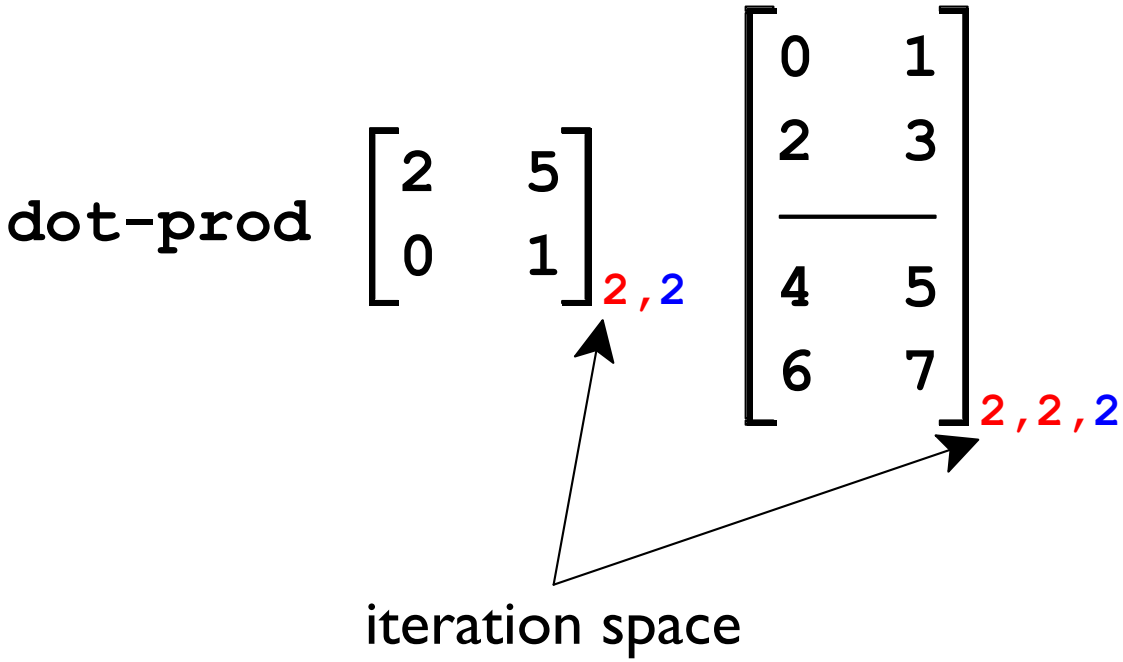




Axis alignment




dot-prod
2 5
0 1
2,2
0 1
2 3
4 5
6 7
2,2,2


dot-prod
2 5
0 1
2,2
0 1
2 3
4 5
6 7
2,2,2
iteration space

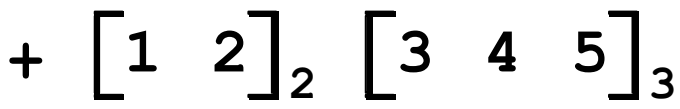
```
f       x       y
```

iteration space?

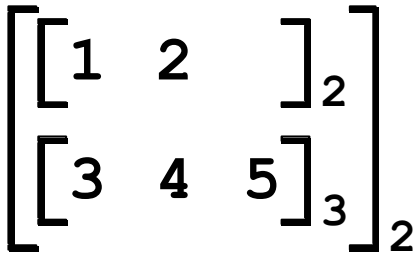
New ways for programs to go wrong

`+` $\left[\texttt{1}\ \ \texttt{2}\right]_{\texttt{2}}$ $\left[\texttt{3}\ \ \texttt{4}\ \ \texttt{5}\right]_{\texttt{3}}$ argument frames have mismatched shape

New ways for programs to go wrong



argument frames have mismatched shape



result cells have mismatched shape

New ways for programs to go wrong

```
+ [1  2]₂ [3  4  5]₃
```

argument frames have mismatched shape

```
⎡[1  2  ]₂⎤
⎣[3  4  5]₃⎦₂
```

result cells have mismatched shape

```
+ [ ]₃,₀,₄ [1].
```

frame is an empty array
indeterminate result cell shape

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\text{type indices} \quad \iota \ ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\text{type indices} \quad \iota \ ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\text{type indices} \quad \iota \ ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\text{type indices} \quad \iota \ ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\text{type indices} \quad \iota \ ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \ \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \texttt{A}_{(S\ 2\ 3)}\texttt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \texttt{NAT} \mid \texttt{SHAPE} \end{array}$$

Typing rank polymorphism

$$\left[\begin{matrix}\texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5}\end{matrix}\right]_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lrcl} \text{type indices} & \iota & ::= & n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= & \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lrcl} \text{type indices} & \iota & ::= & n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= & \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

$$\text{functions} \qquad f \ ::= \ \pi \mid (\lambda\ [(x\ \tau) \ldots]\ e)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \texttt{A}_{(S\ 2\ 3)}\texttt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \ \texttt{NAT} \mid \texttt{SHAPE} \end{array}$$

$$\text{functions} \qquad f \ ::= \ \pi \mid (\lambda\ [(x\ \tau) \ldots]\ e)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= \ n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \ \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

$$\text{functions} \qquad f \ ::= \ \pi \mid (\lambda\ [(x\ \tau) \ldots]\ e)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

$$\text{functions} \qquad f \ ::= \ \pi \mid (\lambda\ [(x\ \tau) \ldots]\ e)$$

Typing rank polymorphism

$$\begin{bmatrix} \texttt{2} & \texttt{4} & \texttt{6} \\ \texttt{1} & \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2,3}} \quad : \quad \mathtt{A}_{(S\ 2\ 3)}\mathtt{Num}$$

$$\begin{array}{lll} \text{type indices} & \iota & ::= n \mid x \mid (S\ \iota \ldots) \mid (+\ \iota\ \iota) \\ \text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE} \end{array}$$

$$\text{functions} \qquad f \ ::= \ \pi \mid (\lambda\ [(x\ \tau) \ldots]\ e)$$

Type equivalence

$$\begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}_{2,3} \quad : \quad \mathtt{A}_{(\mathrm{S}\ 2\ 3)}\mathtt{Num}$$

Type equivalence

$$\begin{bmatrix} 2 & 4 & 6 \\ 1 & 3 & 5 \end{bmatrix}_{2,3} \quad : \quad \mathtt{A}_{(\mathtt{s}\ 2\ 3)}\mathtt{Num}$$

$\mathtt{A}_{(\mathtt{s}\ 2\ 3)}\mathtt{A}_{(\mathtt{s})}\mathtt{Num}$ $\mathtt{A}_{(\mathtt{s}\ 2)}\mathtt{A}_{(\mathtt{s}\ 3)}\mathtt{Num}$ $\mathtt{A}_{(\mathtt{s})}\mathtt{A}_{(\mathtt{s}\ 2\ 3)}\mathtt{Num}$

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

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

$$+ : \mathrm{A}_{(\mathrm{S})}\mathrm{Num} \quad \mathrm{A}_{(\mathrm{S})}\mathrm{Num} \rightarrow \mathrm{A}_{(\mathrm{S})}\mathrm{Num}$$

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

$$\texttt{+} \ : \ \mathrm{A}_{(\mathrm{S})}\mathrm{Num} \quad \mathrm{A}_{(\mathrm{S})}\mathrm{Num} \to \mathrm{A}_{(\mathrm{S})}\mathrm{Num}$$

$$\begin{bmatrix} \texttt{1} & \texttt{4} \\ \texttt{2} & \texttt{3} \end{bmatrix}_{\texttt{2,2}} : \mathrm{A}_{\iota_1}\mathrm{A}_{(\mathrm{S})}\mathrm{Num} \qquad \begin{bmatrix} \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2}} : \mathrm{A}_{\iota_2}\mathrm{A}_{(\mathrm{S})}\mathrm{Num}$$

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

$$\texttt{+} \ : \ \mathrm{A}_{(\mathrm{S})}\mathrm{Num} \quad \mathrm{A}_{(\mathrm{S})}\mathrm{Num} \to \mathrm{A}_{(\mathrm{S})}\mathrm{Num}$$

$$\begin{bmatrix} \texttt{1} & \texttt{4} \\ \texttt{2} & \texttt{3} \end{bmatrix}_{\texttt{2,2}} : \mathrm{A}_{\iota_1}\mathrm{A}_{(\mathrm{S})}\mathrm{Num} \qquad \begin{bmatrix} \texttt{3} & \texttt{5} \end{bmatrix}_{\texttt{2}} : \mathrm{A}_{\iota_2}\mathrm{A}_{(\mathrm{S})}\mathrm{Num}$$

$$\iota_1 = (\mathrm{S}\ 2\ 2) \qquad \iota_2 = (\mathrm{S}\ 2)$$

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

$$\texttt{+} \ : \ \mathtt{A}_{(\mathtt{S})}\mathtt{Num}\quad \mathtt{A}_{(\mathtt{S})}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S})}\mathtt{Num}$$

$$\begin{bmatrix}\texttt{1} & \texttt{4}\\ \texttt{2} & \texttt{3}\end{bmatrix}_{\texttt{2,2}} : \mathtt{A}_{\iota_1}\mathtt{A}_{(\mathtt{S})}\mathtt{Num} \qquad \begin{bmatrix}\texttt{3} & \texttt{5}\end{bmatrix}_{\texttt{2}} : \mathtt{A}_{\iota_2}\mathtt{A}_{(\mathtt{S})}\mathtt{Num}$$

$$\iota_1 = (\mathtt{S}\ 2\ 2) \qquad \iota_2 = (\mathtt{S}\ 2)$$

$$\iota' = Max\ [\![\iota_1, \iota_2]\!] = (\mathtt{S}\ 2\ 2)$$ (n.b. prefix max)

$$+ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \quad \begin{bmatrix} 3 & 5 \end{bmatrix}_2$$

$$+ \; : \; \mathtt{A}_{(\mathtt{S})}\mathtt{Num} \quad \mathtt{A}_{(\mathtt{S})}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S})}\mathtt{Num}$$

$$\begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} : \mathtt{A}_{\iota_1}\mathtt{A}_{(\mathtt{S})}\mathtt{Num} \qquad \begin{bmatrix} 3 & 5 \end{bmatrix}_2 : \mathtt{A}_{\iota_2}\mathtt{A}_{(\mathtt{S})}\mathtt{Num}$$

$$\iota_1 = (\mathtt{S}\ 2\ 2) \qquad \iota_2 = (\mathtt{S}\ 2)$$

$$\iota' = Max\ [\![\iota_1, \iota_2]\!] = (\mathtt{S}\ 2\ 2) \qquad \text{(n.b. prefix max)}$$

$$+ \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \end{bmatrix}_2$$

$$+ \; : \; \mathtt{A}_{(\mathtt{S})}\mathtt{Num} \quad \mathtt{A}_{(\mathtt{S})}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S})}\mathtt{Num}$$

$$\begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix}_{2,2} : \mathtt{A}_{\iota_1}\mathtt{A}_{(\mathtt{S})}\mathtt{Num} \qquad \begin{bmatrix} 3 & 5 \end{bmatrix}_2 : \mathtt{A}_{\iota_2}\mathtt{A}_{(\mathtt{S})}\mathtt{Num}$$

$$\iota_1 = (\mathtt{S}\ 2\ 2) \qquad \iota_2 = (\mathtt{S}\ 2)$$

$$\iota' = Max\ [\![\iota_1, \iota_2]\!] = (\mathtt{S}\ 2\ 2) \qquad \text{(n.b. prefix max)}$$

$$\mathtt{A}_{(\mathtt{S}\ 2\ 2)}\mathtt{A}_{(\mathtt{S})}\mathtt{Num} \;\cong\; \mathtt{A}_{(\mathtt{S}\ 2\ 2)}\mathtt{Num}$$

Type Soundness

"Well-typed programs don't have shape errors."

Type Soundness

"Well-typed programs don't have shape errors."

+ $\left[\texttt{1}\ \ \texttt{2}\right]_2$ $\left[\texttt{3}\ \ \texttt{4}\ \ \texttt{5}\right]_3$ $Max[\![\iota \ldots]\!]$ requires principal frame

Type Soundness

"Well-typed programs don't have shape errors."

$+ \; [1 \;\; 2]_2 \; [3 \;\; 4 \;\; 5]_3$ $\quad Max[\![\iota \ldots]\!]$ requires principal frame

$$\left[[1 \;\; 2 \;\;]_2 \; [3 \;\; 4 \;\; 5]_3\right]_2$$

function term has single output type

Type Soundness

"Well-typed programs don't have shape errors."

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

$Max[\![\iota \ldots]\!]$ requires principal frame

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

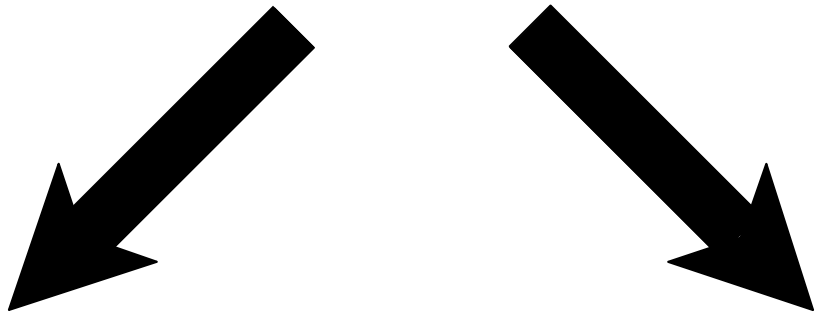
function term has single output type

```
+ [ ]3,0,4 [1].
```

result cell shape determined
by function type

Extract array programming model

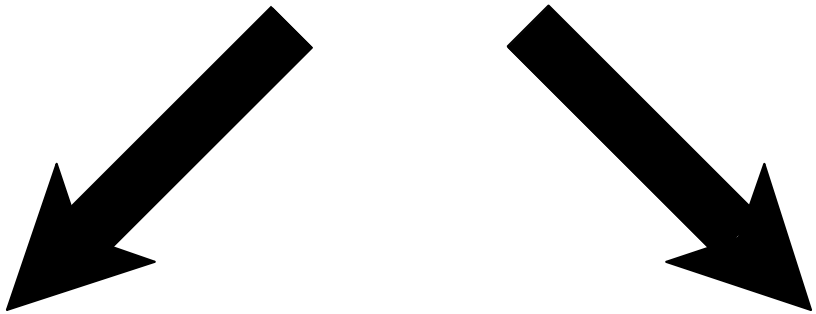
Extract array programming model



Dynamic semantics:
formalization of
rank polymorphism

Static semantics:
iteration space
known statically

Extract array programming model



Dynamic semantics:
formalization of
rank polymorphism

Static semantics:
iteration space
known statically

Not shown:

Arrays in function position → MIMD

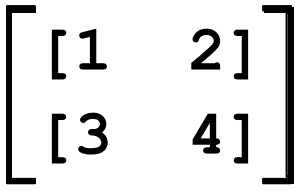
Partially irregular arrays using 'boxes'

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)

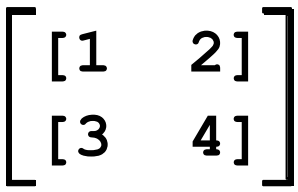
Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)



Related work

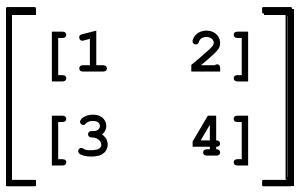
Jay, C.B.: The FISh language definition. Tech. rep. (1998)



: `vec[vec[int]]`

Related work

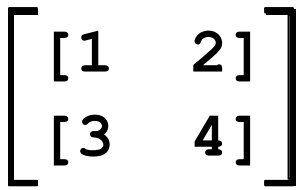
Jay, C.B.: The FISh language definition. Tech. rep. (1998)



```
[1  2]    :  vec[vec[int]]
[3  4]    #  2 × 2 × int-shape
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)

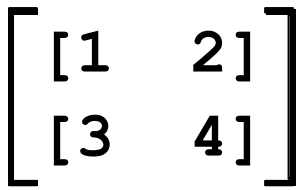


```
[1  2]     :  vec[vec[int]]
[3  4]     #  2 × 2 × int-shape
```

```
dot-prod
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)

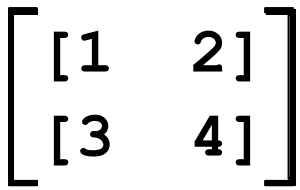


```
[1  2]      :  vec[vec[int]]
[3  4]      #  2 × 2 × int-shape

            :  vec[int] -> int
dot-prod
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)



```
[1  2]
[3  4]
```

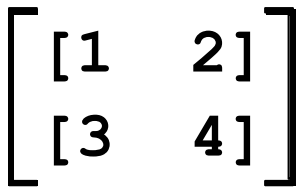
```
:  vec[vec[int]]
#  2 × 2 × int-shape
```

```
dot-prod
```

```
:  vec[int] -> int
#  s × int-shape -> int-shape
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)



```
[1  2]
[3  4]
```

```
:  vec[vec[int]]
#  2 × 2 × int-shape
```

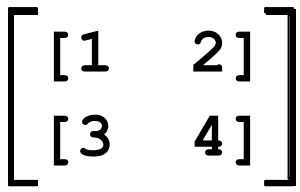
```
dot-prod
```

```
:  vec[int] -> int
#  s × int-shape -> int-shape
```

```
iota
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)



```
[1  2]
[3  4]
```

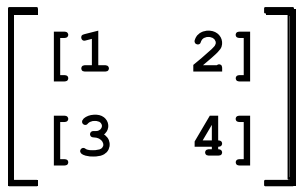
```
:  vec[vec[int]]
#  2 × 2 × int-shape
```

```
dot-prod
:  vec[int] -> int
#  s × int-shape -> int-shape
```

```
iota
:  int -> vec[int]
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)



```
[1  2]
[3  4]
```

```
:  vec[vec[int]]
#  2 × 2 × int-shape
```

```
dot-prod
```

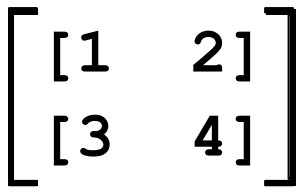
```
:  vec[int] -> int
#  s × int-shape -> int-shape
```

```
iota
```

```
:  int -> vec[int]
not shapeable
```

Related work

Jay, C.B.: The FISh language definition. Tech. rep. (1998)



```
[1  2]
[3  4]
```

```
:  vec[vec[int]]
#  2 × 2 × int-shape
```

```
dot-prod
```

```
:  vec[int] -> int
#  s × int-shape -> int-shape
```

```
iota
```

```
:  int -> vec[int]
not shapeable
```

Regularity enforced too thoroughly

Related work

Data Parallel Haskell: A Status Report. Chakravarty, et al.

Related work

Data Parallel Haskell: A Status Report. Chakravarty, et al.

Different programming model: irregular nested vectors

Related work

Regular, Shape-polymorphic, Parallel Arrays in Haskell. Keller, et al.

Related work

Regular, Shape-polymorphic, Parallel Arrays in Haskell. Keller, et al.

Trouble lifting already-aggregate operations to higher rank

Related work

Regular, Shape-polymorphic, Parallel Arrays in Haskell. Keller, et al.

Trouble lifting already-aggregate operations to higher rank

Possible back-end

Future goal:

Parallelization like a Fortran compiler

without analysis like a Fortran compiler

Future goal:

Parallelization like a Fortran compiler

without analysis like a Fortran compiler

Thanks