Array computation and statically typing shape-polymorphic languages

Justin Slepak

Northeastern University

- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

Iverson: APL

```
for(i=0; i<2; i++)
  for(j=0; j<2; i++)
    B[i][j] = A[i][j] * x;
```

Iverson: APL

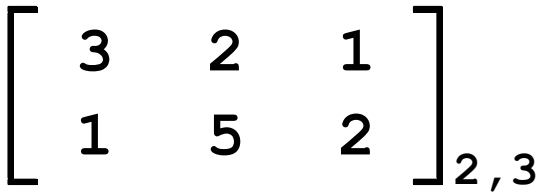
```
for(i=0; i<2; i++)
  for(j=0; j<2; i++)
    B[i][j] = A[i][j] * x;
```

```
let B = map (map (*x)) A
```

Iverson: APL

```
⎡ 3  2  1 ⎤
⎣ 1  5  2 ⎦ * x
```

Data in APL/J: Regular Arrays

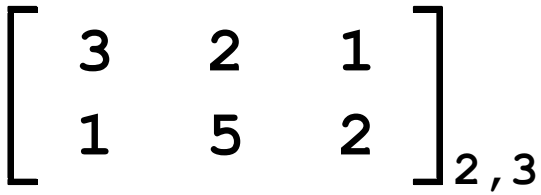


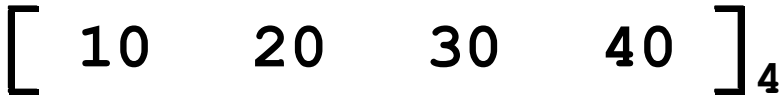
Data in APL/J: Regular Arrays

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

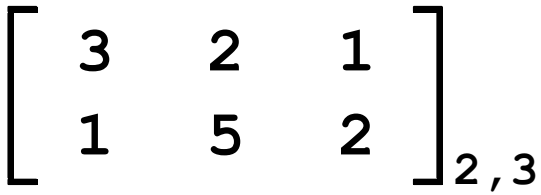
$$\begin{bmatrix} 10 & 20 & 30 & 40 \end{bmatrix}_{4}$$

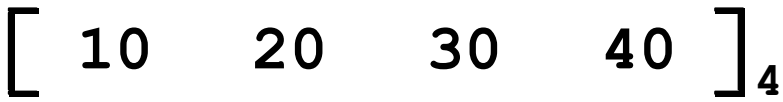
Data in APL/J: Regular Arrays

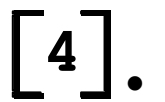




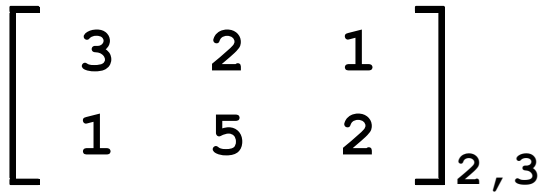
Data in APL/J: Regular Arrays







Data in APL/J: Regular Arrays



atoms: `3, 2, 1, 1, 5, 2`

shape: `2, 3`

Functions in APL/J

```
  * 7
```

signum

Functions in APL/J

```
* 7
```
signum

```
3 * 7
```
multiplication

Functions in APL/J

```
* 7
```
signum

```
3 * 7
```
multiplication

```
* 5 _1 0
```
➡
```
1 _1 0
```

Functions in APL/J

```
* 7
```
signum

```
3 * 7
```
multiplication

```
* 5 _1 0   ➡   1 _1 0
```

```
3 * 5 _1 0   ➡   15 _3 0
```

- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

History of lifting: scalar agreement

```
[7  9  11]3 + [1].
```

one argument scalar

```
[1]. + [7  9  11]3
```

```
[7  9  11]3 + [1  2  3]3
```

same shape

History of lifting: scalar agreement

Not allowed:

$$\begin{bmatrix}7 & 9 & 11\end{bmatrix}_3 + \begin{bmatrix}1 & 2\end{bmatrix}_2$$ shape mismatch

$$\begin{bmatrix}7 & 9\\ 11 & 13\end{bmatrix}_{2,2} + \begin{bmatrix}1 & 2\end{bmatrix}_2$$ lifting rule too limited

History of lifting: argument rank

vector dot product	`1,1`
	.
	.
	.
	.

History of lifting: argument rank

vector dot product	`1,1`
polynomial evaluation	`1,0`

History of lifting: argument rank

vector dot product	`1,1`
polynomial evaluation	`1,0`
matrix inversion	`2`

History of lifting: argument rank

Cells: individual sub-arrays of operator's expected rank

History of lifting: argument rank

Cells: individual sub-arrays of operator's expected rank

Frame: array structure around the cells

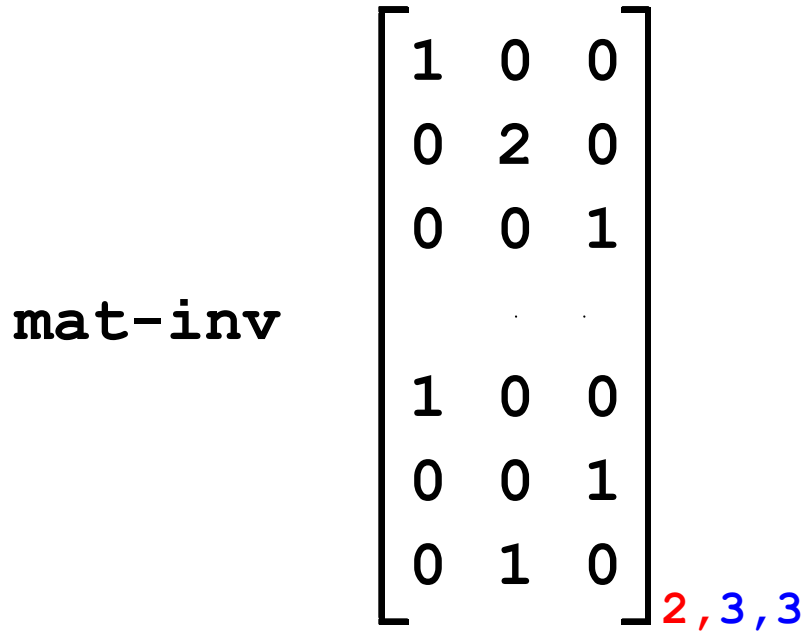
History of lifting: argument rank

Cells: individual sub-arrays of operator's expected rank

Frame: array structure around the cells

n+1 different "frame of cells" views of rank-n array

History of lifting: argument rank



History of lifting: argument rank

mat-inv

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

2 cells:

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

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

History of lifting: argument rank

mat-inv

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

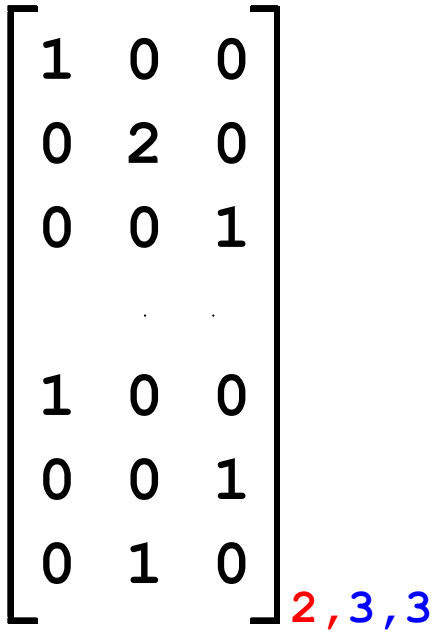
2 result cells:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}_{3,3}$$

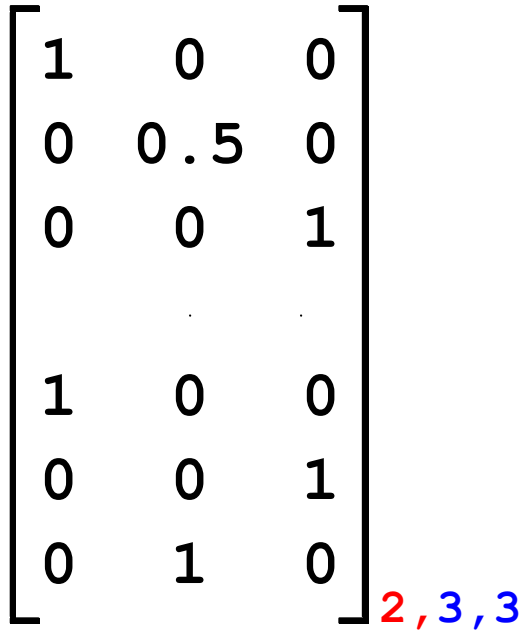
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}_{3,3}$$

History of lifting: argument rank

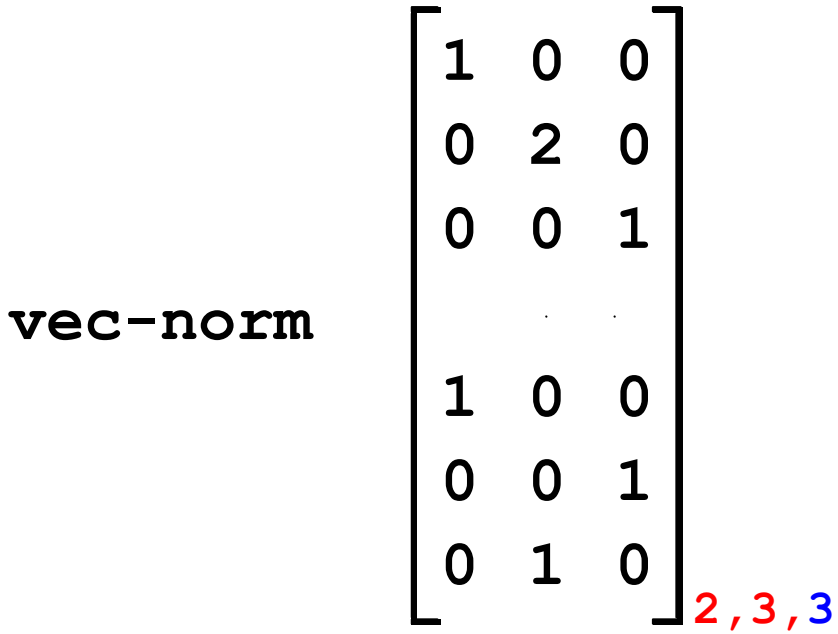
mat-inv



final result:



History of lifting: argument rank



History of lifting: argument rank

`vec-norm` $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \\ & \cdot & \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}_{2,3,3}$

$2 \times 3 = 6$ cells: $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}_3$

$\begin{bmatrix} 0 & 2 & 0 \end{bmatrix}_3$

$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}_3$

$\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}_3$

$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}_3$

$\begin{bmatrix} 0 & 1 & 0 \end{bmatrix}_3$

History of lifting: argument rank

2 × 3 = 6 result cells: [1].

`vec-norm`

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

[2].

[1].

[1].

[1].

[1].

History of lifting: argument rank

vec-norm

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

final result:

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

History of lifting: argument rank

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \quad \bullet \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

History of lifting: argument rank

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \quad \bullet \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}_{2,2} \quad \bullet \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

History of lifting: argument rank

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \bullet \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}_{2,2} \bullet \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \bullet \begin{bmatrix} 7 & 9 \end{bmatrix}_{2}$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \bullet \begin{bmatrix} 11 & 13 \end{bmatrix}_{2}$$

History of lifting: argument rank

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \quad + \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

History of lifting: argument rank

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \quad + \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

still not allowed

History of lifting: prefix agreement

one frame must be prefix of the other frame

History of lifting: prefix agreement

one frame must be prefix of the other frame

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \quad \bullet \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

expects rank 1

$\bullet \sqsubseteq 2$

History of lifting: prefix agreement

one frame must be prefix of the other frame

$$\begin{bmatrix} 1 & 2 \end{bmatrix}_{2} \quad \bullet \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

expects rank 1

- $\bullet \sqsubseteq 2$
- $\bullet$ ++ 2 = 2

History of lifting: prefix agreement

one frame must be prefix of the other frame

$$\left[1\ \ 2\right]_{2} \quad \bullet \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

expects rank 1

$\bullet \sqsubseteq 2$

$\bullet$ ++ 2 = 2

$$\left[1\ \ 2\right]_{2} \quad + \quad \begin{bmatrix} 7 & 9 \\ 11 & 13 \end{bmatrix}_{2,2}$$

expects rank 0

$2 \sqsubseteq 2,2$

History of lifting: prefix agreement

one frame must be prefix of the other frame

$$\begin{bmatrix}1 & 2\end{bmatrix}_{2} \quad \bullet \quad \begin{bmatrix}7 & 9\\ 11 & 13\end{bmatrix}_{2,2}$$

expects rank 1

$\bullet \sqsubseteq 2$

$\bullet \mathbin{++} 2 = 2$

$$\begin{bmatrix}1 & 2\end{bmatrix}_{2} \quad + \quad \begin{bmatrix}7 & 9\\ 11 & 13\end{bmatrix}_{2,2}$$

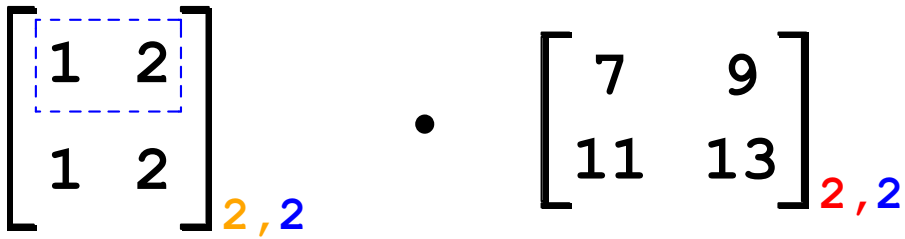
expects rank 0

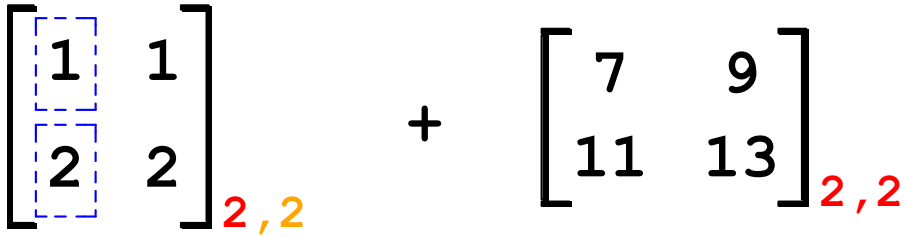
$2 \sqsubseteq 2,2$

$2 \mathbin{++} 2 = 2,2$

History of lifting: prefix agreement

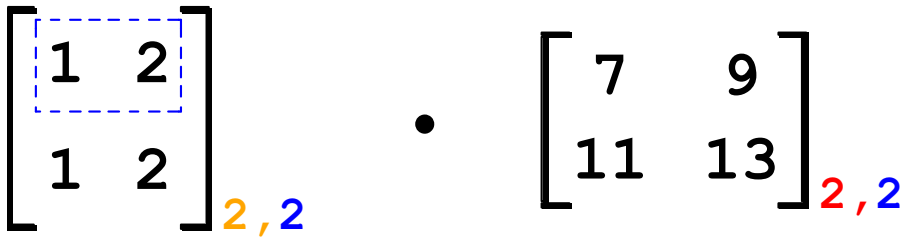
expand into new frame by replicating each cell

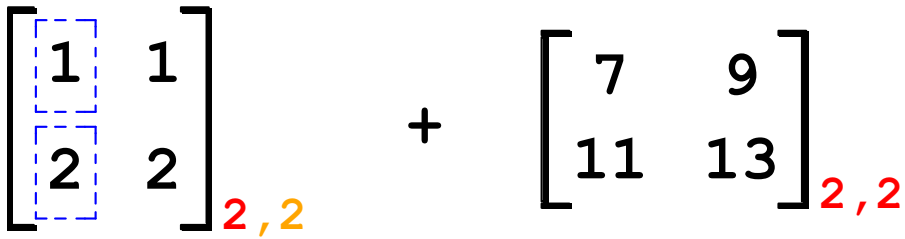




History of lifting: prefix agreement

expand into new frame by replicating each cell





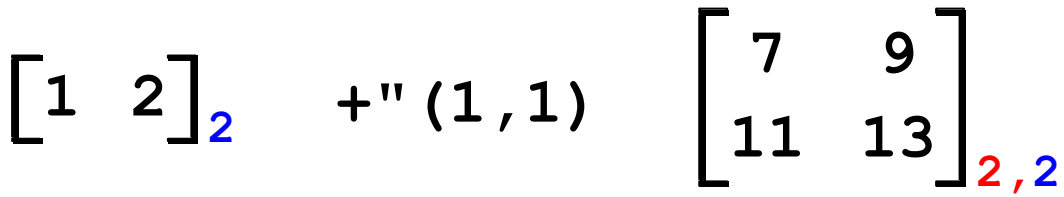
argument rank determines what axes to expand along

History of lifting: prefix agreement

Reranking: treat function as having a different rank

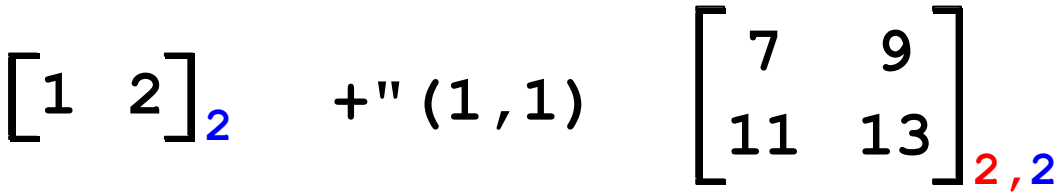
History of lifting: prefix agreement

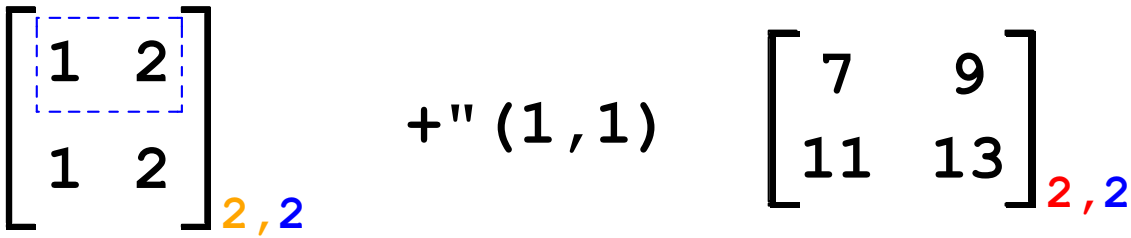
Reranking: treat function as having a different rank



History of lifting: prefix agreement

Reranking: treat function as having a different rank





History of lifting: prefix agreement

$$\begin{bmatrix} 100 & 200 \\ 300 & 400 \end{bmatrix}_{2,2} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ \\ 5 & 6 \\ 7 & 8 \end{bmatrix}_{2,2,2}$$

History of lifting: prefix agreement

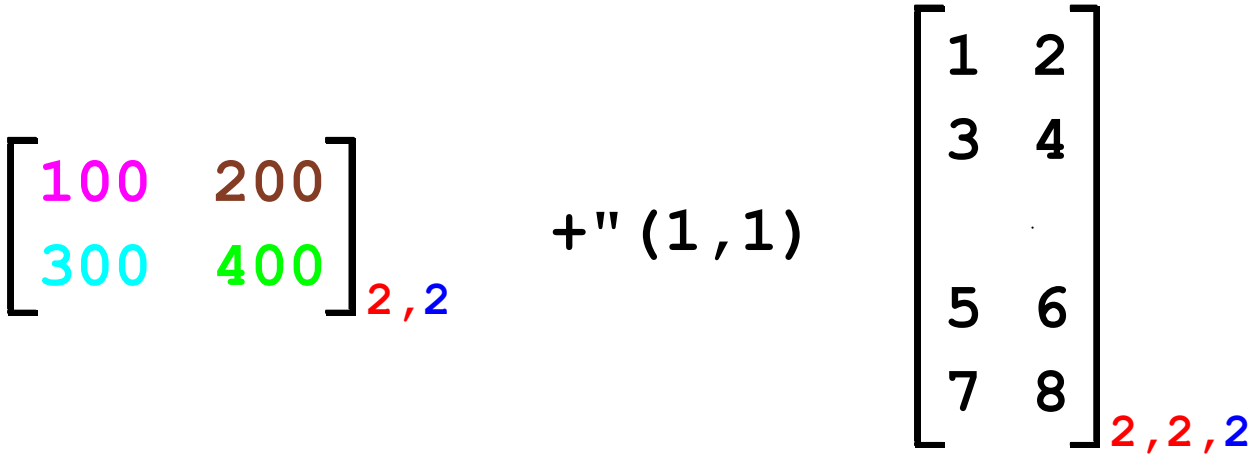
$$\begin{bmatrix} 100 & 100 \\ 200 & 200 \\ & \\ 300 & 300 \\ 400 & 400 \end{bmatrix}_{2,2,2} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ & \\ 5 & 6 \\ 7 & 8 \end{bmatrix}_{2,2,2}$$

History of lifting: prefix agreement

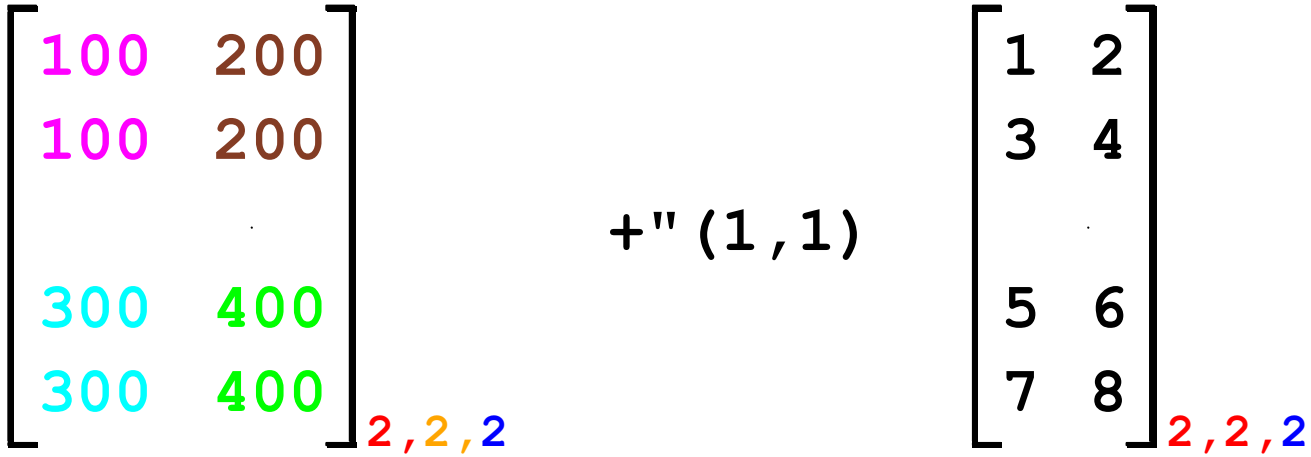
$$\begin{bmatrix} 100 & 100 \\ 200 & 200 \\ \\ 300 & 300 \\ 400 & 400 \end{bmatrix}_{2,2,2} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ \\ 5 & 6 \\ 7 & 8 \end{bmatrix}_{2,2,2}$$

$$\begin{bmatrix} 101 & 102 \\ 203 & 204 \\ \\ 305 & 306 \\ 407 & 408 \end{bmatrix}_{2,2,2}$$

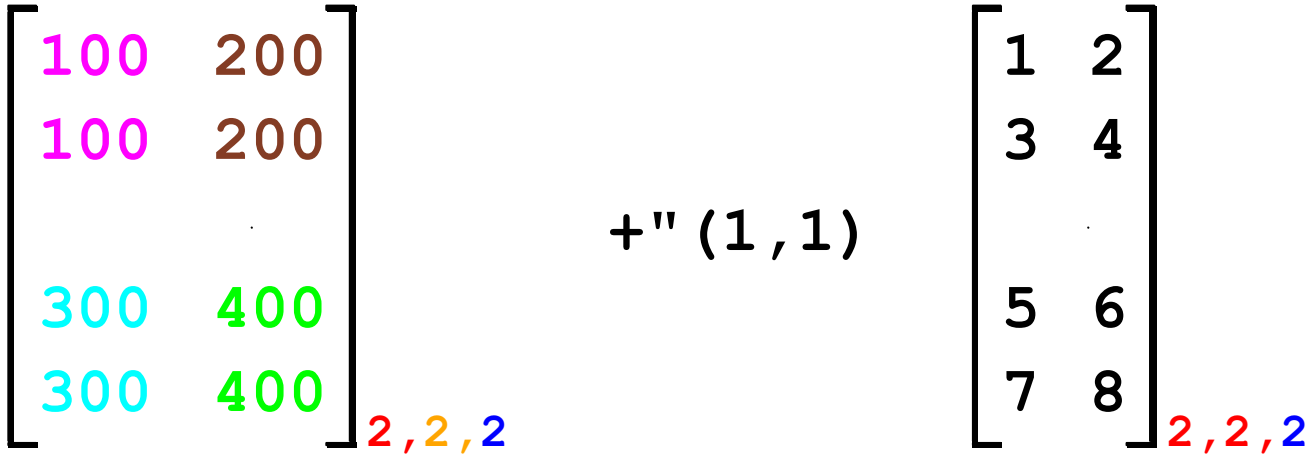
History of lifting: prefix agreement

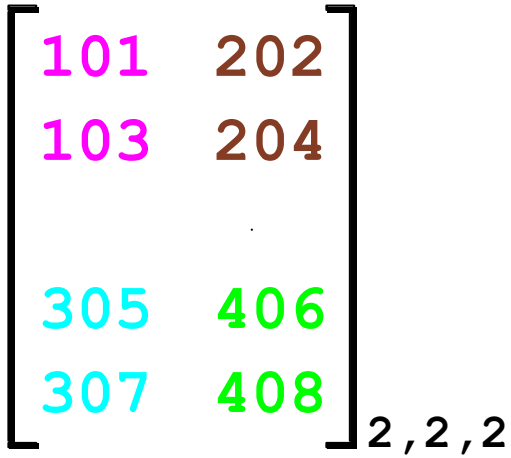


History of lifting: prefix agreement

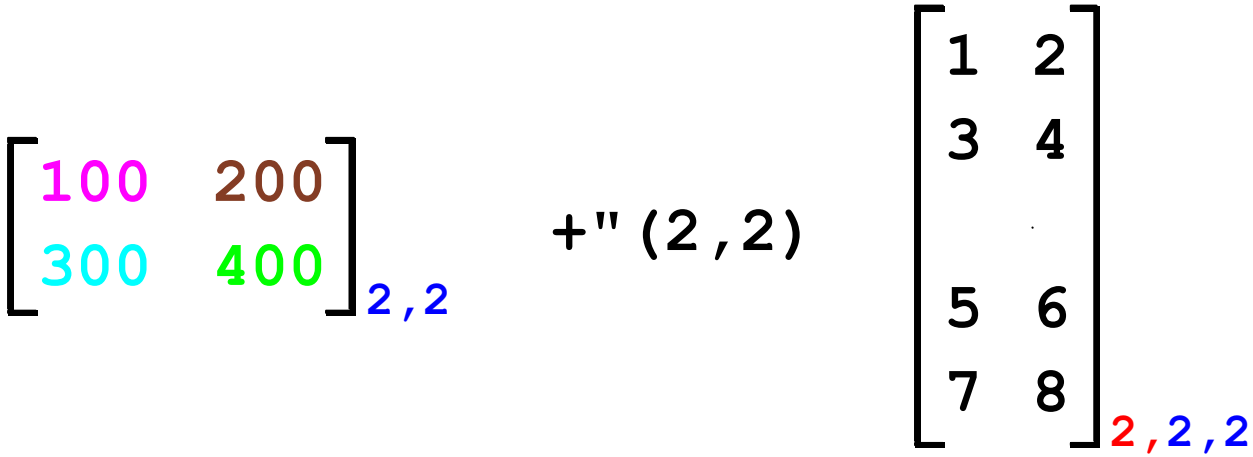


History of lifting: prefix agreement

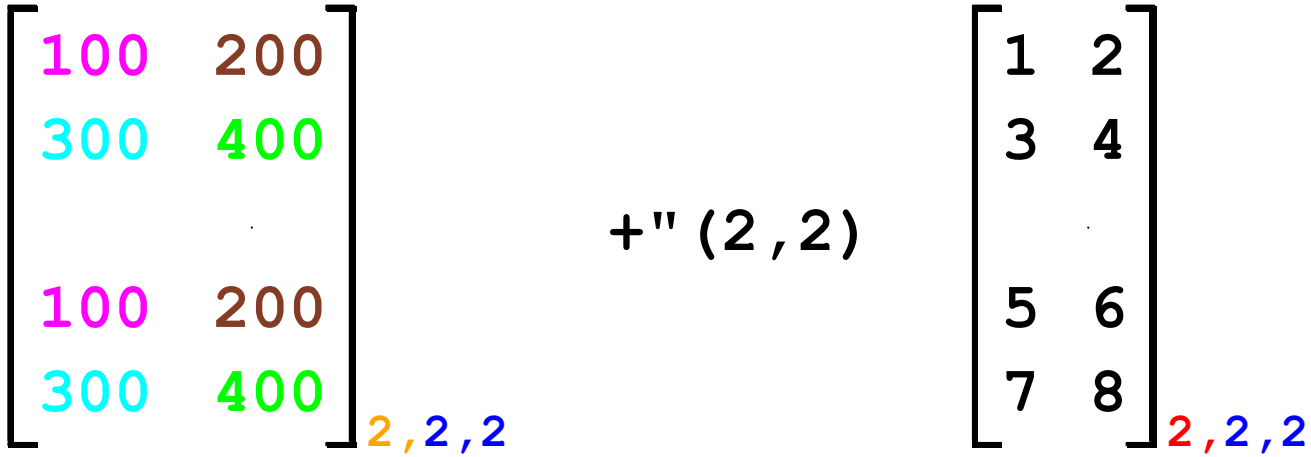




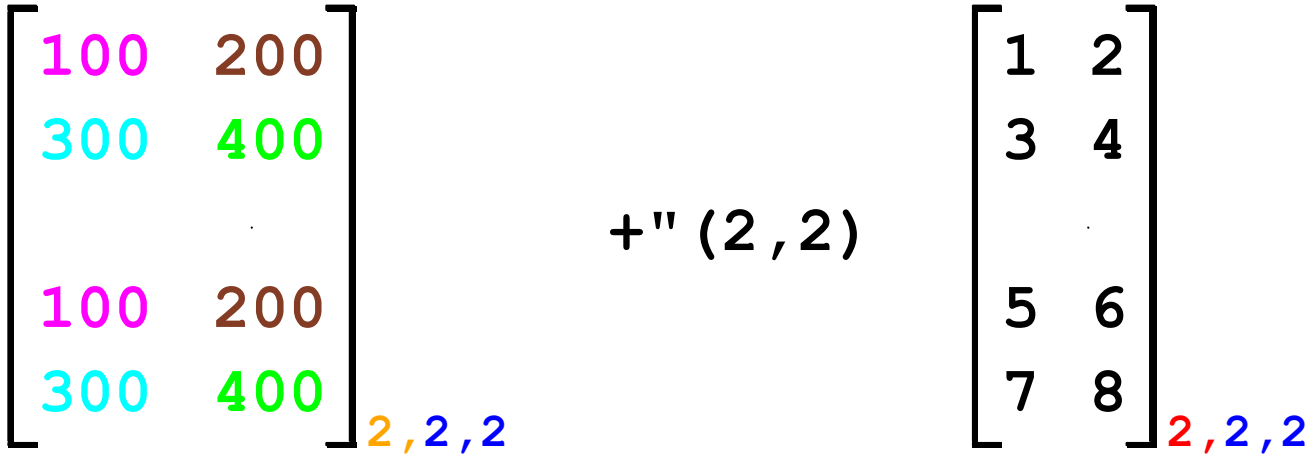
History of lifting: prefix agreement

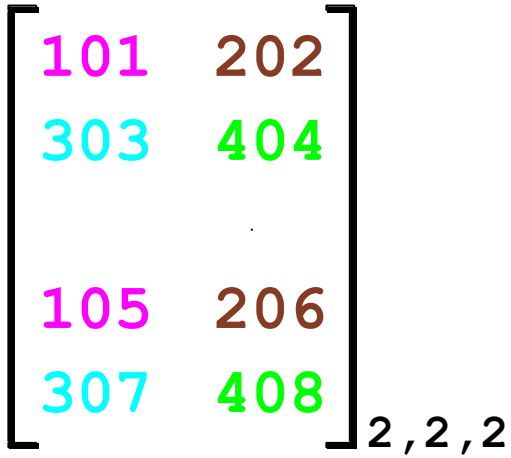


History of lifting: prefix agreement



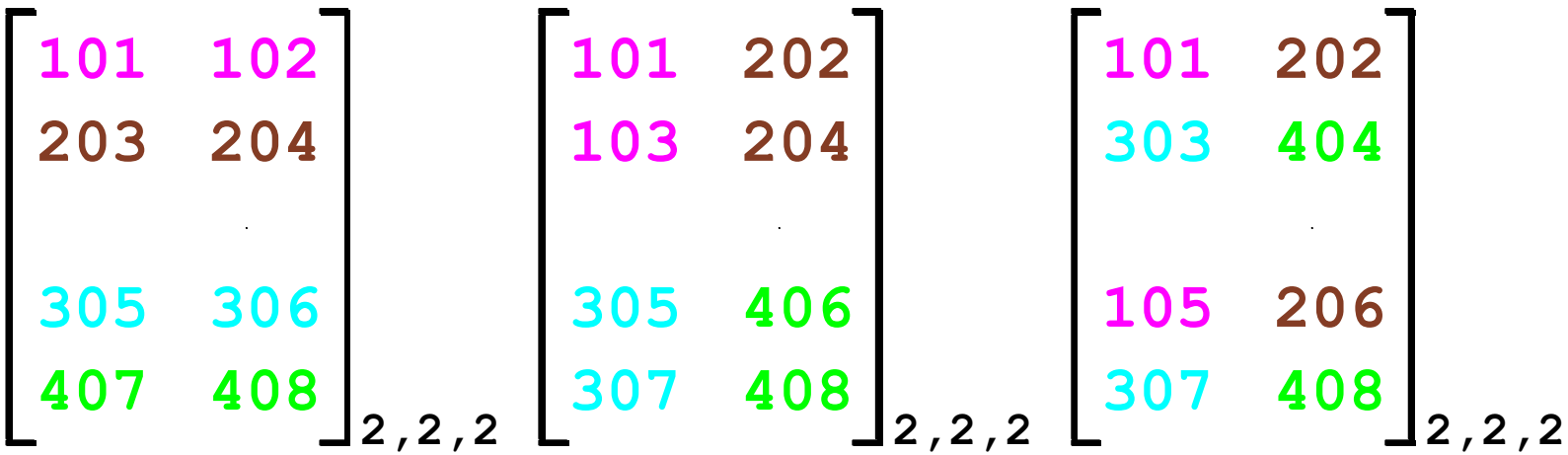
History of lifting: prefix agreement





History of lifting: prefix agreement

different expected rank leads to different lifting behavior



- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

Generalizing: higher arity

```
lerp [0  0]2 [7  1]2 [4].
```

Generalizing: higher arity

```
lerp [0  0]₂ [7  1]₂ [4].
```

p_0 p_1 x

Generalizing: higher arity

```
lerp [0  0]₂ [7  1]₂ [4].
```

p_0	p_1	x
1	1	0

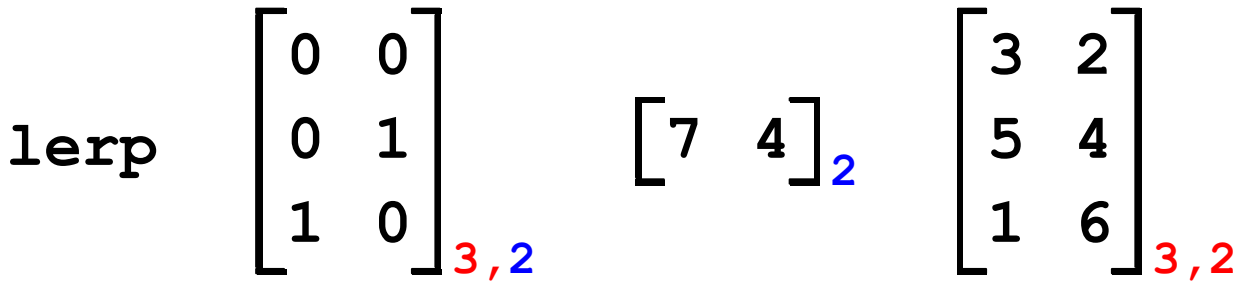
Generalizing: higher arity

```
lerp [0  0]₂ [7  1]₂ [4].
```

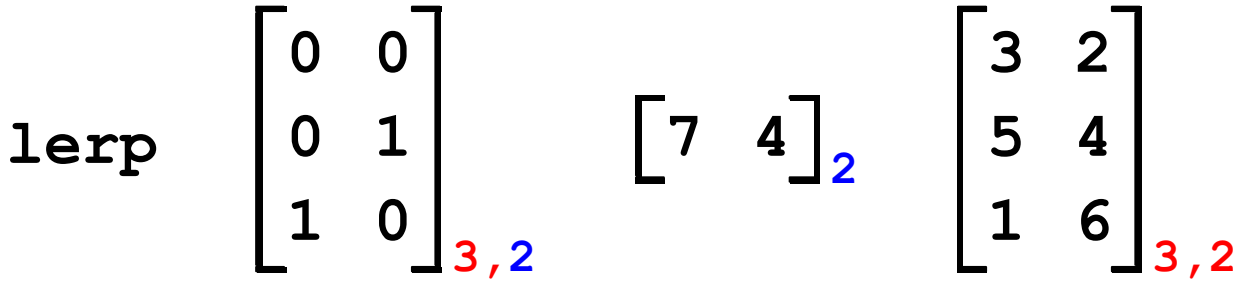
p_0	p_1	x
`1`	`1`	`0`

goal: lift on any parameter

Generalizing: higher arity

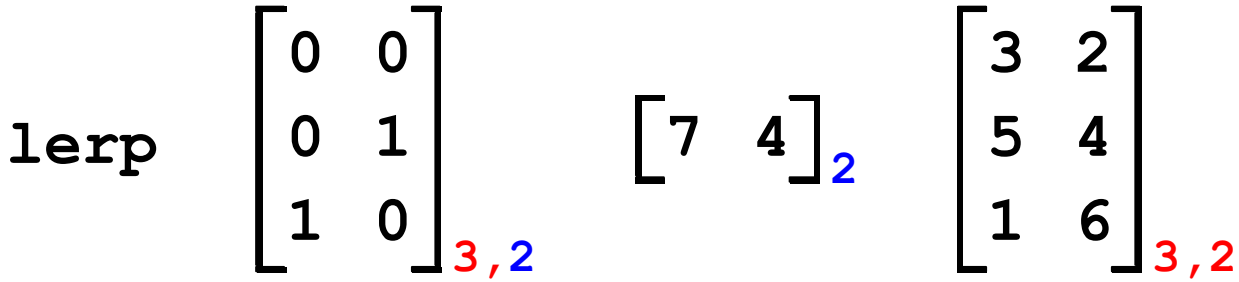


Generalizing: higher arity



all frames must be prefix of one "principal frame"

Generalizing: higher arity



all frames must be prefix of one "principal frame"

- ⊑ 3 ⊑ 3,2

Generalizing: higher arity

Generalizing: higher order

```
repeat [3]. [inc  dec]2
```

Generalizing: higher order

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

Generalizing: higher order

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

$$(\ \texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2\ )\ [10].$$

Generalizing: higher order

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

$$[\texttt{(+3)}\ \ \texttt{(-3)}]_2\ \ [10].$$

Generalizing: higher order

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

$$[\texttt{(+3)}\ \ \texttt{(-3)}]_2\ \ [10].$$

array in function position

Generalizing: higher order

$$\texttt{repeat}\ [3].\ [\texttt{inc}\ \ \texttt{dec}]_2 \mapsto [\texttt{(+3)}\ \ \texttt{(-3)}]_2$$

$$\texttt{apply}\ \ [\texttt{(+3)}\ \ \texttt{(-3)}]_2\ \ [10].$$

array in function position

Generalizing: higher order

```
apply [(+3) (-3)]₂ [10].
```

what is **`apply`**'s expected rank?

```
0, ? ...
```

Generalizing: higher order

```
apply [(+3) (-3)]₂ [10].
```

what is `apply`'s expected rank?

```
0, ? ...
```

fill in with ranks expected by `(+3)` and `(-3)`

Generalizing: higher order

```
apply [(+3) (-3)]₂ [10].
```

what is `apply`'s expected rank?

```
0, ? ...
```

fill in with ranks expected by `(+3)` and `(-3)`

```
0, 0
```

Generalizing: higher order

$$\left[\texttt{(+3)}\quad\texttt{(-3)}\right]_2\ \left[\texttt{10}\right].$$

Generalizing: higher order

```
[(+3)  (-3)]2 [10].                    0, 0
```

Generalizing: higher order

$$[(+3) \quad (-3)]_2 \; [10 \quad 10]_2 \qquad\qquad 0, \; 0$$

Generalizing: higher order

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+ - *]3 [1  3  5]3 [2].
```

Generalizing: higher order

```
[(+3)  (-3)]₂ [10  10]₂                    0, 0

[+ - *]₃ [1  3  5]₃ [2].                   0, 0, 0
```

Generalizing: higher order

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2  2  2]3           0, 0, 0
```

Generalizing: higher order

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2  2  2]3           0, 0, 0

[vec-norm  vec-sum]2 [1  3  5]3
```

Generalizing: higher order

```
[(+3)  (-3)]2 [10  10]2                    0, 0

[+  -  *]3 [1  3  5]3 [2  2  2]3           0, 0, 0

[vec-norm  vec-sum]2 [1  3  5]3            0, 1
```

Generalizing: higher order

$$\left[(+3)\ \ (-3)\right]_2\ \left[10\ \ 10\right]_2 \qquad 0,\ 0$$

$$\left[+\ -\ *\right]_3\ \left[1\ \ 3\ \ 5\right]_3\ \left[2\ \ 2\ \ 2\right]_3 \qquad 0,\ 0,\ 0$$

$$\left[\text{vec-norm}\ \ \text{vec-sum}\right]_2\ \begin{bmatrix}1 & 3 & 5\\ 1 & 3 & 5\end{bmatrix}_{2,3} \qquad 0,\ 1$$

Generalizing: higher order

```
[(+3)  (-3)]₂ [10  10]₂                    0, 0

[+  -  *]₃ [1  3  5]₃ [2  2  2]₃           0, 0, 0

[vec-norm  vec-sum]₂ [1  3  5]
                     [1  3  5]₂,₃          0, 1

[vec-norm  mat-inv]₂ [1  3]
                     [2  5]₂,₂
```

Generalizing: higher order

$$[(+3)\ \ (-3)]_2\ [10\ \ 10]_2 \qquad 0,\ 0$$

$$[+\ -\ *]_3\ [1\ \ 3\ \ 5]_3\ [2\ \ 2\ \ 2]_3 \qquad 0,\ 0,\ 0$$

$$[\texttt{vec-norm}\ \ \texttt{vec-sum}]_2 \begin{bmatrix} 1 & 3 & 5 \\ 1 & 3 & 5 \end{bmatrix}_{2,3} \qquad 0,\ 1$$

$$[\texttt{vec-norm}\ \ \texttt{mat-inv}]_2 \begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix}_{2,2}$$

not allowed
(expected ranks
do not match)

- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

Abstract syntax

atom **vs** scalar

Abstract syntax

atom	vs	scalar
`6`		`[6].`

Abstract syntax

atom	vs	scalar
`6`		`[6].`
`+`		`[+].`

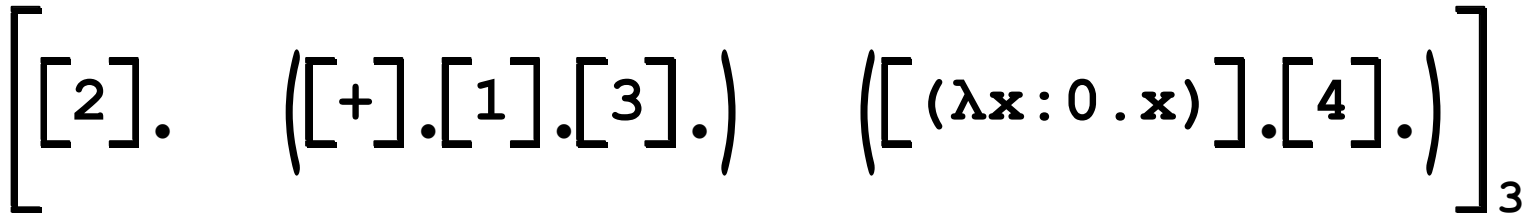
Abstract syntax

atom	vs	scalar
`6`		`[6].`
`+`		`[+].`
`(λx:0.x)`		`[(λx:0.x)].`

Abstract syntax

atom	vs	scalar
`6`		`[6].`
`+`		`[+].`
`(λx:0.x)`		`[(λx:0.x)].`

syntactically nested arrays



Abstract syntax

arrays $\qquad \alpha \;::=\; [l \ldots]_{n\ldots}$

Abstract syntax

$$\begin{array}{lll} \text{arrays} & \alpha & ::= \; [l \ldots]_{n\ldots} \\ \text{elements} & l & ::= \; e \mid b \mid f \end{array}$$

Abstract syntax

$$
\begin{array}{lll}
\text{arrays} & \alpha & ::= \; [l \ldots]_{n\ldots} \\
\text{elements} & l & ::= \; e \mid b \mid f \\
\text{functions} & f & ::= \; \pi \mid \; (\lambda \, [(x \; n) \ldots] \, e)
\end{array}
$$

Abstract syntax

$$
\begin{array}{lrcl}
\text{arrays} & \alpha & ::= & [l \ldots]_{n\ldots} \\
\text{elements} & l & ::= & e \mid b \mid f \\
\text{functions} & f & ::= & \pi \mid (\lambda\, [(x\ n) \ldots]\ e) \\
\text{expressions} & e & ::= & \alpha \mid x \mid (e\ e\ \ldots)
\end{array}
$$

Abstract syntax

$$
\begin{array}{lll}
\text{arrays} & \alpha & ::= \; [l \ldots]_{n\ldots} \\
\text{elements} & l & ::= \; e \mid b \mid f \\
\text{functions} & f & ::= \; \pi \mid (\lambda \, [(x \; n) \ldots] \, e) \\
\text{expressions} & e & ::= \; \alpha \mid x \mid (e \; e \; \ldots) \\
 & & \\
\text{values} & v & ::= \; b \mid f \mid [b \ldots]_{n\ldots} \mid [f \ldots]_{n\ldots}
\end{array}
$$

Abstract syntax

$$
\begin{array}{lll}
\text{arrays} & \alpha & ::= [l \ldots]_{n\ldots} \\
\text{elements} & l & ::= e \mid b \mid f \\
\text{functions} & f & ::= \pi \mid (\lambda\, [(x\ n) \ldots]\ e) \\
\text{expressions} & e & ::= \alpha \mid x \mid (e\ e\ \ldots) \\
 & & \\
\text{values} & v & ::= b \mid f \mid [b \ldots]_{n\ldots} \mid [f \ldots]_{n\ldots} \\
\text{contexts} & E & ::= \Box \mid (v \ldots E\ e \ldots) \mid [v \ldots E\ e \ldots]_{n\ldots}
\end{array}
$$

Semantics

$$[+].\ [10\ \ 20]_2\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

$$\mapsto_{lift} \quad \begin{bmatrix}+ & +\\ + & +\end{bmatrix}_{2,2}\ \begin{bmatrix}10 & 10\\ 20 & 20\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

$$\mapsto_{map} \quad \begin{bmatrix}([+].[10].[3].) & ([+].[10].[5].)\\ ([+].[20].[7].) & ([+].[20].[9].)\end{bmatrix}_{2,2}$$

$$\mapsto_{\delta}{}^{4} \quad \begin{bmatrix}[13].[15].\\ [27].[29].\end{bmatrix}_{2,2}$$

$$\mapsto_{collapse} \quad \begin{bmatrix}13 & 15\\ 27 & 29\end{bmatrix}_{2,2}$$

Semantics

$$[+].\ [10\ \ 20]_2\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

$$\mapsto_{lift} \quad \begin{bmatrix}+ & +\\ + & +\end{bmatrix}_{2,2}\ \begin{bmatrix}10 & 10\\ 20 & 20\end{bmatrix}_{2,2}\ \begin{bmatrix}3 & 5\\ 7 & 9\end{bmatrix}_{2,2}$$

$$\mapsto_{map} \quad \begin{bmatrix}([+].[10].[3].) & ([+].[10].[5].)\\ ([+].[20].[7].) & ([+].[20].[9].)\end{bmatrix}_{2,2}$$

$$\mapsto_{\delta}{}^{4} \quad \begin{bmatrix}[13].[15].\\ [27].[29].\end{bmatrix}_{2,2}$$

$$\mapsto_{collapse} \quad \begin{bmatrix}13 & 15\\ 27 & 29\end{bmatrix}_{2,2}$$

Semantics

$$([f \ldots]_{n \ldots}\ v \ldots) \quad \mapsto_{lift}$$

?

Semantics

$$\left(\left[f \ldots\right]_{n \ldots} v \ldots\right) \quad \mapsto_{lift}$$

?

find: $\rho \cdots$ the functions' expected ranks

Semantics

$$([f \ldots]_{n\ldots}\ v \ldots) \quad \mapsto_{lift}$$

?

find: $\rho \cdots$ the functions' expected ranks

$n' \ \ldots$ the principal frame (based on argument shapes)

Semantics

$$([f \ldots]_{n\ldots}\ v \ldots) \quad \mapsto_{lift}$$

?

find: $\rho \ \cdots$ the functions' expected ranks

$n' \ \ldots$ the principal frame (based on argument shapes)

for each v_j, the shape $m_j \ \cdots$ of its cells (according to $\rho \ \ldots$)

Semantics

$$([f \ldots]_{n \ldots}\ v \ldots) \quad \mapsto_{lift}$$

?

find: $\rho \ \cdots$ the functions' expected ranks

$n' \ \ldots$ the principal frame (based on argument shapes)

for each v_j, the shape $m_j \ \cdots$ of its cells (according to $\rho \ \ldots$)

side-condition:

Semantics

$$([f \ldots]_{n\ldots}\ v \ldots) \quad \mapsto_{lift}$$

?

find: $\rho \ \cdots$ the functions' expected ranks

$n' \ \ldots$ the principal frame (based on argument shapes)

for each v_j, the shape $m_j \ \ldots$ of its cells (according to $\rho \ \ldots$)

side-condition: $0 \notin n' \ \ldots$

Semantics

$$([f \ldots]_{n \ldots}\ v \ldots) \quad \mapsto_{lift}$$

?

find: $\rho \ \cdots$ the functions' expected ranks

$n' \ \ldots$ the principal frame (based on argument shapes)

for each v_j, the shape $m_j \ \ldots$ of its cells (according to $\rho \ \ldots$)

side-condition: $0 \notin n' \ \ldots$

$Rank \ [\![v_j]\!] - \rho_j$ not the same for all j

Semantics

$$([f \ldots]_{n\ldots}\ v \ldots) \quad \mapsto_{lift}$$

$$\left(Dup_{0,n'\ \ldots}\ [\![[f\ \ldots]_{n\ \ldots}]\!]\ Dup_{\rho,n'\ \ldots\ m'\ \ldots}\ [\![v]\!]\ \ldots\right)$$

find: $\rho\ \cdots$ the functions' expected ranks
$n'\ \ldots$ the principal frame (based on argument shapes)
for each v_j, the shape $m_j\ \ldots$ of its cells (according to $\rho\ \ldots$)

side-condition: $0 \notin n'\ \ldots$
$Rank\ [\![v_j]\!] - \rho_j$ not the same for all j

Semantics

$$([f \ldots]_{n\ldots}\ v \ldots) \quad \mapsto_{lift}$$

$$\left(Dup_{0,n'\ldots}\ [\![[f\ \ldots]_{n\ \ldots}]\!]\ Dup_{\rho,n'\ \ldots\ m'\ \ldots}\ [\![v]\!]\ \ldots\right)$$

$Dup_{\rho,n\ \ldots}\ [\![\alpha]\!]$: repeat each ρ-cell of α in-place to produce an array of shape *n* ...

Semantics

$$\texttt{[+].}\ \texttt{[10 20]}_{\texttt{2}}\ \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{\texttt{2,2}} \mapsto_{lift}$$

?

Semantics

$$\texttt{[+]. [10 20]}_{\texttt{2}} \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{\texttt{2,2}} \mapsto_{lift}$$

?

$$\rho = 0, 0$$

Semantics

$$\texttt{[+]. [10 20]}_{\texttt{2}} \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{\texttt{2,2}} \mapsto_{lift}$$

?

$$\rho = 0, 0$$

$$n' \ \ldots = 2, 2$$

Semantics

$$\texttt{[+]}. \; \texttt{[10 20]}_{2} \; \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{2,2} \mapsto_{lift}$$

$$\begin{bmatrix} \texttt{+} & \texttt{+} \\ \texttt{+} & \texttt{+} \end{bmatrix}_{2,2} \; \begin{bmatrix} \texttt{10} & \texttt{10} \\ \texttt{20} & \texttt{20} \end{bmatrix}_{2,2} \; \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{2,2}$$

$$\rho = 0, 0$$

$$n' \; \ldots = 2, 2$$

Semantics

$$([f\ \ldots]_{n\,\ldots}\ v\ldots) \quad \mapsto_{map}$$

$$[([f]_{\bullet}\ \alpha\ \ldots)\ \ldots]_{n\,\ldots}$$

find: $\rho\ \cdots$ the functions' expected ranks

$(\alpha\ \ldots)\ \ldots$ the collated cells of $v\ \ldots$

side-condition: $Rank\ [\![v_j]\!] - n_j = k > 0$ for all j

i.e., $n\ \ldots$ is the frame shape for all $v\ \ldots$

Semantics

$$([f\ \ldots]_{n\,\ldots}\ v\ldots) \quad \mapsto_{map}$$

$$[([f]_\bullet\ \alpha\ \ldots)\ \ldots]_{n\,\ldots}$$

$$((\alpha\ \ldots)\ \ldots) = (Cells_\rho\,[\![v]\!]\ \ldots)^\top$$

$Cells_\rho\,[\![\alpha]\!]$: separate out the ρ-cells of α

transpose to get: $((v_{0,0}\ v_{1,0}\ \ldots)\ (v_{0,1}\ v_{1,1}\ \ldots)\ldots)$

apply f_0 to all the 0th cells, f_1 to all the 1st cells, etc.

Semantics

$$\begin{bmatrix} \texttt{+} & \texttt{+} \\ \texttt{+} & \texttt{+} \end{bmatrix}_{\texttt{2,2}} \begin{bmatrix} \texttt{10} & \texttt{10} \\ \texttt{20} & \texttt{20} \end{bmatrix}_{\texttt{2,2}} \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{\texttt{2,2}} \mapsto_{map}$$

?

Semantics

$$\begin{bmatrix} \texttt{+} & \texttt{+} \\ \texttt{+} & \texttt{+} \end{bmatrix}_{\texttt{2,2}} \begin{bmatrix} \texttt{10} & \texttt{10} \\ \texttt{20} & \texttt{20} \end{bmatrix}_{\texttt{2,2}} \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{\texttt{2,2}} \mapsto_{map}$$

?

$$\rho = 0, 0$$

Semantics

$$\begin{bmatrix} \texttt{+} & \texttt{+} \\ \texttt{+} & \texttt{+} \end{bmatrix}_{2,2} \begin{bmatrix} \texttt{10} & \texttt{10} \\ \texttt{20} & \texttt{20} \end{bmatrix}_{2,2} \begin{bmatrix} \texttt{3} & \texttt{5} \\ \texttt{7} & \texttt{9} \end{bmatrix}_{2,2} \mapsto_{map}$$

?

$$\rho = 0, 0$$

$$(\alpha \ \ldots) \ \ldots = \texttt{((10 3) (10 5) (20 7) (20 9))}$$

Semantics

$$\begin{bmatrix} + & + \\ + & + \end{bmatrix}_{2,2} \begin{bmatrix} 10 & 10 \\ 20 & 20 \end{bmatrix}_{2,2} \begin{bmatrix} 3 & 5 \\ 7 & 9 \end{bmatrix}_{2,2} \mapsto_{map}$$

$$\begin{bmatrix} (\,[+].[10].[3].\,) & (\,[+].[10].[5].\,) \\ (\,[+].[20].[7].\,) & (\,[+].[20].[9].\,) \end{bmatrix}_{2,2}$$

$$\rho = 0, 0$$

$$(\alpha \ldots) \ldots = \texttt{((10 3) (10 5) (20 7) (20 9))}$$

Semantics

$$([\pi]_\bullet\ v \ldots) \quad \mapsto_\delta \quad \delta_\pi\,[v\ \ldots]$$

side-condition: $Argrank\,[\![\pi]\!] = (Rank\,[\![v]\!]\ \ldots)$

$$[\texttt{+}]_\bullet\ [\texttt{10}]_\bullet\ [\texttt{3}]_\bullet\ \mapsto_\delta\ [\texttt{13}]_\bullet$$

Semantics

$$\begin{bmatrix} (\,[+].[10].[3].\,) & (\,[+].[10].[5].\,) \\ (\,[+].[20].[7].\,) & (\,[+].[20].[9].\,) \end{bmatrix}_{2,2}$$

$$\mapsto_{\delta}{}^{4} \quad \begin{bmatrix} [13].[15]. \\ [27].[29]. \end{bmatrix}_{2,2}$$

Semantics

$$[\alpha \ \ldots]_{n\,\ldots} \quad \mapsto_{collapse} \quad [Atoms\,[\![\alpha]\!] \ \ldots]_{n\,\ldots,Shape[\![\alpha]\!]}$$

side-condition: all α have the same shape
no α is an application form

$$\begin{bmatrix} [13]. & [15]. \\ [27]. & [29]. \end{bmatrix}_{2,2} \quad \mapsto_{collapse} \quad \begin{bmatrix} 13 & 15 \\ 27 & 29 \end{bmatrix}_{2,2}$$

Stumbling blocks

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

argument frames have mismatched shape

Stumbling blocks

```
[+]. [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

Stumbling blocks

```
[+]. [1 2]₂ [3 4 5]₃
```

argument frames have mismatched shape

```
[[1 2  ]₂]
[[3 4 5]₃]₂
```

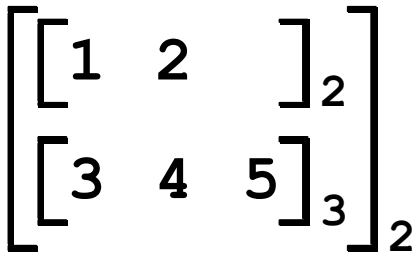
result cells have mismatched shape

```
[+]. [ ]₀ [1].
```

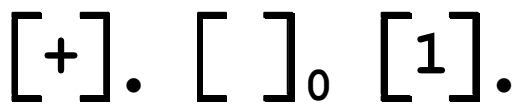
frame is an empty array
indeterminate result cell shape

Stumbling blocks

argument frames have mismatched shape



result cells have mismatched shape



frame is an empty array
indeterminate result cell shape

solution: type system

- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

Abstract syntax (Remora)

types $\quad \tau, \sigma \ ::= \ B \mid x \mid (\tau \ \ldots \ \rightarrow \sigma) \mid \mathtt{A}_\iota \tau$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid \ (\forall \, [x \ \ldots] \, \tau) \ \mid \ (\Pi \, [(x \ \gamma) \ \ldots] \, \tau)$

Abstract syntax (Remora)

$$
\begin{array}{lll}
\text{types} & \tau, \sigma & ::= B \mid x \mid (\tau \ \ldots \ \rightarrow \sigma) \mid \mathtt{A}_{\iota}\tau \\
& & \quad \mid (\forall\,[x \ \ldots]\,\tau) \mid (\Pi\,[(x\ \gamma) \ \ldots]\,\tau) \\
\text{type indices} & \iota, \kappa & ::= n \mid x \mid (\mathtt{S}\ \iota \ \ldots) \mid (+\ \iota\ \kappa)
\end{array}
$$

Abstract syntax (Remora)

$$
\begin{array}{lll}
\text{types} & \tau, \sigma & ::= B \mid x \mid (\tau \ \ldots \ \rightarrow \sigma) \mid \mathtt{A}_{\iota}\tau \\
 & & \quad \mid (\forall\,[x \ \ldots]\,\tau) \mid (\Pi\,[(x\ \gamma) \ \ldots]\,\tau) \\
\text{type indices} & \iota, \kappa & ::= n \mid x \mid (\mathtt{S}\ \iota \ \ldots) \mid (+\ \iota\ \kappa) \\
\text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE}
\end{array}
$$

Abstract syntax (Remora)

$$
\begin{array}{lll}
\text{types} & \tau, \sigma & ::= B \mid x \mid (\tau \ \ldots \ \rightarrow \sigma) \mid \mathtt{A}_{\iota}\tau \\
 & & \mid (\forall\,[x \ \ldots]\,\tau) \mid (\Pi\,[(x\ \gamma) \ \ldots]\,\tau) \\
\text{type indices} & \iota, \kappa & ::= n \mid x \mid (\mathtt{S}\ \iota \ \ldots) \mid (+\ \iota\ \kappa) \\
\text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE} \\
\text{expressions} & e & ::= \ldots \mid (\mathtt{T}\lambda\,[x \ \ldots]\,e) \mid (\mathtt{T\text{-}APP}\ e\ \tau \ \ldots) \\
 & & \mid (\mathtt{I}\lambda\,[(x\ \gamma) \ \ldots]\,e) \mid (\mathtt{I\text{-}APP}\ e\ \iota \ \ldots)
\end{array}
$$

Abstract syntax (Remora)

$$
\begin{array}{lll}
\text{types} & \tau, \sigma & ::= B \mid x \mid (\tau \ \ldots \ \rightarrow \sigma) \mid \mathtt{A}_{\iota}\tau \\
 & & \mid (\forall\,[x \ \ldots]\,\tau) \mid (\Pi\,[(x\ \gamma) \ \ldots]\,\tau) \\
\text{type indices} & \iota, \kappa & ::= n \mid x \mid (\mathtt{S}\ \iota \ \ldots) \mid (+\ \iota\ \kappa) \\
\text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE} \\
\text{expressions} & e & ::= \ldots \mid (\mathtt{T}\lambda\,[x \ \ldots]\,e) \mid (\mathtt{T\text{-}APP}\ e\ \tau \ \ldots) \\
 & & \mid (\mathtt{I}\lambda\,[(x\ \gamma) \ \ldots]\,e) \mid (\mathtt{I\text{-}APP}\ e\ \iota \ \ldots) \\
\text{arrays} & \alpha & ::= [l \ \ldots]^{\tau}
\end{array}
$$

Abstract syntax (Remora)

$$
\begin{array}{lll}
\text{types} & \tau, \sigma & ::= B \mid x \mid (\tau \ \ldots \ \rightarrow \sigma) \mid \mathtt{A}_{\iota}\tau \\
 & & \mid (\forall\,[x \ \ldots]\,\tau) \mid (\Pi\,[(x\ \gamma) \ \ldots]\,\tau) \\
\text{type indices} & \iota, \kappa & ::= n \mid x \mid (\mathtt{S}\ \iota \ \ldots) \mid (+\ \iota\ \kappa) \\
\text{index sorts} & \gamma & ::= \mathtt{NAT} \mid \mathtt{SHAPE} \\
\text{expressions} & e & ::= \ldots \mid (\mathtt{T}\lambda\,[x \ \ldots]\,e) \mid (\mathtt{T\text{-}APP}\ e\ \tau \ \ldots) \\
 & & \mid (\mathtt{I}\lambda\,[(x\ \gamma) \ \ldots]\,e) \mid (\mathtt{I\text{-}APP}\ e\ \iota \ \ldots) \\
\text{arrays} & \alpha & ::= [l \ \ldots]^{\tau} \\
\text{functions} & f & ::= \pi \mid (\lambda\,[(x\ \tau) \ \ldots]\ e)
\end{array}
$$

Typing rules

$$\frac{\Gamma; \Delta; \Theta \vdash l_j : \tau \quad \text{for each } l_j \in l \ \ldots \qquad \mathit{Product}\,[\![n \ \ldots]\!] = \mathit{Length}\,[\![\mathit{elt} \ \ldots]\!]}{\Gamma; \Delta; \Theta \vdash [l \ \ldots]^{\mathtt{A}(\mathtt{S}\ n \ \ldots)\tau} : \mathtt{A}_{(\mathtt{S}\ n \ \ldots)}\tau}$$

Typing rules

$$\frac{\Gamma; \Delta; \Theta \vdash l_j : \tau \quad \text{for each } l_j \in l \ \ldots \qquad \mathit{Product}\,[\![n \ \ldots]\!] = \mathit{Length}\,[\![\mathit{elt} \ \ldots]\!]}{\Gamma; \Delta; \Theta \vdash [l \ \ldots]^{\mathtt{A}(\mathtt{S}\ n \ \ldots)\tau} : \mathtt{A}_{(\mathtt{S}\ n \ \ldots)}\tau}$$

each element must match annotated element type

Typing rules

$$\frac{\Gamma; \Delta; \Theta \vdash l_j : \tau \quad \text{for each } l_j \in l \ \ldots \qquad \mathit{Product}\,[\![n \ \ldots]\!] = \mathit{Length}\,[\![\mathit{elt} \ \ldots]\!]}{\Gamma; \Delta; \Theta \vdash [l \ \ldots]^{\mathtt{A}(\mathtt{S}\ n \ \ldots)\tau} : \mathtt{A}_{(\mathtt{S}\ n \ \ldots)}\tau}$$

each element must match annotated element type

must have as many elements as the product of the dimensions

Typing rules

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

Typing rules

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

type variables not allowed to represent array types

e.g., $\texttt{A}_{(\texttt{S})}\tau$ represents "any scalar type" instead of "any type"

Typing rules

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

Typing rules

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

actual arguments must be frames around expected argument types

Typing rules

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

actual arguments must be frames around expected argument types

set of frames must have maximum under prefix ordering

Type system

shape-based type system offers shape-based safety:

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

Type system

$$\iota' = Max\,[\![\iota, \kappa\ \ldots]\!]$$

prevents mismatched argument frames

Type system

$$\iota' = Max\,[\![\iota, \kappa\ \ldots]\!]$$

prevents mismatched argument frames

$$\Gamma; \Delta; \Theta \vdash l_j : \tau \quad \text{for each } l_j \in l\ \ldots$$

prevents mismatched result cell shapes

Type system

$$\iota' = Max\,[\![\iota, \kappa\ \ldots]\!]$$

prevents mismatched argument frames

$$\Gamma; \Delta; \Theta \vdash l_j : \tau \quad \text{for each } l_j \in l\ \ldots$$

prevents mismatched result cell shapes

$$(\sigma\ \ldots\ \rightarrow \tau)$$

the necessary information about cell shape

- Introduction: Iverson's languages
- Development of shape polymorphism
- Generalizing further
- Core calculus for shape polymorphism
- Type system
- Conclusion: Near future work

Type inference

```
(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)))]))])))
```

Type inference

```
(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)))]))])))
```

```
(A [] [(λ ([f-l 0] [f-r 0] [f-j 0])
          (A [] [(λ [(x +inf.0) (y +inf.0)]
                    (f-j (f-l x) (f-r y)))]))])
```

Type inference difficulties

type information encodes expected rank: type-directed lifting

Type inference difficulties

type information encodes expected rank: type-directed lifting

two instantiations of a `fold` operation for a 4×3 matrix:

$$\left(\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S}\ )}\mathtt{Num}\right) \quad \text{vs} \quad \left(\mathtt{A}_{(\mathtt{S}\ 4,3)}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Num}\right)$$

Type inference difficulties

type information encodes expected rank: type-directed lifting

two instantiations of a `fold` operation for a `4×3` matrix:

$$\left(\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S}\ )}\mathtt{Num}\right) \quad \text{vs} \quad \left(\mathtt{A}_{(\mathtt{S}\ 4,3)}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Num}\right)$$

choice of types and indices affects result

Type inference difficulties

type information encodes expected rank: type-directed lifting

two instantiations of a `fold` operation for a 4×3 matrix:

$\left(\mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S}\ )}\mathtt{Num}\right)$ vs $\left(\mathtt{A}_{(\mathtt{S}\ 4,3)}\mathtt{Num} \to \mathtt{A}_{(\mathtt{S}\ 3)}\mathtt{Num}\right)$

choice of types and indices affects result

$$(\lambda\ [(x : \_)(y : \_)]\ ([+]_{\bullet}\ x\ y))$$

Type inference

`(<function-expr> arg1 arg2)`

infer function is 0,0→0

Type inference

`(<function-expr> arg1 arg2)`

infer function is 0,0→0

arguments require 1,2→2 (need to go beyond unification)

Type inference

`(<function-expr> arg1 arg2)`

infer function is 0,0→0

arguments require 1,2→2 (need to go beyond unification)

use regular lift? rerank? which reranking?

Type inference

```
(<fold-op> arg3)
```

Type inference

`(<fold-op> arg3)`

type inference chooses which axis

Type inference

`(<fold-op> arg3)`

type inference chooses which axis

poor fit for unification vars/type environment?

Type inference

What explicit annotations are reasonable to expect?

What sort of inference strategy is appropriate?

What lifting/reranking should be done implicitly?

Questions?