What every machine-learning researcher should know about AD

Jeffrey Mark Siskind

PURDUE UNIVERSITY

Friay 10 April 2015

Daniel Paul Barrett
Anchal Dube
Pranay Gupta
Seungwoon Ko
Alexey Andreyevich Radul
Barak Avrum Pearlmutter

This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-10-2-0060. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

- From programs to gradients

$$\overleftarrow{\mathcal{J}} : f \mapsto \nabla f$$

- From programs to gradients

$$\overleftarrow{\mathcal{J}} : f \mapsto \nabla f$$

- From gradients to optimization

$$\text{argmax } f \stackrel{\triangle}{=} \text{fixpoint of } \{\mathbf{x}_{i+1} = \mathbf{x}_i - (\nabla\nabla f(\mathbf{x}_i))^{-1}\nabla f(\mathbf{x}_i)\}$$

- From programs to gradients

$$\overleftarrow{\mathcal{J}} : f \mapsto \nabla f$$

- From gradients to optimization

$$\text{argmax } f \triangleq \text{fixpoint of } \{\mathbf{x}_{i+1} = \mathbf{x}_i - (\nabla\nabla f(\mathbf{x}_i))^{-1}\nabla f(\mathbf{x}_i)\}$$

- From optimization to a science of design

PERFORMANCE SPLINECONTROLPOINTS $\triangleq$
  let WING $\triangleq$ SPLINETOSURFACE SPLINECONTROLPOINTS;
    AIRFLOW $\triangleq$ PDESOLVER (WING, NAVIERSTOKES);
    LIFT, DRAG $\triangleq$ SURFACEINTEGRAL (WING, AIRFLOW, FORCE)
  in DESIGNMETRIC (LIFT, DRAG, (WEIGHT WING))

argmax PERFORMANCE

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

Taylor, B. (1715). *Methodus Incrementorum Directa et Inversa*. London.

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε,

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε,

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1!

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

Preserves control flow: Augments original values with derivatives.

The Essence of Forward-Mode AD

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

Preserves control flow: Augments original values with derivatives.

$(\mathcal{D}\, f)$ is $\mathcal{O}(1)$ relative to f (both space and time).

Arithmetic on Complex Numbers

$a + b\mathrm{i}$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Complex Numbers

$a + b\mathrm{i}$
$\mathrm{i}^2 = -1$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Complex Numbers

$a + b\mathrm{i}$
$\mathrm{i}^2 = -1$

$$(a_1 + b_1\mathrm{i}) + (a_2 + b_2\mathrm{i}) = (a_1 + a_2) + (b_1 + b_2)\mathrm{i}$$
$$(a_1 + b_1\mathrm{i}) \times (a_2 + b_2\mathrm{i}) = (a_1 \times a_2) + (a_1 \times b_2 + a_2 \times b_1)\mathrm{i} + (b_1 \times b_2)\mathrm{i}^2$$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Complex Numbers

$a + b\mathrm{i}$
$\mathrm{i}^2 = -1$

$$(a_1 + b_1\mathrm{i}) + (a_2 + b_2\mathrm{i}) = (a_1 + a_2) + (b_1 + b_2)\mathrm{i}$$
$$(a_1 + b_1\mathrm{i}) \times (a_2 + b_2\mathrm{i}) = (a_1 \times a_2) + (a_1 \times b_2 + a_2 \times b_1)\mathrm{i} + (b_1 \times b_2)\mathrm{i}^2$$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Complex Numbers

$a + b\mathrm{i}$
$\mathrm{i}^2 = -1$

$$(a_1 + b_1\mathrm{i}) + (a_2 + b_2\mathrm{i}) = (a_1 + a_2) + (b_1 + b_2)\mathrm{i}$$
$$(a_1 + b_1\mathrm{i}) \times (a_2 + b_2\mathrm{i}) = (a_1 \times a_2 - b_1 \times b_2) + (a_1 \times b_2 + a_2 \times b_1)\mathrm{i}$$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Complex Numbers

$a + b\mathrm{i}$
$\mathrm{i}^2 = -1$

$$(a_1 + b_1\mathrm{i}) + (a_2 + b_2\mathrm{i}) = (a_1 + a_2) + (b_1 + b_2)\mathrm{i}$$
$$(a_1 + b_1\mathrm{i}) \times (a_2 + b_2\mathrm{i}) = (a_1 \times a_2 - b_1 \times b_2) + (a_1 \times b_2 + a_2 \times b_1)\mathrm{i}$$

$\langle a, b \rangle$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Complex Numbers

$a + b\mathrm{i}$
$\mathrm{i}^2 = -1$

$$(a_1 + b_1\mathrm{i}) + (a_2 + b_2\mathrm{i}) = (a_1 + a_2) + (b_1 + b_2)\mathrm{i}$$
$$(a_1 + b_1\mathrm{i}) \times (a_2 + b_2\mathrm{i}) = (a_1 \times a_2 - b_1 \times b_2) + (a_1 \times b_2 + a_2 \times b_1)\mathrm{i}$$

$\langle a, b \rangle$

$$\langle a_1, b_1 \rangle + \langle a_2, b_2 \rangle = \langle (a_1 + a_2), (b_1 + b_2) \rangle$$
$$\langle a_1, b_1 \rangle \times \langle a_2, b_2 \rangle = \langle (a_1 \times a_2 - b_1 \times b_2), (a_1 \times b_2 + a_2 \times b_1) \rangle$$

Hamilton, W. R. (1837). *Theory of conjugate functions, or algebraic couples; with a preliminary and elementary essay on algebra as the science of pure time*. Transactions of the Royal Irish Academy, **17**(1):293–422.

Arithmetic on Dual Numbers

$x + x'\varepsilon$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

Arithmetic on Dual Numbers

$x + x'\varepsilon$
$\varepsilon^2 = 0$, but $\varepsilon \neq 0$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

Arithmetic on Dual Numbers

$x + x'\varepsilon$
$\varepsilon^2 = 0$, but $\varepsilon \neq 0$

$$(x_1 + x_1'\varepsilon) + (x_2 + x_2'\varepsilon) = (x_1 + x_2) + (x_1' + x_2')\varepsilon$$
$$(x_1 + x_1'\varepsilon) \times (x_2 + x_2'\varepsilon) = (x_1 \times x_2) + (x_1 \times x_2' + x_2 \times x_1')\varepsilon + (x_1' + x_2')\varepsilon^2$$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

Arithmetic on Dual Numbers

$x + x'\varepsilon$
$\varepsilon^2 = 0$, but $\varepsilon \neq 0$

$$(x_1 + x_1'\varepsilon) + (x_2 + x_2'\varepsilon) = (x_1 + x_2) + (x_1' + x_2')\varepsilon$$
$$(x_1 + x_1'\varepsilon) \times (x_2 + x_2'\varepsilon) = (x_1 \times x_2) + (x_1 \times x_2' + x_2 \times x_1')\varepsilon + (x_1' + x_2')\varepsilon^2$$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

Arithmetic on Dual Numbers

$x + x'\varepsilon$
$\varepsilon^2 = 0$, but $\varepsilon \neq 0$

$$(x_1 + x_1'\varepsilon) + (x_2 + x_2'\varepsilon) = (x_1 + x_2) + (x_1' + x_2')\varepsilon$$
$$(x_1 + x_1'\varepsilon) \times (x_2 + x_2'\varepsilon) = (x_1 \times x_2) + (x_1 \times x_2' + x_2 \times x_1')\varepsilon$$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

Arithmetic on Dual Numbers

$x + x'\varepsilon$
$\varepsilon^2 = 0$, but $\varepsilon \neq 0$

$$(x_1 + x_1'\varepsilon) + (x_2 + x_2'\varepsilon) = (x_1 + x_2) + (x_1' + x_2')\varepsilon$$
$$(x_1 + x_1'\varepsilon) \times (x_2 + x_2'\varepsilon) = (x_1 \times x_2) + (x_1 \times x_2' + x_2 \times x_1')\varepsilon$$

$\langle x, x' \rangle$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

Arithmetic on Dual Numbers

$x + x'\varepsilon$
$\varepsilon^2 = 0$, but $\varepsilon \neq 0$

$$(x_1 + x_1'\varepsilon) + (x_2 + x_2'\varepsilon) = (x_1 + x_2) + (x_1' + x_2')\varepsilon$$
$$(x_1 + x_1'\varepsilon) \times (x_2 + x_2'\varepsilon) = (x_1 \times x_2) + (x_1 \times x_2' + x_2 \times x_1')\varepsilon$$

$\langle x, x' \rangle$

$$\langle x_1, x_1' \rangle + \langle x_2, x_2' \rangle = \langle (x_1 + x_2), (x_1' + x_2') \rangle$$
$$\langle x_1, x_1' \rangle \times \langle x_2, x_2' \rangle = \langle (x_1 \times x_2), (x_1 \times x_2' + x_2 \times x_1') \rangle$$

Clifford, W. K. (1873). *Preliminary Sketch of Bi-quaternions*. Proceedings of the London Mathematical Society, **4**:381–95.

$$z = g(f(x))$$

$$\begin{aligned} z &= g(f(x)) \\ &= (f \circ g)(x) \end{aligned}$$

$$\begin{aligned} z &= g(f(x)) \\ &= (f \circ g)(x) \end{aligned}$$

$$\begin{aligned} y &= f(x) \\ z &= g(y) \end{aligned}$$

$$\begin{aligned} z &= g(f(x)) \\ &= (f \circ g)(x) \end{aligned}$$

$$\begin{aligned} y &= f(x) \\ z &= g(y) \end{aligned}$$

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$$

$$\begin{aligned} z &= g(f(x)) \\ &= (f \circ g)(x) \end{aligned}$$

$$\begin{aligned} y &= f(x) \\ z &= g(y) \end{aligned}$$

$$\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}x}$$

$$\mathcal{D}\ (f \circ g)\ x = (\mathcal{D}\ g\ y) \times (\mathcal{D}\ f\ x)$$

$$f = f_1 \circ \cdots \circ f_n$$

$$f = f_1 \circ \cdots \circ f_n$$

$$\begin{aligned} \mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\ &\vdots \\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \end{aligned}$$

$$f = f_1 \circ \cdots \circ f_n$$

$$\begin{aligned} \mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\ &\vdots \\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \end{aligned}$$

$$\mathcal{J}\, f\ \mathbf{x}_0 = (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_1\ \mathbf{x}_0)$$

$$f = f_1 \circ \cdots \circ f_n$$

$$\begin{aligned} \mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\ &\vdots \\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \end{aligned}$$

$$\begin{aligned} \mathcal{J}\, f\ \mathbf{x}_0 &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_1\ \mathbf{x}_0) \\ (\mathcal{J}\, f\ \mathbf{x}_0)^\top &= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \end{aligned}$$

$$\overrightarrow{\mathbf{X}_n} = \mathcal{J}\, f\, \mathbf{x}_0$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n} &= \mathcal{J}\, f\, \mathbf{x}_0 \\ &= (\mathcal{J}\, f_n\, \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_2\, \mathbf{x}_1) \times (\mathcal{J}\, f_1\, \mathbf{x}_0)\end{aligned}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n} &= \mathcal{J}\, f\, \mathbf{x}_0 \\ &= (\mathcal{J}\, f_n\, \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_2\, \mathbf{x}_1) \times (\mathcal{J}\, f_1\, \mathbf{x}_0)\end{aligned}$$

$$\overrightarrow{\mathbf{X}_1} = (\mathcal{J}\, f_1\, \mathbf{x}_0)$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n} &= \mathcal{J}\, f\, \mathbf{x}_0 \\ &= (\mathcal{J}\, f_n\, \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_2\, \mathbf{x}_1) \times (\mathcal{J}\, f_1\, \mathbf{x}_0)\end{aligned}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_1} &= (\mathcal{J}\, f_1\, \mathbf{x}_0) \\ \overrightarrow{\mathbf{X}_2} &= (\mathcal{J}\, f_2\, \mathbf{x}_1) \times \overrightarrow{\mathbf{X}_1}\end{aligned}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n} &= \mathcal{J}\, f\, \mathbf{x}_0 \\ &= (\mathcal{J}\, f_n\, \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_2\, \mathbf{x}_1) \times (\mathcal{J}\, f_1\, \mathbf{x}_0)\end{aligned}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_1} &= (\mathcal{J}\, f_1\, \mathbf{x}_0) \\ \overrightarrow{\mathbf{X}_2} &= (\mathcal{J}\, f_2\, \mathbf{x}_1) \times \overrightarrow{\mathbf{X}_1} \\ &\vdots \\ \overrightarrow{\mathbf{X}_n} &= (\mathcal{J}\, f_n\, \mathbf{x}_{n-1}) \times \overrightarrow{\mathbf{X}_{n-1}}\end{aligned}$$

$$\overline{\mathbf{X}_0} = (\mathcal{J}\, f\ \mathbf{x}_0)^\top$$

$$\begin{aligned}\overline{\mathbf{X}_0} &= (\mathcal{J}\, f\ \mathbf{x}_0)^\top \\ &= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top\end{aligned}$$

$$\overline{\mathbf{X}_0} = (\mathcal{J}\, f\ \mathbf{x}_0)^\top$$
$$= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top$$

$$\overline{\mathbf{X}_{n-1}} = (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top$$

$$\overline{\mathbf{X}_0} = (\mathcal{J}\, f\ \mathbf{x}_0)^\top$$
$$= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top$$

$$\overline{\mathbf{X}_{n-1}} = (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top$$
$$\overline{\mathbf{X}_{n-2}} = (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times \overline{\mathbf{X}_{n-1}}$$

$$\overline{\mathbf{X}_0} = (\mathcal{J}\, f\ \mathbf{x}_0)^\top$$
$$= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top$$

$$\overline{\mathbf{X}_{n-1}} = (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top$$
$$\overline{\mathbf{X}_{n-2}} = (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times \overline{\mathbf{X}_{n-1}}$$
$$\vdots$$
$$\overline{\mathbf{X}_0} = (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \overline{\mathbf{X}_1}$$

$$
\begin{array}{rcl@{\qquad}rcl}
\overrightarrow{\mathbf{X}_1} & = & (\mathcal{J}\, f_1\ \mathbf{x}_0) & \overleftarrow{\mathbf{X}_{n-1}} & = & (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \\
\overrightarrow{\mathbf{X}_2} & = & (\mathcal{J}\, f_2\ \mathbf{x}_1) \times \overrightarrow{\mathbf{X}_1} & \overleftarrow{\mathbf{X}_{n-2}} & = & (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times \overleftarrow{\mathbf{X}_{n-1}} \\
 & \vdots & & & \vdots & \\
\overrightarrow{\mathbf{X}_n} & = & (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \overrightarrow{\mathbf{X}_{n-1}} & \overleftarrow{\mathbf{X}_0} & = & (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \overleftarrow{\mathbf{X}_1}
\end{array}
$$

$$
\begin{array}{rcl}
\overrightarrow{\mathbf{X}_1} & = & (\mathcal{J}\, f_1\ \mathbf{x}_0) \\
\overrightarrow{\mathbf{X}_2} & = & (\mathcal{J}\, f_2\ \mathbf{x}_1) \times \overrightarrow{\mathbf{X}_1} \\
 & \vdots & \\
\overrightarrow{\mathbf{X}_n} & = & (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \overrightarrow{\mathbf{X}_{n-1}}
\end{array}
\qquad
\begin{array}{rcl}
\overleftarrow{\mathbf{X}_{n-1}} & = & (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \\
\overleftarrow{\mathbf{X}_{n-2}} & = & (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times \overleftarrow{\mathbf{X}_{n-1}} \\
 & \vdots & \\
\overleftarrow{\mathbf{X}_0} & = & (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \overleftarrow{\mathbf{X}_1}
\end{array}
$$

$$
\begin{aligned}
\overrightarrow{\mathbf{X}_1} &= (\mathcal{J}\, f_1\ \mathbf{x}_0) \\
\overrightarrow{\mathbf{X}_2} &= (\mathcal{J}\, f_2\ \mathbf{x}_1) \times \overrightarrow{\mathbf{X}_1} \\
&\vdots \\
\overrightarrow{\mathbf{X}_n} &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \overrightarrow{\mathbf{X}_{n-1}}
\end{aligned}
\qquad
\begin{aligned}
\overleftarrow{\mathbf{X}_{n-1}} &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \\
\overleftarrow{\mathbf{X}_{n-2}} &= (\mathcal{J}\, f_{n-1}\ \mathbf{x}_{n-2})^\top \times \overleftarrow{\mathbf{X}_{n-1}} \\
&\vdots \\
\overleftarrow{\mathbf{X}_0} &= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \overleftarrow{\mathbf{X}_1}
\end{aligned}
$$

$$\overline{\mathbf{x}_n'} = (\mathcal{J}\, f\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'}$$

$$\begin{aligned}\overline{\mathbf{x}_n'} &= (\mathcal{J}\, f\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'} \\ &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_1\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'}\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_n'} &= (\mathcal{J}\, f\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'} \\ &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_1\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'}\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_1'} &= (\mathcal{J}\, f_1\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'} \\ &\vdots \\ \overline{\mathbf{x}_n'} &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \overline{\mathbf{x}_{n-1}'}\end{aligned}$$

$$\overline{\mathbf{x}_0} = (\mathcal{J}\, f\ \mathbf{x}_0)^\top \times \overline{\mathbf{x}_n}$$

$$\begin{aligned}\overline{\mathbf{x}_0} &= (\mathcal{J}\, f\ \mathbf{x}_0)^\top \times \overline{\mathbf{x}_n} \\ &= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \times \overline{\mathbf{x}_n}\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_0} &= (\mathcal{J}\, f\ \mathbf{x}_0)^\top \times \overline{\mathbf{x}_n} \\ &= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \times \overline{\mathbf{x}_n}\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_{n-1}} &= (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top \times \overline{\mathbf{x}_n} \\ &\vdots \\ \overline{\mathbf{x}_0} &= (\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \overline{\mathbf{x}_1}\end{aligned}$$

$$\mathbf{y} = \mathbf{A} \times \mathbf{x}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{A} \times \mathbf{x} \\ &= f(\mathbf{x})\end{aligned}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{A} \times \mathbf{x} \\ &= f(\mathbf{x})\end{aligned}$$

$$\overrightarrow{\mathbf{x}_n} = (\mathcal{J}\, f\ \mathbf{x}_0) \times \overrightarrow{\mathbf{x}_0}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{A} \times \mathbf{x} \\ &= f(\mathbf{x})\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_n'} &= (\mathcal{J}\, f\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'} \\ &= \overline{f'}\ \mathbf{x}_0\ \overline{\mathbf{x}_0'}\end{aligned}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{A} \times \mathbf{x} \\ &= f(\mathbf{x})\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_n'} &= (\mathcal{J}\, f\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'} \\ &= \overline{f'}\ \mathbf{x}_0\ \overline{\mathbf{x}_0'}\end{aligned}$$

$$\overline{\mathbf{x}_0} = (\mathcal{J}\, f\ \mathbf{x}_0)^\top \times \overline{\mathbf{x}_n}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{A} \times \mathbf{x} \\ &= f(\mathbf{x})\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_n'} &= (\mathcal{J}\, f\ \mathbf{x}_0) \times \overline{\mathbf{x}_0'} \\ &= \overline{f}'\ \mathbf{x}_0\ \overline{\mathbf{x}_0'}\end{aligned}$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_0} &= (\mathcal{J}\, f\ \mathbf{x}_0)^\top \times \overleftarrow{\mathbf{x}_n} \\ &= \overleftarrow{f}\ \mathbf{x}_0\ \overleftarrow{\mathbf{x}_n}\end{aligned}$$

$$\overrightarrow{\mathbf{X}_n}[;j] = \overrightarrow{f}\ \mathbf{x}_0\ \overrightarrow{\mathbf{e}_j}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n}[;j] &= \overrightarrow{f}\ \mathbf{x}_0\ \overrightarrow{\mathbf{e}_j} \\ &= (\mathcal{J}\, f\ \mathbf{x}_0)[;j]\end{aligned}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n}[;j] &= \overrightarrow{f}\ \mathbf{x}_0\ \overrightarrow{\mathbf{e}_j} \\ &= (\mathcal{J}\, f\ \mathbf{x}_0)[;j]\end{aligned}$$

$$\overleftarrow{\mathbf{X}_0}[;i] = \overleftarrow{f}\ \mathbf{x}_0\ \overleftarrow{\mathbf{e}_i}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n}[;j] &= \overrightarrow{f}\ \mathbf{x}_0\ \overrightarrow{\mathbf{e}_j} \\ &= (\mathcal{J}\ f\ \mathbf{x}_0)[;j]\end{aligned}$$

$$\begin{aligned}\overleftarrow{\mathbf{X}_0}[;i] &= \overleftarrow{f}\ \mathbf{x}_0\ \overleftarrow{\mathbf{e}_i} \\ &= (\mathcal{J}\ f\ \mathbf{x}_0)^\top[;i]\end{aligned}$$

$$\begin{aligned}\overrightarrow{\mathbf{X}_n}[;j] &= \overrightarrow{f}\ \mathbf{x}_0\ \overrightarrow{\mathbf{e}_j} \\ &= (\mathcal{J}\, f\ \mathbf{x}_0)[;j]\end{aligned}$$

$$\begin{aligned}\overleftarrow{\mathbf{X}_0}[;i] &= \overleftarrow{f}\ \mathbf{x}_0\ \overleftarrow{\mathbf{e}_i} \\ &= (\mathcal{J}\, f\ \mathbf{x}_0)^\top[;i] \\ &= (\mathcal{J}\, f\ \mathbf{x}_0)[i;]\end{aligned}$$

$$\mathbf{y} = \mathbf{B} \times (\mathbf{A} \times \mathbf{x})$$

$$\begin{aligned}\mathbf{y} &= \mathbf{B} \times (\mathbf{A} \times \mathbf{x}) \\ &= (\mathbf{B} \times \mathbf{A}) \times \mathbf{x}\end{aligned}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{B} \times (\mathbf{A} \times \mathbf{x}) \\ &= (\mathbf{B} \times \mathbf{A}) \times \mathbf{x} \\ &= g(f(\mathbf{x}))\end{aligned}$$

$$\begin{aligned}\mathbf{y} &= \mathbf{B} \times (\mathbf{A} \times \mathbf{x}) \\ &= (\mathbf{B} \times \mathbf{A}) \times \mathbf{x} \\ &= g(f(\mathbf{x})) \\ &= (f \circ g)(\mathbf{x})\end{aligned}$$

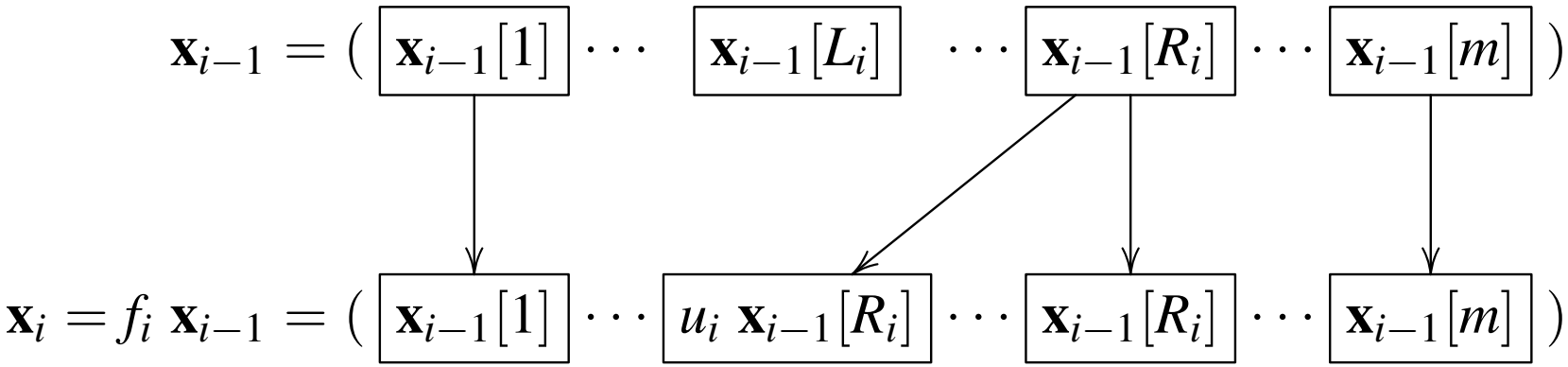
$$\begin{aligned}\mathbf{y} &= \mathbf{B} \times (\mathbf{A} \times \mathbf{x}) \\ &= (\mathbf{B} \times \mathbf{A}) \times \mathbf{x} \\ &= g(f(\mathbf{x})) \\ &= (f \circ g)(\mathbf{x})\end{aligned}$$

$$(\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_1\ \mathbf{x}_0) = (\overline{f_1}'\ \mathbf{x}_0) \circ \cdots \circ (\overline{f_n}'\ \mathbf{x}_{n-1})$$

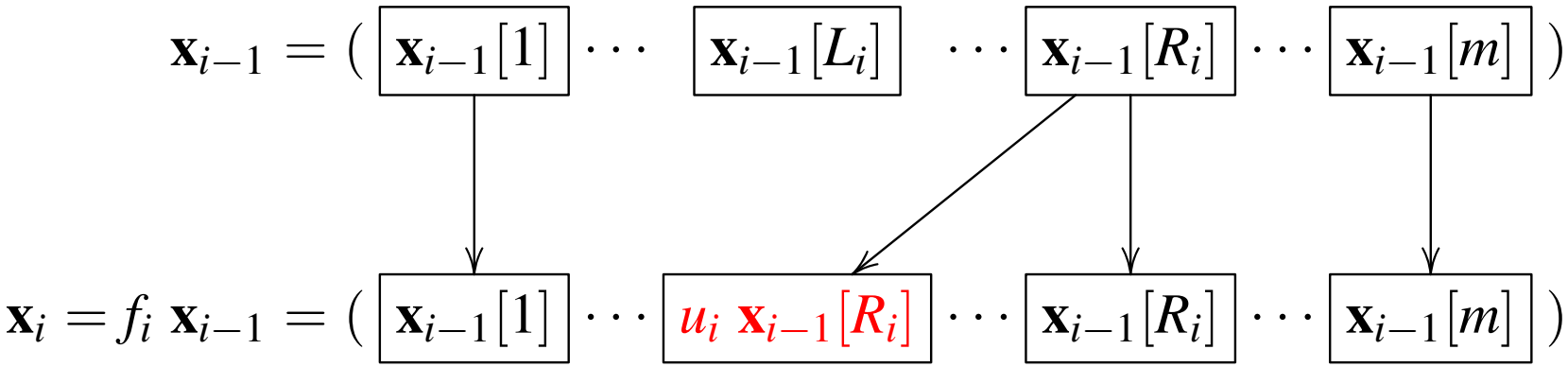
$$\begin{aligned}\mathbf{y} &= \mathbf{B} \times (\mathbf{A} \times \mathbf{x}) \\ &= (\mathbf{B} \times \mathbf{A}) \times \mathbf{x} \\ &= g(f(\mathbf{x})) \\ &= (f \circ g)(\mathbf{x})\end{aligned}$$

$$(\mathcal{J}\, f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\, f_1\ \mathbf{x}_0) = (\overrightarrow{f_1}\ \mathbf{x}_0) \circ \cdots \circ (\overrightarrow{f_n}\ \mathbf{x}_{n-1})$$

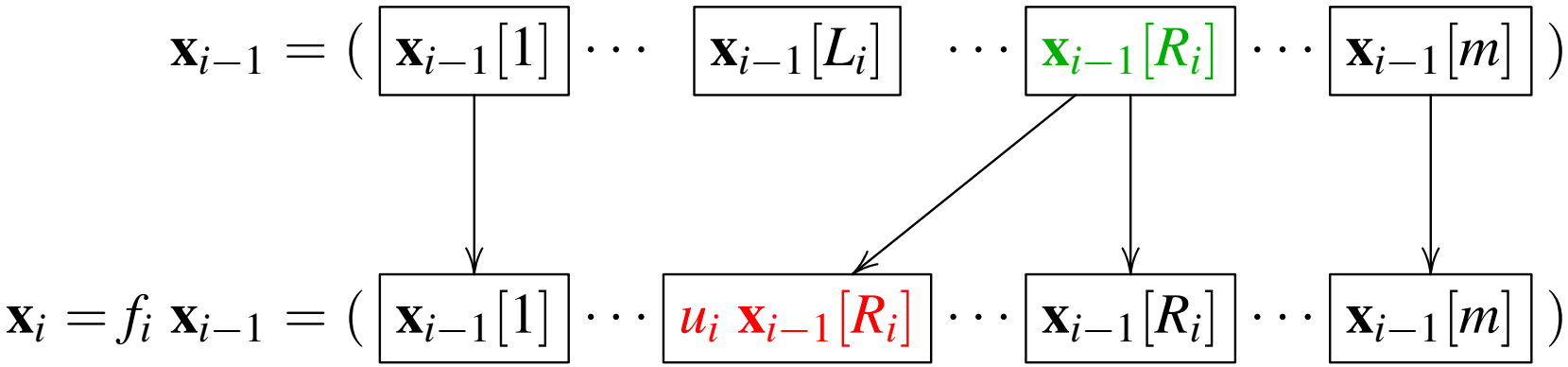
$$(\mathcal{J}\, f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\, f_n\ \mathbf{x}_{n-1})^\top = (\overleftarrow{f_n}\ \mathbf{x}_{n-1}) \circ \cdots \circ (\overleftarrow{f_1}\ \mathbf{x}_0)$$



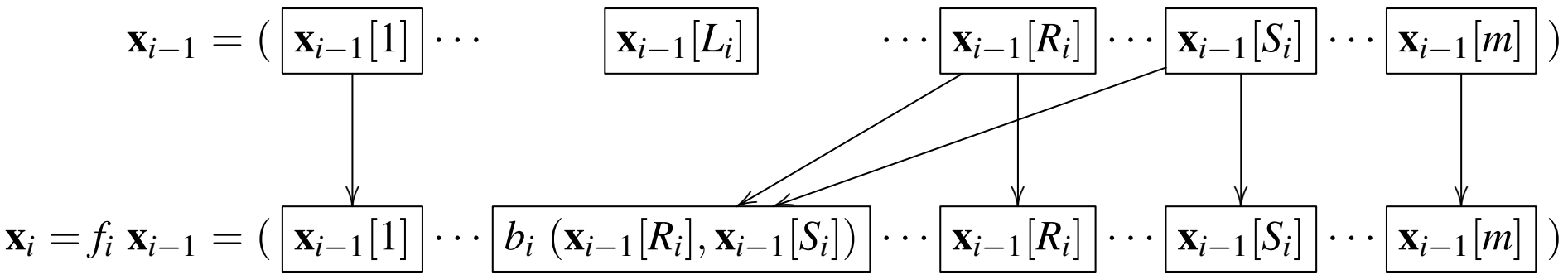
$$\mathbf{x}[L_i] := u_i\ \mathbf{x}[R_i]$$



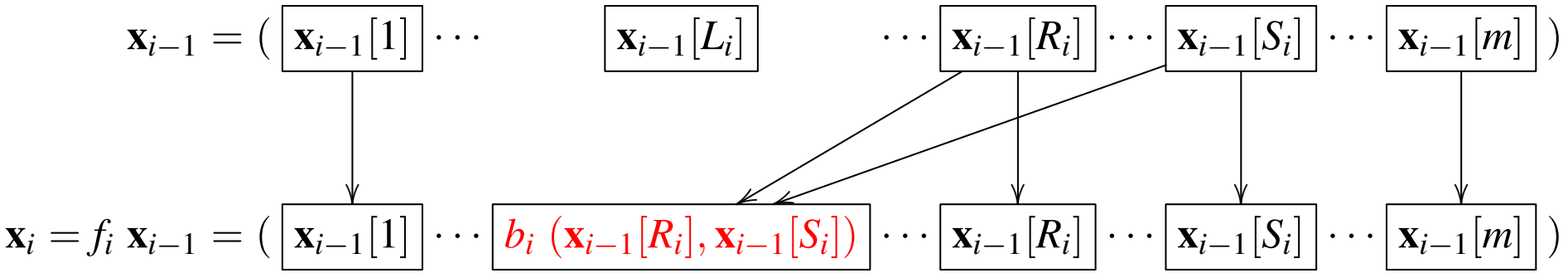
$$\mathbf{x}[L_i] := u_i\ \mathbf{x}[R_i]$$



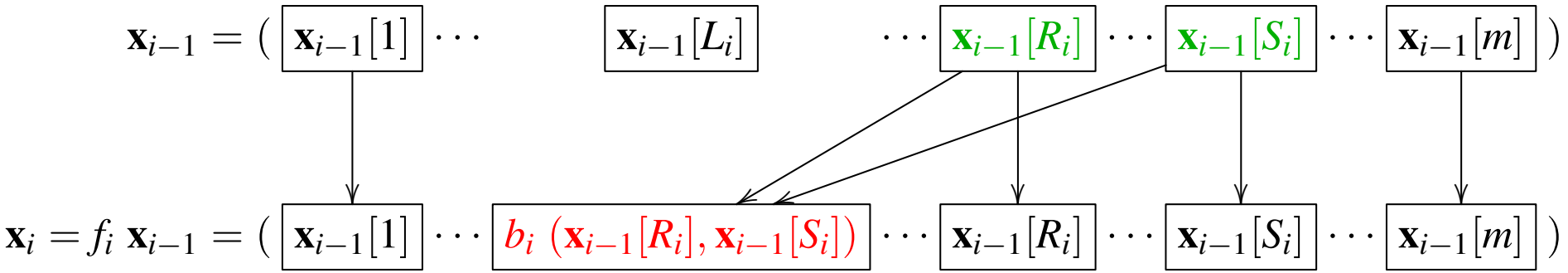
$$\mathbf{x}[L_i] := u_i\ \mathbf{x}[R_i]$$



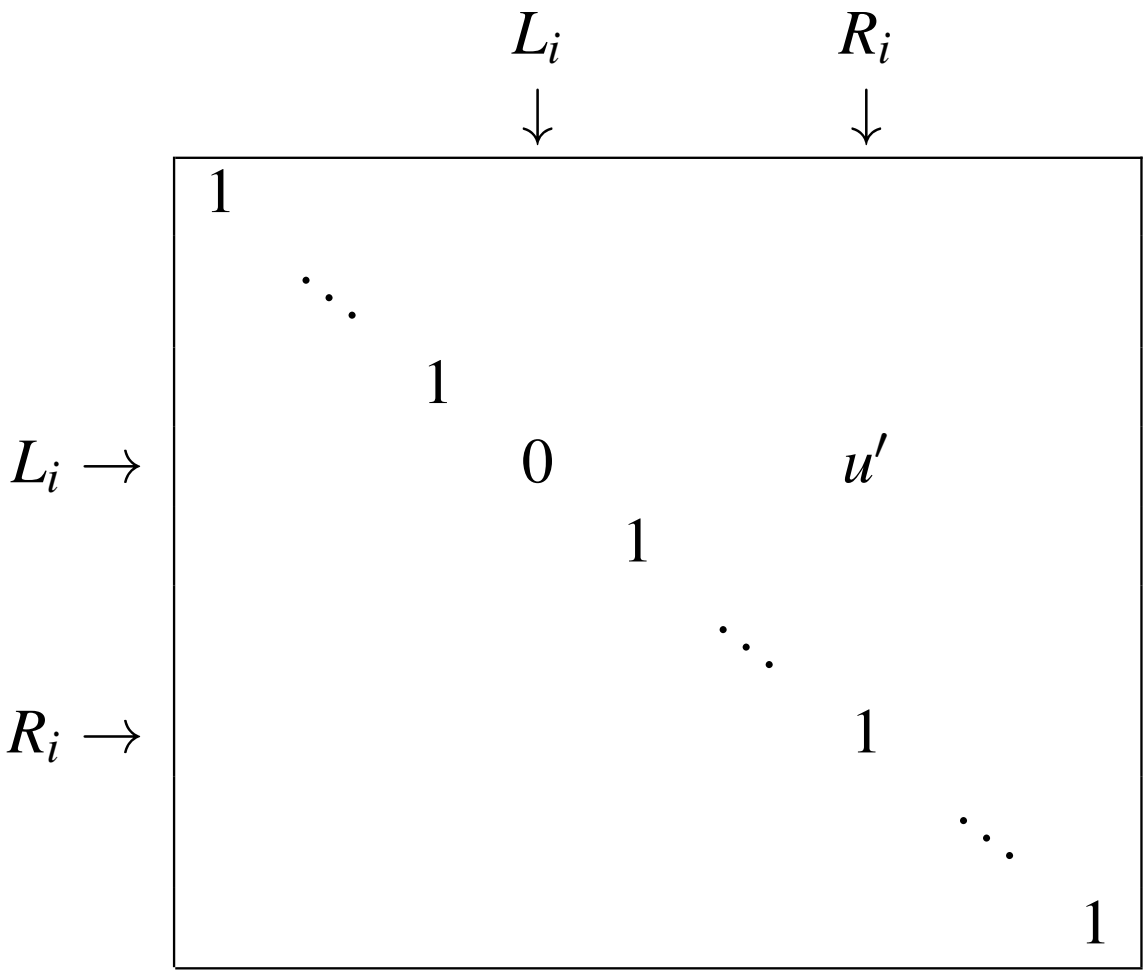
$$\mathbf{x}[L_i] := b\ (\mathbf{x}[R_i], \mathbf{x}[S_i])$$



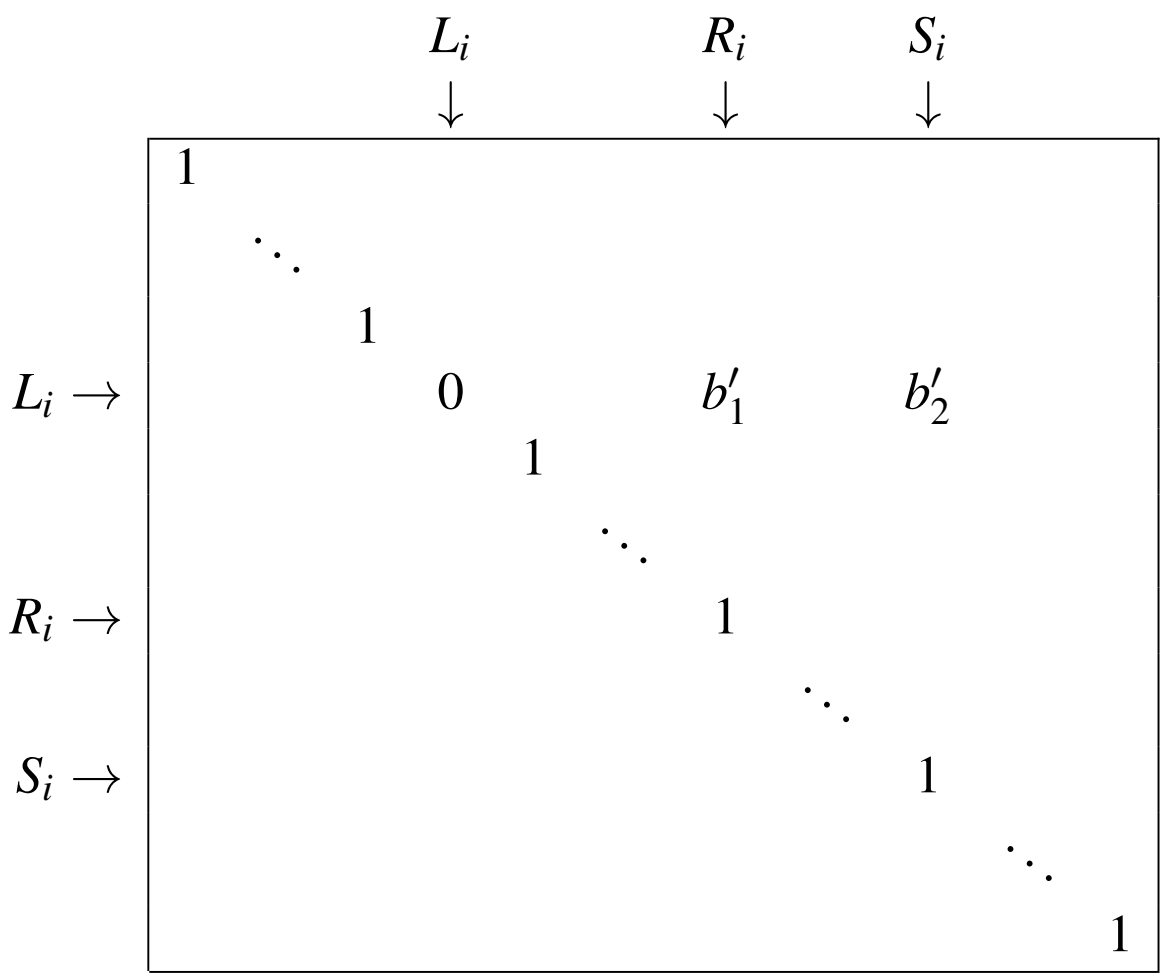
$$\mathbf{x}[L_i] := b\ (\mathbf{x}[R_i], \mathbf{x}[S_i])$$



$$\mathbf{x}[L_i] := b\ (\mathbf{x}[R_i], \mathbf{x}[S_i])$$



$$u' = \mathcal{D}\, u_i\, \mathbf{x}_{i-1}[R_i]$$



$$b'_1 = \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i])$$
$$b'_2 = \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i])$$

$$\overline{\mathbf{x}_i'} \quad = \quad \overline{f_i'}\ \mathbf{x}_{i-1}\ \overline{\mathbf{x}_{i-1}'}$$

$$\begin{aligned}\overrightarrow{\mathbf{x}_i} &= \overrightarrow{f_i}\ \mathbf{x}_{i-1}\ \overrightarrow{\mathbf{x}_{i-1}} \\ &= (\mathcal{J}\, f_i\ \mathbf{x}_{i-1}) \times \overrightarrow{\mathbf{x}_{i-1}}\end{aligned}$$

$$\begin{aligned}\overline{\mathbf{x}_i} &= \overline{f_i}\ \mathbf{x}_{i-1}\ \overline{\mathbf{x}_{i-1}} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overline{\mathbf{x}_{i-1}}\end{aligned}$$

$$\begin{pmatrix} \overline{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[L_i - 1] \\ u' \times \overline{\mathbf{x}_{i-1}}[R_i] \\ \overline{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & 0 & & & u' & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overline{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[L_i - 1] \\ \overline{\mathbf{x}_{i-1}}[L_i] \\ \overline{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[m] \end{pmatrix}$$

$$u' = \mathcal{D}\ u_i\ \mathbf{x}_{i-1}[R_i]$$

$$\begin{aligned}\overline{\mathbf{x}_i} &= \overline{f_i}\ \mathbf{x}_{i-1}\ \overline{\mathbf{x}_{i-1}} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overline{\mathbf{x}_{i-1}}\end{aligned}$$

$$\begin{pmatrix} \overline{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[L_i - 1] \\ u' \times \overline{\mathbf{x}_{i-1}}[R_i] \\ \overline{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & 0 & & u' & & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overline{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[L_i - 1] \\ \overline{\mathbf{x}_{i-1}}[L_i] \\ \overline{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[m] \end{pmatrix}$$

$$u' = \mathcal{D}\ u_i\ \mathbf{x}_{i-1}[R_i]$$

$$\begin{aligned}\overline{\mathbf{x}_i} &= \overline{f_i}\ \mathbf{x}_{i-1}\ \overline{\mathbf{x}_{i-1}} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overline{\mathbf{x}_{i-1}}\end{aligned}$$

$$\begin{pmatrix} \overline{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[L_i - 1] \\ u' \times \overline{\mathbf{x}_{i-1}}[R_i] \\ \overline{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & 0 & & u' & & \\ & & & & 1 & & & \\ & & & & & \ddots & & \\ & & & & & & 1 & \\ & & & & & & & \ddots \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overline{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[L_i - 1] \\ \overline{\mathbf{x}_{i-1}}[L_i] \\ \overline{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overline{\mathbf{x}_{i-1}}[m] \end{pmatrix}$$

$$u' = \mathcal{D}\ u_i\ \mathbf{x}_{i-1}[R_i]$$

$$\overrightarrow{\mathbf{x}_i} \quad = \quad \overrightarrow{f_i}\ \mathbf{x}_{i-1}\ \overrightarrow{\mathbf{x}_{i-1}}$$

$$\begin{aligned}\overline{\mathbf{x}_i'} &= \overline{f_i'}\ \mathbf{x}_{i-1}\ \overline{\mathbf{x}_{i-1}'} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overline{\mathbf{x}_{i-1}'}\end{aligned}$$

$$
\begin{aligned}
\overrightarrow{\mathbf{x}_i} &= \overrightarrow{f_i}\ \mathbf{x}_{i-1}\ \overrightarrow{\mathbf{x}_{i-1}} \\
&= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overrightarrow{\mathbf{x}_{i-1}}
\end{aligned}
$$

$$
\begin{pmatrix}
\overrightarrow{\mathbf{x}_{i-1}}[1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i - 1] \\
b'_1 \times \overrightarrow{\mathbf{x}_{i-1}}[R_i] + b'_2 \times \overrightarrow{\mathbf{x}_{i-1}}[S_i] \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i + 1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[R_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[S_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[m]
\end{pmatrix}
=
\begin{pmatrix}
1 & & & & & & & & & \\
& \ddots & & & & & & & & \\
& & 1 & & & & & & & \\
& & & 0 & & b'_1 & & b'_2 & & \\
& & & & 1 & & & & & \\
& & & & & \ddots & & & & \\
& & & & & & 1 & & & \\
& & & & & & & \ddots & & \\
& & & & & & & & 1 & \\
& & & & & & & & & 1
\end{pmatrix}
\begin{pmatrix}
\overrightarrow{\mathbf{x}_{i-1}}[1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i - 1] \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i] \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i + 1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[R_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[S_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[m]
\end{pmatrix}
$$

$$
\begin{aligned}
b'_1 &= \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \\
b'_2 &= \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i])
\end{aligned}
$$

$$
\begin{aligned}
\overrightarrow{\mathbf{x}_i} &= \overrightarrow{f_i}\ \mathbf{x}_{i-1}\ \overrightarrow{\mathbf{x}_{i-1}} \\
&= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overrightarrow{\mathbf{x}_{i-1}}
\end{aligned}
$$

$$
\begin{pmatrix}
\overrightarrow{\mathbf{x}_{i-1}}[1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i - 1] \\
b'_1 \times \overrightarrow{\mathbf{x}_{i-1}}[R_i] + b'_2 \times \overrightarrow{\mathbf{x}_{i-1}}[S_i] \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i + 1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[R_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[S_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[m]
\end{pmatrix}
=
\begin{pmatrix}
1 & & & & & & & & & \\
& \ddots & & & & & & & & \\
& & 1 & & & & & & & \\
& & & 0 & & b'_1 & & b'_2 & & \\
& & & & 1 & & & & & \\
& & & & & \ddots & & & & \\
& & & & & & 1 & & & \\
& & & & & & & \ddots & & \\
& & & & & & & & 1 & \\
& & & & & & & & & 1
\end{pmatrix}
\begin{pmatrix}
\overrightarrow{\mathbf{x}_{i-1}}[1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i - 1] \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i] \\
\overrightarrow{\mathbf{x}_{i-1}}[L_i + 1] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[R_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[S_i] \\
\vdots \\
\overrightarrow{\mathbf{x}_{i-1}}[m]
\end{pmatrix}
$$

$$
\begin{aligned}
b'_1 &= \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \\
b'_2 &= \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i])
\end{aligned}
$$

$$\begin{aligned}\overrightarrow{\mathbf{x}_i} &= \overrightarrow{f_i}\ \mathbf{x}_{i-1}\ \overrightarrow{\mathbf{x}_{i-1}} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \overrightarrow{\mathbf{x}_{i-1}}\end{aligned}$$

$$\begin{pmatrix} \overrightarrow{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[L_i - 1] \\ b'_1 \times \overrightarrow{\mathbf{x}_{i-1}}[R_i] + b'_2 \times \overrightarrow{\mathbf{x}_{i-1}}[S_i] \\ \overrightarrow{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[S_i] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 0 & & b'_1 & & b'_2 & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & 1 & \\ & & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overrightarrow{\mathbf{x}_{i-1}}[1] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[L_i - 1] \\ \overrightarrow{\mathbf{x}_{i-1}}[L_i] \\ \overrightarrow{\mathbf{x}_{i-1}}[L_i + 1] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[R_i] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[S_i] \\ \vdots \\ \overrightarrow{\mathbf{x}_{i-1}}[m] \end{pmatrix}$$

$$\begin{aligned} b'_1 &= \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \\ b'_2 &= \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \end{aligned}$$

$$\overleftarrow{\mathbf{x}_{i-1}} = \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i}$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i} \\ &= \left(\mathcal{J}\, f_i\ \mathbf{x}_{i-1}\right)^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

$$\begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ 0 \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ u' \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & & & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & u' & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ \overleftarrow{\mathbf{x}_i}[L_i] \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix}$$

$$u' = \mathcal{D}\ u_i\ \mathbf{x}_{i-1}[R_i]$$

$$\begin{aligned}\overline{\mathbf{x}_{i-1}} &= \overline{f_i\ \mathbf{x}_{i-1}}\ \overline{\mathbf{x}_i} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \overline{\mathbf{x}_i}\end{aligned}$$

$$\begin{pmatrix} \overline{\mathbf{x}_i}[1] \\ \vdots \\ \overline{\mathbf{x}_i}[L_i-1] \\ 0 \\ \overline{\mathbf{x}_i}[L_i+1] \\ \vdots \\ u' \times \overline{\mathbf{x}_i}[L_i] + \overline{\mathbf{x}_i}[R_i] \\ \vdots \\ \overline{\mathbf{x}_i}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & & & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & u' & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overline{\mathbf{x}_i}[1] \\ \vdots \\ \overline{\mathbf{x}_i}[L_i-1] \\ \overline{\mathbf{x}_i}[L_i] \\ \overline{\mathbf{x}_i}[L_i+1] \\ \vdots \\ \overline{\mathbf{x}_i}[R_i] \\ \vdots \\ \overline{\mathbf{x}_i}[m] \end{pmatrix}$$

$$u' = \mathcal{D}\ u_i\ \mathbf{x}_{i-1}[R_i]$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i\ \mathbf{x}_{i-1}}\ \overleftarrow{\mathbf{x}_i} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

$$\begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i-1] \\ 0 \\ \overleftarrow{\mathbf{x}_i}[L_i+1] \\ \vdots \\ u' \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & & & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & u' & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i-1] \\ \overleftarrow{\mathbf{x}_i}[L_i] \\ \overleftarrow{\mathbf{x}_i}[L_i+1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix}$$

$$u' = \mathcal{D}\ u_i\ \mathbf{x}_{i-1}[R_i]$$

$$\overline{\mathbf{x}_{i-1}} \quad = \quad \overline{f_i}\ \mathbf{x}_{i-1}\ \overline{\mathbf{x}_i}$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i} \\ &= (\mathcal{J}\, f_i\ \mathbf{x}_{i-1})^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

$$\begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ 0 \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ b'_1 \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ b'_2 \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[S_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & 0 & & & & & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & b'_1 & & & 1 & & & & \\ & & & & & & & \ddots & & & \\ & & & b'_2 & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ \overleftarrow{\mathbf{x}_i}[L_i] \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[S_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix}$$

$$\begin{aligned} b'_1 &= \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \\ b'_2 &= \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \end{aligned}$$

$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

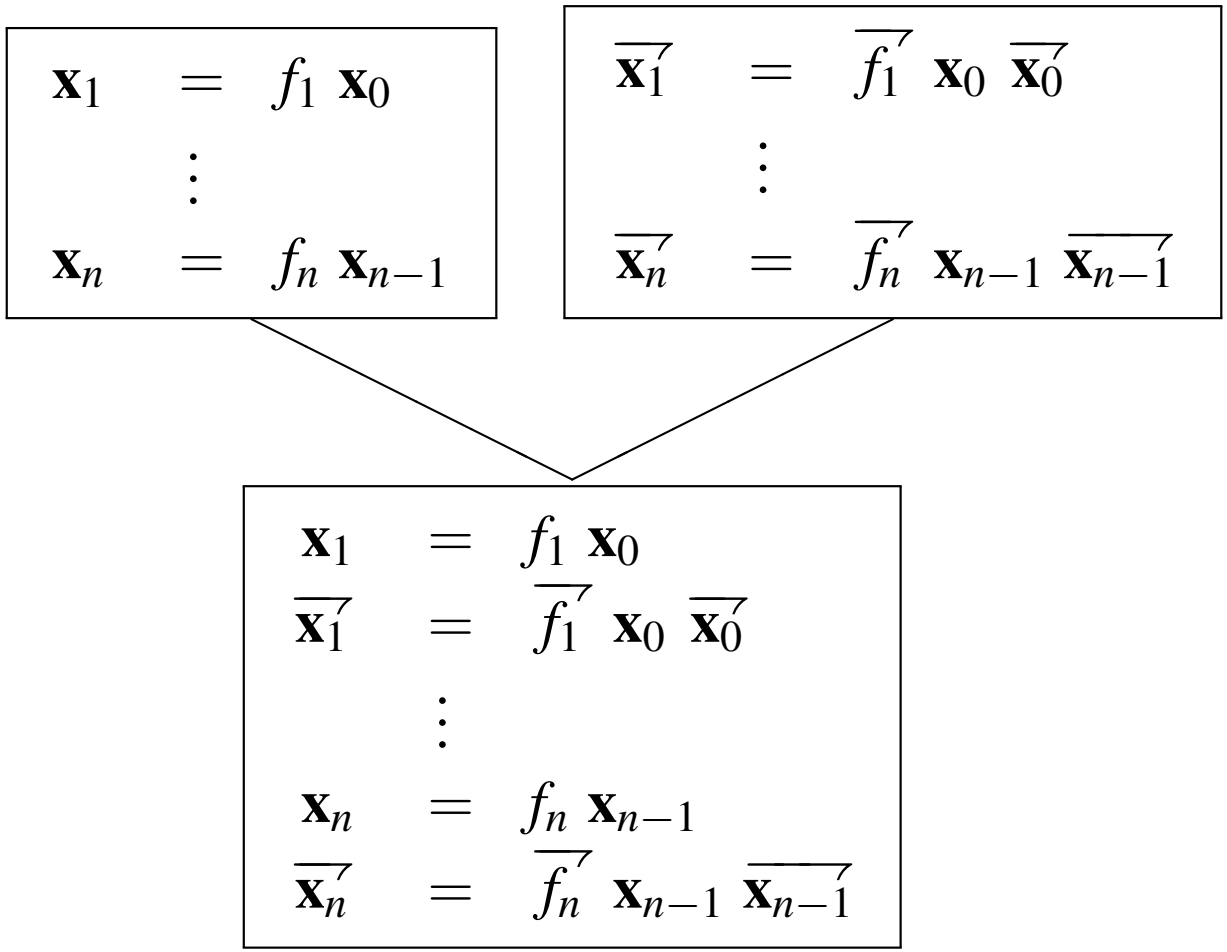
$$\begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ 0 \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ b'_1 \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ b'_2 \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[S_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 0 & & & & & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & b'_1 & & & 1 & & & \\ & & & & & & & \ddots & & \\ & & & b'_2 & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ \overleftarrow{\mathbf{x}_i}[L_i] \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[S_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix}$$

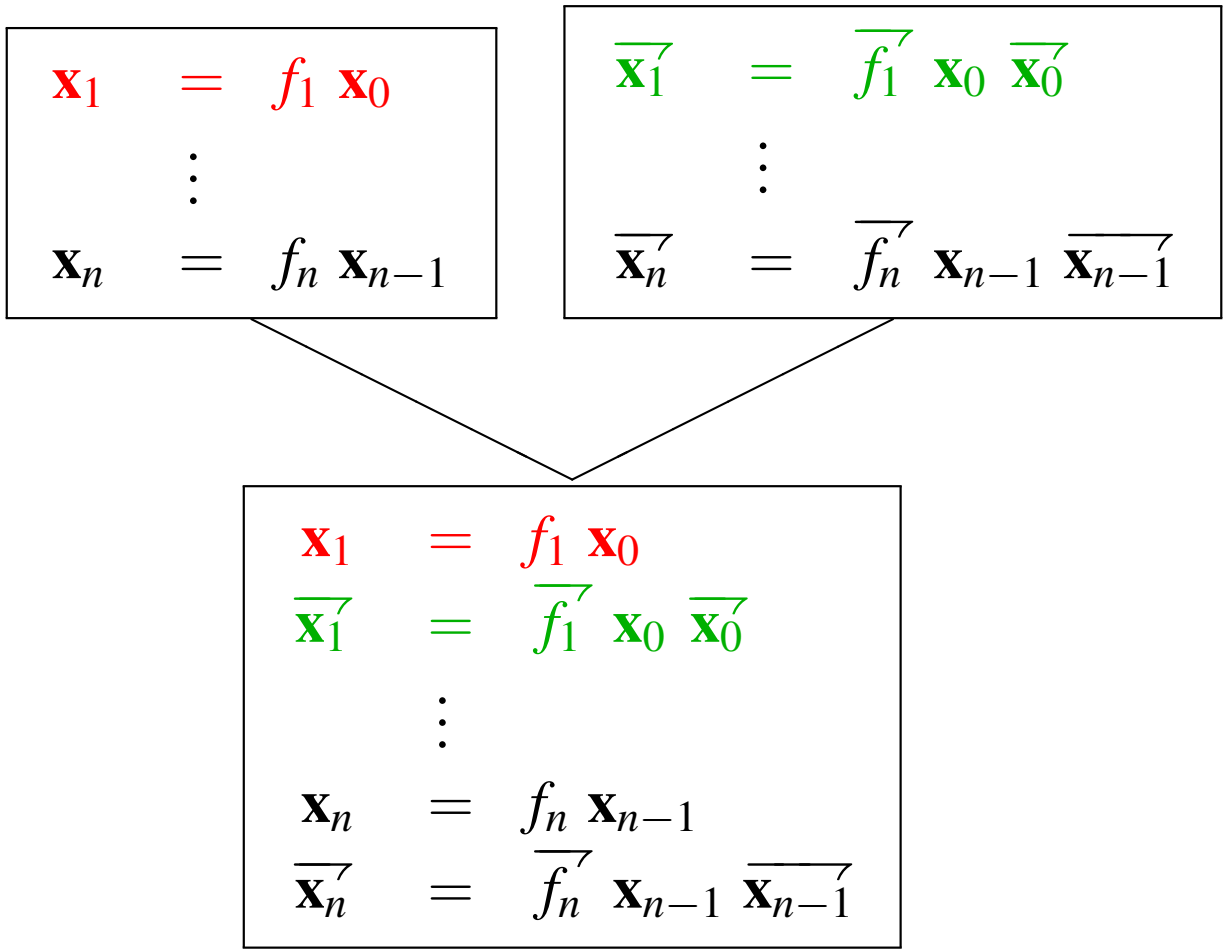
$$\begin{aligned} b'_1 &= \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \\ b'_2 &= \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \end{aligned}$$

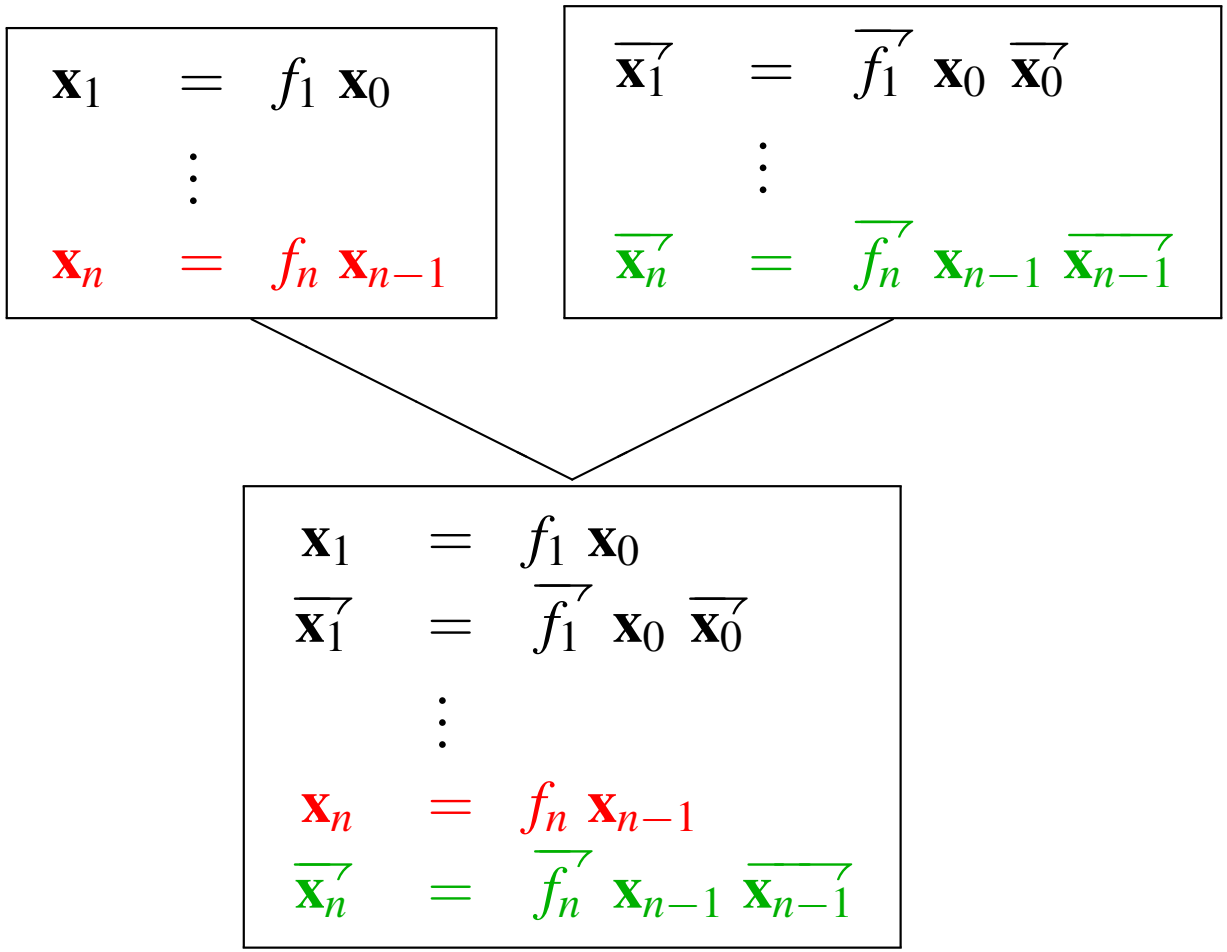
$$\begin{aligned}\overleftarrow{\mathbf{x}_{i-1}} &= \overleftarrow{f_i}\ \mathbf{x}_{i-1}\ \overleftarrow{\mathbf{x}_i} \\ &= (\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \overleftarrow{\mathbf{x}_i}\end{aligned}$$

$$\begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ 0 \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ b'_1 \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ b'_2 \times \overleftarrow{\mathbf{x}_i}[L_i] + \overleftarrow{\mathbf{x}_i}[S_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 0 & & & & & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & b'_1 & & & 1 & & & \\ & & & & & & & \ddots & & \\ & & & b'_2 & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \overleftarrow{\mathbf{x}_i}[1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[L_i - 1] \\ \overleftarrow{\mathbf{x}_i}[L_i] \\ \overleftarrow{\mathbf{x}_i}[L_i + 1] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[R_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[S_i] \\ \vdots \\ \overleftarrow{\mathbf{x}_i}[m] \end{pmatrix}$$

$$\begin{aligned} b'_1 &= \mathcal{D}_1\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \\ b'_2 &= \mathcal{D}_2\ b_i\ (\mathbf{x}_{i-1}[R_i], \mathbf{x}_{i-1}[S_i]) \end{aligned}$$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_1} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_0}$
$\overline{\mathbf{x}_n} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\overline{\mathbf{x}_1} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_0}$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_n} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}}$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_1} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_0}$
$\overline{\mathbf{x}_n} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\overline{\mathbf{x}_1} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_0}$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_n} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}}$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_1} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_0}$
$\overline{\mathbf{x}_n} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\overline{\mathbf{x}_1} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_0}$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_n} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}}$

$$
\begin{array}{rcl}
\mathbf{x}_1 & = & f_1\ \mathbf{x}_0 \\
\overline{\mathbf{x}_1'} & = & \overline{f_1'}\ \mathbf{x}_0\ \overline{\mathbf{x}_0'} \\
 & \vdots & \\
\mathbf{x}_n & = & f_n\ \mathbf{x}_{n-1} \\
\overline{\mathbf{x}_n'} & = & \overline{f_n'}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}'}
\end{array}
$$

$$
\begin{array}{rcl}
(\mathbf{x}_1, \overline{\mathbf{x}_1'}) & = & ((f_1\ \mathbf{x}_0), (\overline{f_1'}\ \mathbf{x}_0\ \overline{\mathbf{x}_0'})) \\
 & \vdots & \\
(\mathbf{x}_n, \overline{\mathbf{x}_n'}) & = & ((f_n\ \mathbf{x}_{n-1}), (\overline{f_n'}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_{n-1}'}))
\end{array}
$$

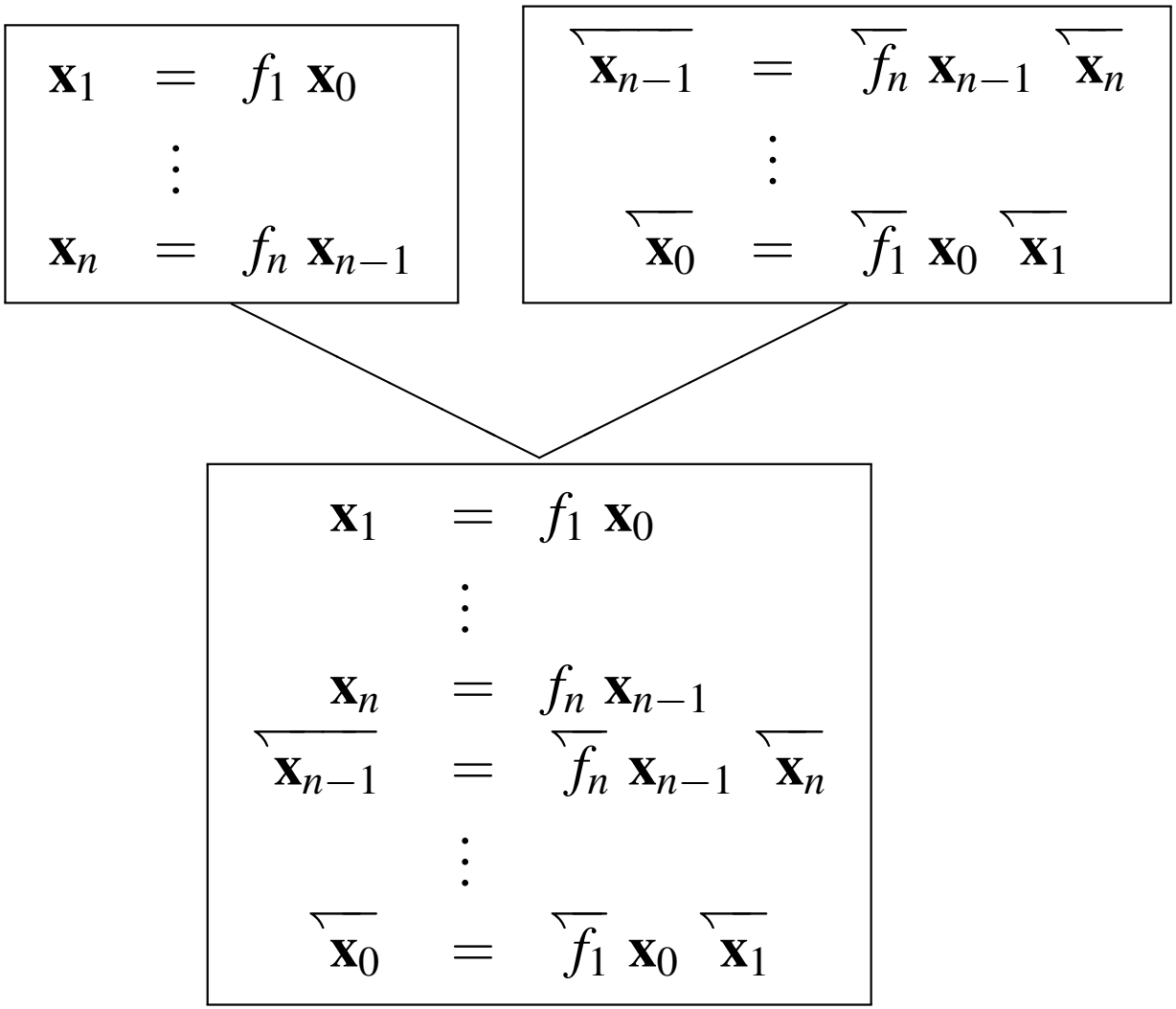
$$\left.\begin{array}{l} \mathbf{x}_1 = f_1\ \mathbf{x}_0 \\ \quad\vdots \\ \mathbf{x}_n = f_n\ \mathbf{x}_{n-1} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} \overrightarrow{\mathbf{x}_1} = \overrightarrow{f_1}\ \overrightarrow{\mathbf{x}_0} \\ \quad\vdots \\ \overrightarrow{\mathbf{x}_n} = \overrightarrow{f_n}\ \overrightarrow{\mathbf{x}_{n-1}} \end{array}\right.$$

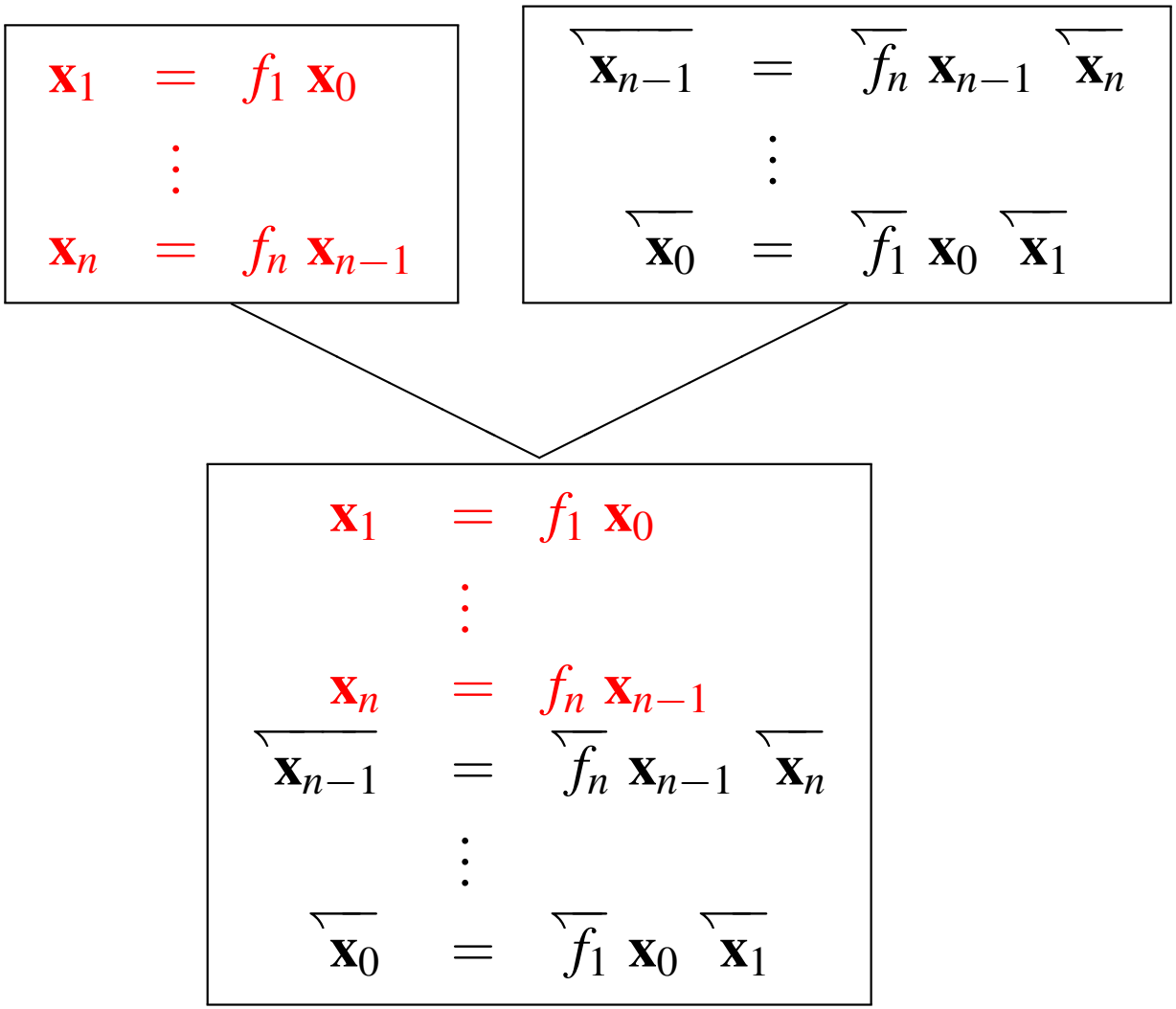
$$\overrightarrow{\mathbf{x}} \equiv (\mathbf{x}, \overline{\mathbf{x}})$$
$$\overrightarrow{f}\ \overrightarrow{\mathbf{x}} \equiv ((f\ \mathbf{x}), (\overline{f}\ \mathbf{x}\ \overline{\mathbf{x}}))$$

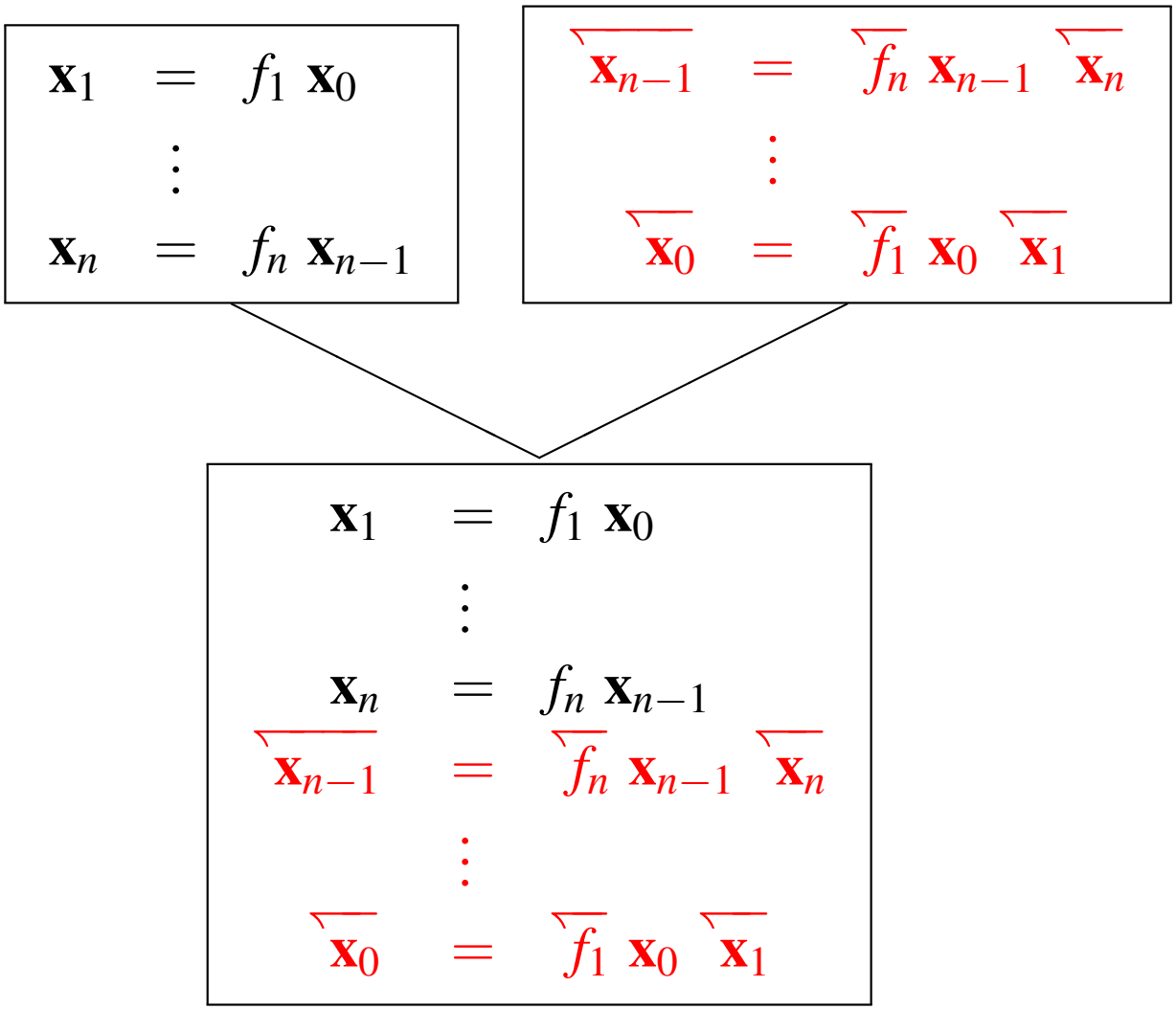
$$x_{L_i} := u_i\ x_{R_i} \quad \rightsquigarrow \quad \overrightarrow{x_{L_i}} := \overrightarrow{u_i}\ \overrightarrow{x_{R_i}}$$
$$x_{L_i} := b_i\ (x_{R_i}, x_{S_i}) \quad \rightsquigarrow \quad \overrightarrow{x_{L_i}} := \overrightarrow{b_i}\ (\overrightarrow{x_{R_i}}, \overrightarrow{x_{S_i}})$$

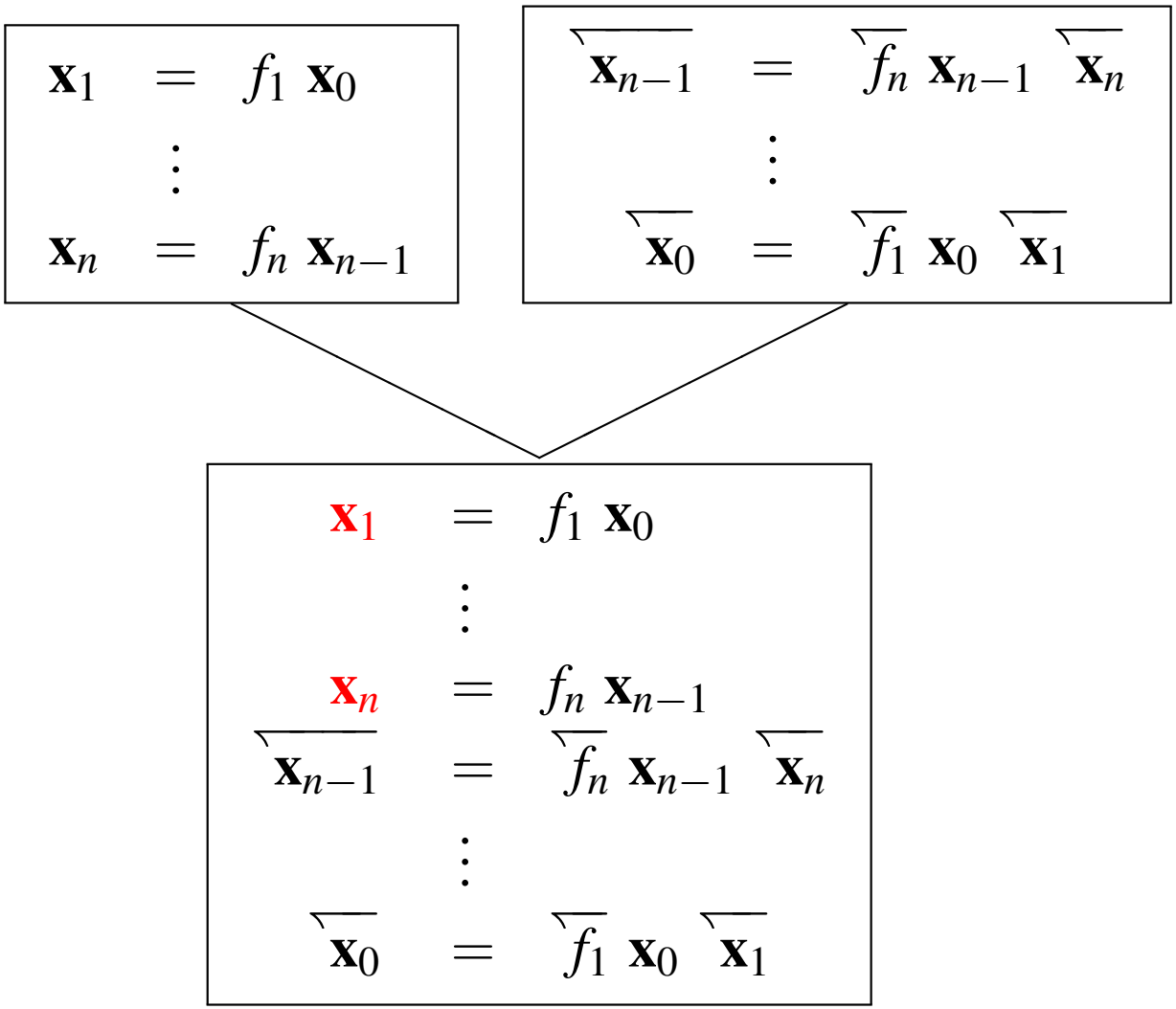
$$\overrightarrow{x} \equiv (x,\ \overline{x'})$$

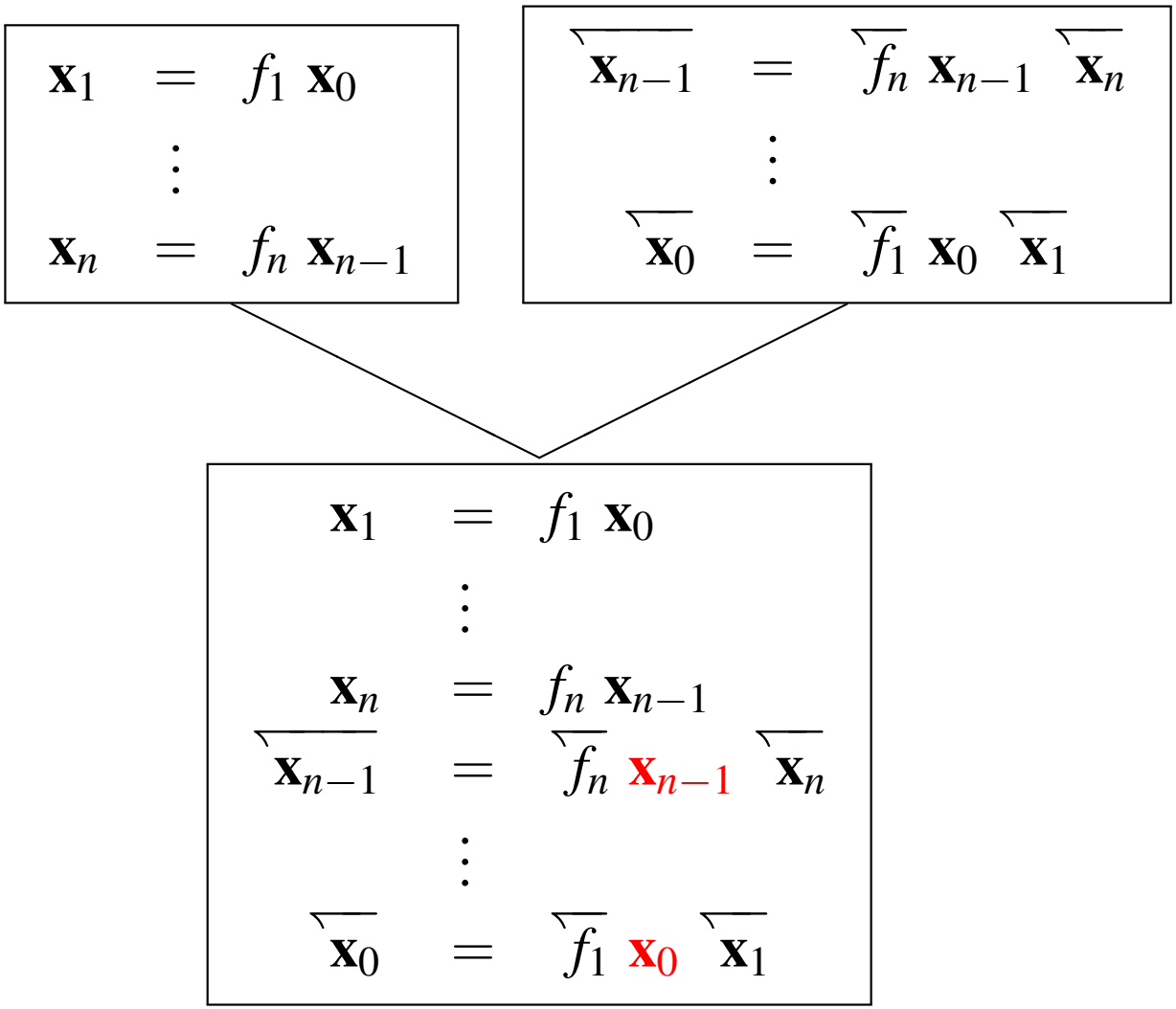
$$\overrightarrow{u}\ \overrightarrow{x} \equiv ((u\ x), ((\mathcal{D}\ u\ x) \times \overline{x'}))$$
$$\overrightarrow{b}\ (\overrightarrow{x_1}, \overrightarrow{x_2}) \equiv ((b\ (x_1, x_2)), ((\mathcal{D}_1\ b\ (x_1, x_2)) \times \overline{x_1'}) + ((\mathcal{D}_2\ b\ (x_1, x_2)) \times \overline{x_2'}))$$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$


$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$
$\mathbf{x}_1 = f_1\ \mathbf{x}_0$
$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$
$\overline{\mathbf{x}_{n-1}} = \overline{f_n}\ \mathbf{x}_{n-1}\ \overline{\mathbf{x}_n}$
$\overline{\mathbf{x}_0} = \overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}_1}$

$$
\begin{aligned}
\mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\
\overline{\mathbf{x}_1} &= \lambda\ \overline{\mathbf{x}}\quad \overline{\mathbf{x}_0}\ (\overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}}) \\
&\vdots \\
\mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \\
\overline{\mathbf{x}_n} &= \lambda\ \overline{\mathbf{x}}\quad \overline{\mathbf{x}_{n-1}}\ (\overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}})
\end{aligned}
$$

$$
\begin{array}{lcl}
\mathbf{x}_1 & = & f_1\ \mathbf{x}_0 \\
\overline{\mathbf{x}_1} & = & \lambda\,\overline{\mathbf{x}}\ \ \overline{\mathbf{x}_0}\ (\overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}}) \\
 & \vdots & \\
\mathbf{x}_n & = & f_n\ \mathbf{x}_{n-1} \\
\overline{\mathbf{x}_n} & = & \lambda\,\overline{\mathbf{x}}\ \ \overline{\mathbf{x}_{n-1}}\ (\overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}})
\end{array}
$$

$$
\begin{array}{lcl}
(\mathbf{x}_1, \overline{\mathbf{x}_1}) & = & ((f_1\ \mathbf{x}_0), (\lambda\,\overline{\mathbf{x}}\ \ \overline{\mathbf{x}_0}\ (\overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}}))) \\
 & \vdots & \\
(\mathbf{x}_n, \overline{\mathbf{x}_n}) & = & ((f_n\ \mathbf{x}_{n-1}), (\lambda\,\overline{\mathbf{x}}\ \ \overline{\mathbf{x}_{n-1}}\ (\overline{f_1}\ \mathbf{x}_0\ \overline{\mathbf{x}})))
\end{array}
$$

$$\left.\begin{array}{l} \mathbf{x}_1 = f_1\ \mathbf{x}_0 \\ \quad\vdots \\ \mathbf{x}_n = f_n\ \mathbf{x}_{n-1} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} \overleftarrow{\mathbf{x}_1} = \overleftarrow{f_1}\ \overleftarrow{\mathbf{x}_0} \\ \quad\vdots \\ \overleftarrow{\mathbf{x}_n} = \overleftarrow{f_n}\ \overleftarrow{\mathbf{x}_{n-1}} \end{array}\right.$$

$$\begin{aligned} \overleftarrow{\mathbf{x}} &\equiv (\mathbf{x}, \overline{\mathbf{x}}) \\ \overleftarrow{f}\ \overleftarrow{\mathbf{x}} &\equiv ((f\ \mathbf{x}), (\lambda\ \overline{\mathbf{x}}\ \ \overline{\mathbf{x}}\ (\overline{f}\ \mathbf{x}\ \overline{\mathbf{x}}))) \end{aligned}$$

$$\overleftarrow{f}\ \mathbf{x} \equiv \mathbf{begin}\ \bar{x} := \lambda\overline{\mathbf{x}}\quad \bar{x}\,(\overline{f}\ \mathbf{x}\ \overline{\mathbf{x}});\\ (f\ \mathbf{x})\ \mathbf{end}$$

$$x_{L_i} := u_i\ x_{R_i} \quad \rightsquigarrow \quad \begin{cases} \bar{x}\ := \lambda[\,]\ \textbf{begin}\ \overleftarrow{x_{R_i}} +:= (\mathcal{D}\ u_i\ \overleftarrow{x_{R_i}}) \times \overleftarrow{x_{L_i}}; \\ \qquad\qquad\qquad\qquad \overleftarrow{x_{L_i}} := 0; \\ \qquad\qquad\qquad\qquad \bar{x}\ [\,]\ \textbf{end} \\ \overleftarrow{x_{L_i}} := u_i\ \overleftarrow{x_{R_i}} \end{cases}$$

$$x_{L_i} := b_i\ (x_{R_i}, x_{S_i}) \quad \rightsquigarrow \quad \begin{cases} \bar{x}\ := \lambda[\,]\ \textbf{begin}\ \overleftarrow{x_{R_i}} +:= (\mathcal{D}_1\ b_i\ (\overleftarrow{x_{R_i}}, \overleftarrow{x_{S_i}})) \times \overleftarrow{x_{L_i}}; \\ \qquad\qquad\qquad\qquad \overleftarrow{x_{S_i}} +:= (\mathcal{D}_2\ b_i\ (\overleftarrow{x_{R_i}}, \overleftarrow{x_{S_i}})) \times \overleftarrow{x_{L_i}}; \\ \qquad\qquad\qquad\qquad \overleftarrow{x_{L_i}} := 0; \\ \qquad\qquad\qquad\qquad \bar{x}\ [\,]\ \textbf{end} \\ \overleftarrow{x_{L_i}} := b_i\ (\overleftarrow{x_{R_i}}, \overleftarrow{x_{S_i}}) \end{cases}$$

$$\overleftarrow{x} \equiv x$$

$$x_{L_i} := u_i\ x_{R_i} \quad \rightsquigarrow \quad \begin{cases} \overleftarrow{x_{L_i}} := u_i\ \overleftarrow{x_{R_i}} \\ \quad\vdots \\ \overleftarrow{x_{R_i}} +:= (\mathcal{D}\ u_i\ \overleftarrow{x_{R_i}}) \times \overleftarrow{x_{L_i}}; \\ \overleftarrow{x_{L_i}} := 0 \end{cases}$$

$$x_{L_i} := b_i\ (x_{R_i}, x_{S_i}) \quad \rightsquigarrow \quad \begin{cases} \overleftarrow{x_{L_i}} := b_i\ (\overleftarrow{x_{R_i}}, \overleftarrow{x_{S_i}}) \\ \quad\vdots \\ \overleftarrow{x_{R_i}} +:= (\mathcal{D}_1\ b_i\ (\overleftarrow{x_{R_i}}, \overleftarrow{x_{S_i}})) \times \overleftarrow{x_{L_i}}; \\ \overleftarrow{x_{S_i}} +:= (\mathcal{D}_2\ b_i\ (\overleftarrow{x_{R_i}}, \overleftarrow{x_{S_i}})) \times \overleftarrow{x_{L_i}}; \\ \overleftarrow{x_{L_i}} := 0 \end{cases}$$

$$\overleftarrow{x} \equiv x$$

The Functional Reverse-Mode Transformation

$$
\begin{aligned}
x_{L_1} &:= u_1\ x_{S_1} \\
&\vdots \\
x_{L_n} &:= u_n\ x_{S_n}
\end{aligned}
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n}
\end{array}\right\} \rightsquigarrow
\left\{\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n} \\
\overline{x_1} & := & 0 \\
 & \vdots & \\
\overline{x_m} & := & 0 \\
\overline{x_{S_n}} & +:= & (\mathcal{D}\ u_n\ x_{S_n}) \times \overline{x_{L_n}} \\
\overline{x_{L_n}} & := & 0 \\
 & \vdots & \\
\overline{x_{S_1}} & +:= & (\mathcal{D}\ u_1\ x_{S_1}) \times \overline{x_{L_1}} \\
\overline{x_{L_1}} & := & = 0
\end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n}
\end{array}\right\} \rightsquigarrow
\left\{\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n} \\
\overline{x_1} & := & 0 \\
 & \vdots & \\
\overline{x_m} & := & 0 \\
\overline{x_{S_n}} & +:= & (\mathcal{D}\ u_n\ x_{S_n}) \times \overline{x_{L_n}} \\
\overline{x_{L_n}} & := & 0 \\
 & \vdots & \\
\overline{x_{S_1}} & +:= & (\mathcal{D}\ u_1\ x_{S_1}) \times \overline{x_{L_1}} \\
\overline{x_{L_1}} & := & = 0
\end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n}
\end{array}\right\} \rightsquigarrow
\left\{\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n} \\
\overline{x_1} & := & 0 \\
 & \vdots & \\
\overline{x_m} & := & 0 \\
\overline{x_{S_n}} & +:= & (\mathcal{D}\ u_n\ x_{S_n}) \times \overline{x_{L_n}} \\
\overline{x_{L_n}} & := & 0 \\
 & \vdots & \\
\overline{x_{S_1}} & +:= & (\mathcal{D}\ u_1\ x_{S_1}) \times \overline{x_{L_1}} \\
\overline{x_{L_1}} & := & = 0
\end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n}
\end{array}\right\} \rightsquigarrow
\left\{\begin{array}{rcl}
x_{L_1} & := & u_1\ x_{S_1} \\
 & \vdots & \\
x_{L_n} & := & u_n\ x_{S_n} \\
\overline{x_1} & := & 0 \\
 & \vdots & \\
\overline{x_m} & := & 0 \\
\overline{x_{S_n}} & +:= & (\mathcal{D}\ u_n\ x_{S_n}) \times \overline{x_{L_n}} \\
\overline{x_{L_n}} & := & 0 \\
 & \vdots & \\
\overline{x_{S_1}} & +:= & (\mathcal{D}\ u_1\ x_{S_1}) \times \overline{x_{L_1}} \\
\overline{x_{L_1}} & := & = 0
\end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \\ \overline{x_0} & := & 0 \\ & \vdots & \\ \overline{x_{n-1}} & := & 0 \\ \overline{x_{S_n}} & +:= & (\mathcal{D}\ u_n\ x_{S_n}) \times \overline{x_n} \\ & \vdots & \\ \overline{x_{S_1}} & +:= & (\mathcal{D}\ u_1\ x_{S_1}) \times \overline{x_1} \end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \\ \overline{x_0} & := & 0 \\ & \vdots & \\ \overline{x_{n-1}} & := & 0 \\ \overline{x_{S_n}} & +:= & (\mathcal{D}\ u_n\ x_{S_n}) \times \overline{x_n} \\ & \vdots & \\ \overline{x_{S_1}} & +:= & (\mathcal{D}\ u_1\ x_{S_1}) \times \overline{x_1} \end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$\overline{u_i} \stackrel{\triangle}{=} \lambda \overline{x}\ (\mathcal{D}\ u_i\ x_{S_i}) \times \overline{x}$$

$$\left.\begin{array}{lcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \\ \overline{x_0} & := & 0 \\ & \vdots & \\ \overline{x_{n-1}} & := & 0 \\ \overline{x_{S_n}} & +:= & \overline{u_n}\ \overline{x_n} \\ & \vdots & \\ \overline{x_{S_1}} & +:= & \overline{u_1}\ \overline{x_1} \end{array}\right.$$

The Functional Reverse-Mode Transformation

$$\overleftarrow{u}\ x \triangleq ((u\ x), (\lambda \overleftarrow{x} (\mathcal{D}\ u\ x) \times \overleftarrow{x}))$$

$$\left.\begin{array}{lcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} (x_1, \overline{x_1}) & = & \overleftarrow{u_1}\ x_{S_1} \\ & \vdots & \\ (x_n, \overline{x_n}) & = & \overleftarrow{u_n}\ x_{S_n} \\ \overleftarrow{x_0} & := & 0 \\ & \vdots & \\ \overleftarrow{x_{n-1}} & := & 0 \\ \overleftarrow{x_{S_n}} & +:= & \overline{x_n}\ \overleftarrow{x_n} \\ & \vdots & \\ \overleftarrow{x_{S_1}} & +:= & \overline{x_1}\ \overleftarrow{x_1} \end{array}\right.$$

The Functional Reverse-Mode Transformation

$$\overleftarrow{u}\ x \stackrel{\triangle}{=} ((u\ x), (\lambda \overleftarrow{x} (\mathcal{D}\ u\ x) \times \overleftarrow{x}))$$

$$\left.\begin{array}{lcl} x_1 & = & u_1\ x_{S_1} \\ & \vdots & \\ x_n & = & u_n\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} (x_1, \overline{x_1}) & = & \overleftarrow{u_1}\ x_{S_1} \\ & \vdots & \\ (x_n, \overline{x_n}) & = & \overleftarrow{u_n}\ x_{S_n} \\ \overleftarrow{x_0} & := & 0 \\ & \vdots & \\ \overleftarrow{x_{n-1}} & := & 0 \\ \overleftarrow{x_{S_n}} & +:= & \overline{x_n}\ \overleftarrow{x_n} \\ & \vdots & \\ \overleftarrow{x_{S_1}} & +:= & \overline{x_1}\ \overleftarrow{x_1} \end{array}\right.$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl} x_1 & = & x_{R_1}\ x_{S_1} \\ & \vdots & \\ x_n & = & x_{R_n}\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} (\overleftarrow{x_1}, \overline{x_1}) & = & \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}} \\ & \vdots & \\ (\overleftarrow{x_n}, \overline{x_n}) & = & \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\ \overleftarrow{x_0} & := & 0 \\ & \vdots & \\ \overleftarrow{x_{n-1}} & := & 0 \\ \overleftarrow{x_{S_n}} & +:= & \overline{x_n}\ \overleftarrow{x_n} \\ & \vdots & \\ \overleftarrow{x_{S_1}} & +:= & \overline{x_1}\ \overleftarrow{x_1} \end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{lcl} x_1 & = & x_{R_1}\ x_{S_1} \\ & \vdots & \\ x_n & = & x_{R_n}\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} (\overleftarrow{x_1}, \overline{x_1}) & = & \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}} \\ & \vdots & \\ (\overleftarrow{x_n}, \overline{x_n}) & = & \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\ \overleftarrow{x_0} & := & 0 \\ & \vdots & \\ \overleftarrow{x_{n-1}} & := & 0 \\ \overleftarrow{x_{S_n}} & +:= & \overline{x_n}\ \overleftarrow{x_n} \\ & \vdots & \\ \overleftarrow{x_{S_1}} & +:= & \overline{x_1}\ \overleftarrow{x_1} \end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl} x_1 & = & x_{R_1}\ x_{S_1} \\ & \vdots & \\ x_n & = & x_{R_n}\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} (\overleftarrow{x_1}, \overleftarrow{x_1}) & = & \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}} \\ & \vdots & \\ (\overleftarrow{x_n}, \overleftarrow{x_n}) & = & \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\ \overleftarrow{x_0} & := & 0 \\ & \vdots & \\ \overleftarrow{x_{n-1}} & := & 0 \\ \overleftarrow{x_{S_n}} & +:= & \overleftarrow{x_n}\ \overleftarrow{x_n} \\ & \vdots & \\ \overleftarrow{x_{S_1}} & +:= & \overleftarrow{x_1}\ \overleftarrow{x_1} \end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl} x_1 & = & x_{R_1}\ x_{S_1} \\ & \vdots & \\ x_n & = & x_{R_n}\ x_{S_n} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{rcl} (\overleftarrow{x_1}, \overline{x_1}) & = & \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}} \\ & \vdots & \\ (\overleftarrow{x_n}, \overline{x_n}) & = & \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\ \overleftarrow{x_0} & := & 0 \\ & \vdots & \\ \overleftarrow{x_{n-1}} & := & 0 \\ \overleftarrow{x_{S_n}} & +:= & \overline{x_n}\ \overleftarrow{x_n} \\ & \vdots & \\ \overleftarrow{x_{S_1}} & +:= & \overline{x_1}\ \overleftarrow{x_1} \end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{rcl}
x_1 & = & x_{R_1}\ x_{S_1} \\
 & \vdots & \\
x_n & = & x_{R_n}\ x_{S_n}
\end{array}\right\} \rightsquigarrow
\left\{\begin{array}{rcl}
(\overleftarrow{x_1}, \overleftarrow{x_1}) & = & \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}} \\
 & \vdots & \\
(\overleftarrow{x_n}, \overleftarrow{x_n}) & = & \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\
\overleftarrow{x_0} & := & \mathbf{0}\ (\overleftarrow{\mathcal{J}}^{-1}\ \overleftarrow{x_0}) \\
 & \vdots & \\
\overleftarrow{x_{n-1}} & := & \mathbf{0}\ (\overleftarrow{\mathcal{J}}^{-1}\ \overleftarrow{x_{n-1}}) \\
\overleftarrow{x_{S_n}} & \oplus := & \overleftarrow{x_n}\ \overleftarrow{x_n} \\
 & \vdots & \\
\overleftarrow{x_{S_1}} & \oplus := & \overleftarrow{x_1}\ \overleftarrow{x_1}
\end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{l}
\lambda x_0\ \textbf{let}\ x_1 \stackrel{\triangle}{=} x_{R_1}\ x_{S_1}; \\
\qquad\vdots \\
\qquad x_n \stackrel{\triangle}{=} x_{R_n}\ x_{S_n} \\
\textbf{in}\ x_n\ \textbf{end}
\end{array}\right\} \rightsquigarrow \left\{\begin{array}{l}
\lambda \overleftarrow{x_0}\ \textbf{let}\ (\overleftarrow{x_1}, \overline{x_1}) \stackrel{\triangle}{=} \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}}; \\
\qquad\qquad\vdots \\
\qquad\qquad (\overleftarrow{x_n}, \overline{x_n}) \stackrel{\triangle}{=} \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\
\qquad \textbf{in}\ (\overleftarrow{x_n}, (\lambda \overline{x_n}\ \textbf{let}\ \overline{x_0} \stackrel{\triangle}{=} \mathbf{0}\ (\overleftarrow{\mathcal{J}}^{-1}\ \overleftarrow{x_0}); \\
\qquad\qquad\qquad\vdots \\
\qquad\qquad\qquad \overline{x_{n-1}} \stackrel{\triangle}{=} \mathbf{0}\ (\overleftarrow{\mathcal{J}}^{-1}\ \overleftarrow{x_{n-1}}); \\
\qquad\qquad\qquad \overline{x_{S_n}} \oplus\!\!= \stackrel{\triangle}{=} \overline{x_n}\ \overline{x_n}; \\
\qquad\qquad\qquad\vdots \\
\qquad\qquad\qquad \overline{x_{S_1}} \oplus\!\!= \stackrel{\triangle}{=} \overline{x_1}\ \overline{x_1} \\
\qquad\qquad \textbf{in}\ \overline{x_0}\ \textbf{end}))\ \textbf{end}
\end{array}\right.
$$

The Functional Reverse-Mode Transformation

$$
\left.\begin{array}{l}
\lambda x_0\ \textbf{let}\ x_1 \stackrel{\triangle}{=} x_{R_1}\ x_{S_1}; \\
\qquad\vdots \\
\qquad x_n \stackrel{\triangle}{=} x_{R_n}\ x_{S_n} \\
\textbf{in}\ x_n\ \textbf{end}
\end{array}\right\} \rightsquigarrow
$$

$$
\left\{\begin{array}{l}
\lambda \overleftarrow{x_0}\ \textbf{let}\ (\overleftarrow{x_1}, \overleftarrow{x_1}) \stackrel{\triangle}{=} \overleftarrow{x_{R_1}}\ \overleftarrow{x_{S_1}}; \\
\qquad\qquad\vdots \\
\qquad\qquad (\overleftarrow{x_n}, \overleftarrow{x_n}) \stackrel{\triangle}{=} \overleftarrow{x_{R_n}}\ \overleftarrow{x_{S_n}} \\
\qquad \textbf{in}\ (\overleftarrow{x_n}, (\lambda \overleftarrow{x_n}\ \textbf{let}\ \overleftarrow{x_0} \stackrel{\triangle}{=} \mathbf{0}\ (\overleftarrow{\mathcal{T}}^{-1}\ \overleftarrow{x_0}); \\
\qquad\qquad\qquad\vdots \\
\qquad\qquad\qquad \overleftarrow{x_{n-1}} \stackrel{\triangle}{=} \mathbf{0}\ (\overleftarrow{\mathcal{T}}^{-1}\ \overleftarrow{x_{n-1}}); \\
\qquad\qquad\qquad \overleftarrow{x_{S_n}} \oplus\!\!\stackrel{\triangle}{=} \overleftarrow{x_n}\ \overleftarrow{x_n}; \\
\qquad\qquad\qquad\vdots \\
\qquad\qquad\qquad \overleftarrow{x_{S_1}} \oplus\!\!\stackrel{\triangle}{=} \overleftarrow{x_1}\ \overleftarrow{x_1} \\
\qquad\qquad \textbf{in}\ \overleftarrow{x_0}\ \textbf{end}))\ \textbf{end}
\end{array}\right.
$$

The above is a white lie. The truth is a lot more complicated.

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))
```

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))
```

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
```

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
(𝒟 (𝒟 f))
(𝒟 (lambda (x) ... (𝒟 (lambda (y) ...) ...) ...) ...)
```

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
(𝒟 (𝒟 f))
(𝒟 (lambda (x) ... (𝒟 (lambda (y) ...) ...) ...) ...)
```

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
(𝒟 (𝒟 f))
(𝒟 (lambda (x) ... (𝒟 (lambda (y) ...) ...) ...) ...)
```

Convenient

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
(𝒟 (𝒟 f))
(𝒟 (lambda (x) ... (𝒟 (lambda (y) ...) ...) ...) ...)
```

Convenient but slow

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
(𝒟 (𝒟 f))
(𝒟 (lambda (x) ... (𝒟 (lambda (y) ...) ...) ...) ...)
```

Convenient but slow

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define +
 (let ((+ +))
  (lambda (x1 x2)
   (make-bundle (+ (primal x1) (primal x2))
                (+ (tangent x1) (tangent x2))))))

(define *
 (let ((+ +) (* *))
  (lambda (x1 x2)
   (make-bundle (* (primal x1) (primal x2))
                (+ (* (primal x1) (tangent x2))
                   (* (tangent x1) (primal x2)))))))

(define ((D f) x) (tangent (f (make-bundle x 1))))

(define (f x) (* 2 (* x (* x x))))

(𝒟 f)
(𝒟 (𝒟 f))
(𝒟 (lambda (x) ... (𝒟 (lambda (y) ...) ...) ...) ...)
```

Convenient but slow

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define ((D f) x)
 (fluid-let ((+ (lambda (x1 x2)
                  (make-bundle (+ (primal x1) (primal x2))
                               (+ (tangent x1) (tangent x2)))))
             (* (lambda (x1 x2)
                  (make-bundle (* (primal x1) (primal x2))
                               (+ (* (primal x1) (tangent x2))
                                  (* (tangent x1) (primal x2)))))))
  (tangent (f (make-bundle x 1)))))
```

```
(define (f x) (* 2 (* x (* x x))))

(D f)
(D (D f))
(D (lambda (x) ... (D (lambda (y) ...) ...) ...) ...)
```

Convenient but slow

Dynamic Overloading: SCMUTILS

```
(define-structure bundle primal tangent)
(define (primal p) (if (bundle? p) (bundle-primal p) p))
(define (tangent p) (if (bundle? p) (bundle-tangent p) 0))

(define ((D f) x)
 (fluid-let ((+ (lambda (x1 x2)
                  (make-bundle (+ (primal x1) (primal x2))
                               (+ (tangent x1) (tangent x2)))))
             (* (lambda (x1 x2)
                  (make-bundle (* (primal x1) (primal x2))
                               (+ (* (primal x1) (tangent x2))
                                  (* (tangent x1) (primal x2)))))))
  (tangent (f (make-bundle x 1)))))
```

```
(define (f x) (* 2 (* x (* x x))))

(D f)
(D (D f))
(D (lambda (x) ... (D (lambda (y) ...) ...) ...) ...)
```

Convenient but slow

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end
```

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

Fast

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

```
AD_TOP = f
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

```
AD_TOP = f
AD_IVARS = x
AD_DVARS = f
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

```
AD_TOP = f
AD_IVARS = x
AD_DVARS = f
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end

function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

```
AD_TOP = f
AD_IVARS = x
AD_DVARS = f
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)
double precision x, f
f = 2.0d0*x*x*x
end
```

```
AD_TOP = f
AD_IVARS = x
AD_DVARS = f
```

```
function gf(x, gx, gresult)
double precision x, gx, gf, gresult
gf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
end
```

```
AD_TOP = gf
AD_IVARS = x, gx
AD_DVARS = gf, gresult
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)                                    AD_TOP = f
double precision x, f                            AD_IVARS = x
f = 2.0d0*x*x*x                                  AD_DVARS = f
end

function gf(x, gx, gresult)                      AD_TOP = gf
double precision x, gx, gf, gresult              AD_IVARS = x, gx
gf = 2.0d0*x*x*x                                 AD_DVARS = gf, gresult
gresult = 6.0d0*x*x*gx
end

function ggf(x, gx, gx, ggx, gresult, ggresult, gresult)
double precision x, gx, gx, ggx, ggf, gresult, gresult, ggresult
ggf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
gresult = 6.0d0*x*x*gx
ggresult = 6.0d0*x*x*ggx+12.0d0*x*gx*gx
end
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)                                    AD_TOP = f
double precision x, f                            AD_IVARS = x
f = 2.0d0*x*x*x                                  AD_DVARS = f
end

function gf(x, gx, gresult)                      AD_TOP = gf
double precision x, gx, gf, gresult              AD_IVARS = x, gx
gf = 2.0d0*x*x*x                                 AD_DVARS = gf, gresult
gresult = 6.0d0*x*x*gx
end

function ggf(x, gx, gx, ggx, gresult, ggresult, gresult)
double precision x, gx, gx, ggx, ggf, gresult, gresult, ggresult
ggf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
gresult = 6.0d0*x*x*gx
ggresult = 6.0d0*x*x*ggx+12.0d0*x*gx*gx
end
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)                                    AD_TOP = f
double precision x, f                            AD_IVARS = x
f = 2.0d0*x*x*x                                  AD_DVARS = f
end

function gf(x, gx, gresult)                      AD_TOP = gf
double precision x, gx, gf, gresult              AD_IVARS = x, gx
gf = 2.0d0*x*x*x                                 AD_DVARS = gf, gresult
gresult = 6.0d0*x*x*gx
end

function ggf(x, gx, gx, ggx, gresult, ggresult, gresult)
double precision x, gx, gx, ggx, ggf, gresult, gresult, ggresult
ggf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
gresult = 6.0d0*x*x*gx
ggresult = 6.0d0*x*x*ggx+12.0d0*x*gx*gx
end
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)                                    AD_TOP = f
double precision x, f                            AD_IVARS = x
f = 2.0d0*x*x*x                                  AD_DVARS = f
end

function gf(x, gx, gresult)                      AD_TOP = gf
double precision x, gx, gf, gresult              AD_IVARS = x, gx
gf = 2.0d0*x*x*x                                 AD_DVARS = gf, gresult
gresult = 6.0d0*x*x*gx
end

function ggf(x, gx, gx, ggx, gresult, ggresult, gresult)
double precision x, gx, gx, ggx, ggf, gresult, gresult, ggresult
ggf = 2.0d0*x*x*x
gresult = 6.0d0*x*x*gx
gresult = 6.0d0*x*x*gx
ggresult = 6.0d0*x*x*ggx+12.0d0*x*gx*gx
end
```

Fast but inconvenient

Preprocessor: ADIFOR and TAPENADE

```
function f(x)                                    AD_TOP = f
double precision x, f                            AD_IVARS = x
f = 2.0d0*x*x*x                                  AD_DVARS = f
end

function gf(x, gx, gresult)                      AD_TOP = gf
double precision x, gx, gf, gresult              AD_IVARS = x, gx
gf = 2.0d0*x*x*x                                 AD_DVARS = gf, gresult
gresult = 6.0d0*x*x*gx                           AD_PREFIX = h
end

function hgf(x, hx, gx, hgx, gresult, hgresult, hresult)
double precision x, hx, gx, hgx, hgf, hresult, gresult, hgresult
hgf = 2.0d0*x*x*x
hresult = 6.0d0*x*x*hx
gresult = 6.0d0*x*x*gx
hgresult = 6.0d0*x*x*hgx+12.0d0*x*gx*hx
end
```

Fast but inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...
```

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...
```

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...

template <typename T>
T f(T x) {return 2*x*x*x;}
T x;
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...

template <typename T>
T f(T x) {return 2*x*x*x;}
T x;
```

Slow and inconvenient

Static Overloading: FADBAD++

```
double f(double x) {return 2*x*x*x;}
double x;
... f(x) ...

F<double> f(F<double> x) {return 2*x*x*x;}
F<double> x;
x.diff(0, 1);
... f(x).d(0) ...

F<F<double> > f(F<F<double> > x) {return 2*x*x*x;}
F<F<double> > x;
x.diff(0, 1);
x.diff(0, 1).diff(0,1);
... f(x).d(0).d(0) ...

template <typename T>
T f(T x) {return 2*x*x*x;}
T x;
```

Slow and inconvenient

Monovariant Flow Analysis: 0-CFA

```
(define (𝒟 f)
 ...)
```

Monovariant Flow Analysis: 0-CFA

```
(define (𝒟 f)
 ...)

(𝒟 (lambda (x) 2x^3))
```

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:$(\lambda x\ 2x^3)$)
 ...)

($\mathcal{D}$ (lambda (x) $2x^3$))

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:$(\lambda x\ 2x^3)$)
 ...:$(\lambda x\ 6x^2)$)

($\mathcal{D}$ (lambda (x) $2x^3$))

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:$(\lambda x\ 2x^3)$)
 ...:$(\lambda x\ 6x^2)$)

($\mathcal{D}$ (lambda (x) $2x^3$)):$(\lambda x\ 6x^2)$

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:$(\lambda x\ 2x^3)$)
 ...:$(\lambda x\ 6x^2)$)

($\mathcal{D}$ (lambda (x) $2x^3$)):$(\lambda x\ 6x^2)$

($\mathcal{D}$ (lambda (x) $3x^4$))

Monovariant Flow Analysis: 0-CFA

`(define (`$\mathcal{D}$ `f:`$(\lambda x\ 2x^3) \cup (\lambda x\ 3x^4)$`)`
`...:`$(\lambda x\ 6x^2)$`)`

`(`$\mathcal{D}$ `(lambda (x)` $2x^3$`)):`$(\lambda x\ 6x^2)$

`(`$\mathcal{D}$ `(lambda (x)` $3x^4$`))`

Monovariant Flow Analysis: 0-CFA

`(define (`$\mathcal{D}$ `f:`$(\lambda x\ 2x^3) \cup (\lambda x\ 3x^4)$`)`
`...:`$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$`)`

`(`$\mathcal{D}$ `(lambda (x)` $2x^3$`)):`$(\lambda x\ 6x^2)$

`(`$\mathcal{D}$ `(lambda (x)` $3x^4$`))`

Monovariant Flow Analysis: 0-CFA

(define (f : $(\lambda x\ 2x^3) \cup (\lambda x\ 3x^4)$)
  ... : $(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$)

(lambda (x) $2x^3$) : $(\lambda x\ 6x^2)$

(lambda (x) $3x^4$) : $(\lambda x\ 12x^3)$

Monovariant Flow Analysis: 0-CFA

`(define (`$\mathcal{D}$ `f:`$(\lambda x\ 2x^3) \cup (\lambda x\ 3x^4)$`)`
`...:`$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$`)`

`(`$\mathcal{D}$ `(lambda (x)` $2x^3$`)):`$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$

`(`$\mathcal{D}$ `(lambda (x)` $3x^4$`)):`$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$

Monovariant Flow Analysis: 0-CFA

(define (𝒟 f:$(\lambda x\ 2x^3) \cup (\lambda x\ 3x^4)$)
 ...:$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$)

(𝒟 (lambda (x) $2x^3$)):$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$

(𝒟 (lambda (x) $3x^4$)):$(\lambda x\ 6x^2) \cup (\lambda x\ 12x^3)$

Monovariant Flow Analysis: 0-CFA

```
(define (𝒟 f)
 ...)
```

Monovariant Flow Analysis: 0-CFA

```
(define (D f)
 ...)
```

$(\mathcal{D}\ (\mathcal{D}\ (\texttt{lambda}\ (\texttt{x})\ e^{2x})))$

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:($\lambda x\, e^{2x}$))
 ...)

($\mathcal{D}$ ($\mathcal{D}$ (lambda (x) e^{2x})))

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:$(\lambda x\, e^{2x})$)
 ...:$(\lambda x\, 2e^{2x})$)

($\mathcal{D}$ ($\mathcal{D}$ (lambda (x) e^{2x})))

Monovariant Flow Analysis: 0-CFA

(define ($\mathcal{D}$ f:$(\lambda x\ e^{2x})$)
 ...:$(\lambda x\ 2e^{2x})$)

($\mathcal{D}$ ($\mathcal{D}$ (lambda (x) e^{2x})):$(\lambda x\ 2e^{2x})$)

Monovariant Flow Analysis: 0-CFA

(define (D f:$(\lambda x\ e^{2x}) \cup (\lambda x\ 2e^{2x})$)
 ...:$(\lambda x\ 2e^{2x})$)

(D (D (lambda (x) e^{2x})):$(\lambda x\ 2e^{2x})$)

Monovariant Flow Analysis: 0-CFA

(define (D f:$(\lambda x\ e^{2x}) \cup (\lambda x\ 2e^{2x})$)
 ...:$(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x})$)

(D (D (lambda (x) e^{2x})):$(\lambda x\ 2e^{2x})$)

Monovariant Flow Analysis: 0-CFA

(define (D f:$(\lambda x\ e^{2x}) \cup (\lambda x\ 2e^{2x})$)
 ...:$(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x})$)

(D (D (lambda (x) e^{2x})):$(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x})$)

Monovariant Flow Analysis: 0-CFA

(define (D f : $(\lambda x\ e^{2x}) \cup (\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x})$)
 ... : $(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x})$)

(D (D (lambda (x) e^{2x})) : $(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x})$)

Monovariant Flow Analysis: 0-CFA

$(\texttt{define}\ \ (\mathcal{D}\ \ \texttt{f}\texttt{:}(\lambda x\ e^{2x}) \cup (\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x}) \cup \ldots)$
$\ \ \ldots\texttt{:}(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x}) \cup \ldots)$

$(\mathcal{D}\ \ (\mathcal{D}\ \ (\texttt{lambda}\ \ (\texttt{x})\ \ e^{2x}))\texttt{:}(\lambda x\ 2e^{2x}) \cup (\lambda x\ 4e^{2x}) \cup \ldots)$

```
(define (𝒟 f) ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define (D f) ...)
```

```
(define (g ...) ... (D (lambda (x) 2x^3)) ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

`(define (`$\mathcal{D}_{\texttt{g}}$ `f) ...)`

`(define (g ...) ... (`$\mathcal{D}$ `(lambda (x)` $2x^3$`)) ...)`

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

`(define (`$\mathcal{D}_{\texttt{g}}$ `f:`$(\lambda x\ 2x^3)$`) ...)`

`(define (g ...) ... (`$\mathcal{D}$ `(lambda (x)` $2x^3$`)) ...)`

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

`(define (`$\mathcal{D}_{\texttt{g}}$ `f:`$(\lambda x\ 2x^3)$`)) ...:`$(\lambda x\ 6x^2)$`)`

`(define (g ...) ... (`$\mathcal{D}$ `(lambda (x)` $2x^3$`)) ...)`

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

`(define (`$\mathcal{D}_{\texttt{g}}$ `f:`$(\lambda x\ 2x^3)$`)) ...:`$(\lambda x\ 6x^2)$`)`

`(define (g ...) ... (`$\mathcal{D}$ `(lambda (x)` $2x^3$`)):`$(\lambda x\ 6x^2)$ `...)`

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

`(define (`$\mathcal{D}_{\texttt{g}}$ `f:`$(\lambda x\ 2x^3)$`)) ...:`$(\lambda x\ 6x^2)$`)`

`(define (g ...) ... (`$\mathcal{D}$ `(lambda (x)` $2x^3$`)):`$(\lambda x\ 6x^2)$ `...)`

`(define (h ...) ... (`$\mathcal{D}$ `(lambda (x)` $3x^4$`)) ...)`

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define (D_g f:(λx 2x^3)) ...:(λx 6x^2))

(define (D_h f) ...)

(define (g ...) ... (D (lambda (x) 2x^3)):(λx 6x^2) ...)

(define (h ...) ... (D (lambda (x) 3x^4)) ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages **or** Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define (D_g f:(λx 2x^3)) ...:(λx 6x^2))

(define (D_h f:(λx 3x^4)) ...)

(define (g ...) ... (D (lambda (x) 2x^3)):(λx 6x^2) ...)

(define (h ...) ... (D (lambda (x) 3x^4)) ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

(define ($\mathcal{D}_{\mathtt{g}}$ f:$(\lambda x\ 2x^3)$) ...:$(\lambda x\ 6x^2)$)

(define ($\mathcal{D}_{\mathtt{h}}$ f:$(\lambda x\ 3x^4)$) ...:$(\lambda x\ 12x^3)$)

(define (g ...) ... ($\mathcal{D}$ (lambda (x) $2x^3$)):$(\lambda x\ 6x^2)$ ...)

(define (h ...) ... ($\mathcal{D}$ (lambda (x) $3x^4$)) ...)

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

`(define (`$\mathcal{D}_{\texttt{g}}$ `f:`$(\lambda x\ 2x^3)$`) ...:`$(\lambda x\ 6x^2)$`)`

`(define (`$\mathcal{D}_{\texttt{h}}$ `f:`$(\lambda x\ 3x^4)$`) ...:`$(\lambda x\ 12x^3)$`)`

`(define (g ...) ... (`$\mathcal{D}$ `(lambda (x)` $2x^3$`)):`$(\lambda x\ 6x^2)$ `...)`

`(define (h ...) ... (`$\mathcal{D}$ `(lambda (x)` $3x^4$`)):`$(\lambda x\ 12x^3)$ `...)`

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define ((compose n f) x)
 (if (zero? n) x ((compose (- n 1) f) (f x))))
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages **or** Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define ((compose n f) x)
 (if (zero? n) x ((compose (- n 1) f) (f x))))

((compose k 𝒟) g)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define ((compose n f) x)
 (if (zero? n) x ((compose (- n 1) f) (f x))))

((compose k D) g)

(define (D_compose f:g) ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define ((compose n f) x)
 (if (zero? n) x ((compose (- n 1) f) (f x))))
```

```
((compose k D) g)
```

```
(define (D_compose f:g) ...)
```

```
(define (D_compose:compose f:g') ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages **or** Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define ((compose n f) x)
 (if (zero? n) x ((compose (- n 1) f) (f x))))
```

```
((compose k D) g)
```

```
(define (D_compose f:g) ...)
```

```
(define (D_compose:compose f:g') ...)
```

```
(define (D_compose:compose:compose f:g'') ...)
```

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

Polyvariant Flow Analysis: k-CFA

with Bounded Context Sensitivity

```
(define ((compose n f) x)
 (if (zero? n) x ((compose (- n 1) f) (f x))))

((compose k D) g)

(define (D_compose f:g) ...)

(define (D_compose:compose f:g') ...)

(define (D_compose:compose:compose f:g'') ...)
```

⋮

Shivers, III, O. G. (1991). *Control-Flow Analysis of Higher-Order Languages* ***or*** *Taming Lambda*. Ph.D. thesis, CMU.

$$\mathcal{E} : e \times \sigma \to v$$

$$\mathcal{E} : e \times \sigma \to v$$

$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$

$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$

$$\mathcal{E} : e \times \sigma \to v$$

$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$

$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$

$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$

$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

$$\mathcal{E} : e \times \sigma \to v$$

$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$

$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$

$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$

$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$

$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.
*I*s suitable for FORTRAN-like programs.

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.
*I*s suitable for FORTRAN-like programs.
Necessary for migrating reflective source-code transformation to compile time.

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.
*I*s suitable for FORTRAN-like programs.
Necessary for migrating reflective source-code transformation to compile time.
Side benefit: union-free

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.
*I*s suitable for FORTRAN-like programs.
Necessary for migrating reflective source-code transformation to compile time.
Side benefit: union-free
No tags, tag checking, tag dispatching, indirect calls

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.
*I*s suitable for FORTRAN-like programs.
Necessary for migrating reflective source-code transformation to compile time.
Side benefits: union-free, no cyclic abstract values
No tags, tag checking, tag dispatching, indirect calls

$$\mathcal{E} : e \times \sigma \to v$$
$$v ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (v_1, v_2) \mid \langle \sigma, e \rangle$$
$$\sigma ::= \{x_1 \mapsto v_1, \ldots\}$$
$$\overline{\mathcal{E}} : e \times \overline{\sigma} \to \overline{v}$$
$$\overline{v} ::= \texttt{\#t} \mid \texttt{\#f} \mid \texttt{()} \mid \mathbb{R} \mid (\overline{v_1}, \overline{v_2}) \mid \langle \overline{\sigma}, e \rangle \mid \overline{\mathbb{R}}$$
$$\overline{\sigma} ::= \{x_1 \mapsto \overline{v_1}, \ldots\}$$

Memoize $\overline{\mathcal{E}}$ indexed (by suitable equivalence relations on) e and $\overline{\sigma}$.
Not suitable for arbitrary (i.e., typical SCHEME, ML, HASKELL, etc.) programs.
*I*s suitable for FORTRAN-like programs.
Necessary for migrating reflective source-code transformation to compile time.
Side benefits: union-free, no cyclic abstract values
No tags, tag checking, tag dispatching, indirect calls
Allows complete unboxing: no allocation, reclamation, indirection

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a,b)$.

Player B's strategy is b.
Player B's return is $B(a,b)$.

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a,b)$.

Player B's strategy is b.
Player B's return is $B(a,b)$.

Equilibria must satisfy:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a,b^*) \qquad b^* = \underset{b}{\operatorname{argmax}}\, B(a^*,b)$$

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a,b)$.

Player B's strategy is b.
Player B's return is $B(a,b)$.

Equilibria must satisfy:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a,b^*) \qquad b^* = \underset{b}{\operatorname{argmax}}\, B(a^*,b)$$

Find by solving:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a, \underset{b}{\operatorname{argmax}}\, B(a^*,b))$$

Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a,b)$.

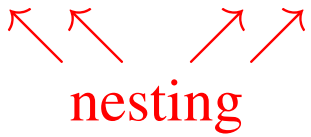
Player B's strategy is b.
Player B's return is $B(a,b)$.

Equilibria must satisfy:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a,b^*) \qquad b^* = \underset{b}{\operatorname{argmax}}\, B(a^*,b)$$

Find by solving:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a, \underset{b}{\operatorname{argmax}}\, B(a^*,b))$$



Two player non-zero-sum continuous-strategy game

Player A's strategy is a.
Player A's return is $A(a,b)$.

Player B's strategy is b.
Player B's return is $B(a,b)$.

Equilibria must satisfy:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a,b^*) \qquad b^* = \underset{b}{\operatorname{argmax}}\, B(a^*,b)$$

Find by solving:

$$a^* = \underset{a}{\operatorname{argmax}}\, A(a, \underset{b}{\operatorname{argmax}}\, B(a^*,b))$$

nesting

aka

$$\underset{a^*}{\operatorname{root}}\ \underset{a}{\operatorname{argmax}}\, A(a, \underset{b}{\operatorname{argmax}}\, B(a^*,b)) - a^*$$

more nesting

$$\underset{a^*}{\text{root}}\ \underset{a}{\text{argmax}}\, A(a, \underset{b}{\text{argmax}}\, B(a^*, b)) - a^*$$

Five levels of nesting. (One from `ROOT`, two from each `ARGMAX`.)

Continuous-Strategy Two-Person Nonzero-Sum Game

FARFEL/FARFALLEN

```
      subroutine deriv2(f, x, y, yprime)
      external f
        adf(tangent(x) = 1.0)
        y = f(x)
        end adf(yprime = tangent(y))
      end

      function root(f, x0, n)
      x = x0
      do 1669 i = 1, n
      call deriv2(f, x, y, yprime)
1669  x = x-y/yprime
      root = x
      end

      function deriv1(f, x)
      external f
        adf(tangent(x) = 1.0)
        y = f(x)
        end adf(deriv1 = tangent(y))
      end
```

Continuous-Strategy Two-Person Nonzero-Sum Game

VLAD/STALIN∇

```
(define (deriv2 f x) (j* f (times 0 f) x 1))

(define (root f x n)
 (if (zero? n)
     x
     (let (((cons y yprime) (deriv2 f x)))
      (root f (- x (/ y yprime)) (- n 1)))))

(define (deriv1 f x)
 (let (((cons y y-tangent) (j* f (times 0 f) x 1))) y-tangent))
```

Continuous-Strategy Two-Person Nonzero-Sum Game

FARFEL/FARFALLEN

```
function argmax(f, x0, n)
  function fprime(x)
  fprime = deriv1(f, x)
  end
argmax = root(fprime, x0, n)
end

subroutine eqlbrm(biga, bigb, astar, bstar, n)
external biga, bigb
  function f(astar)
    function g(a)
      function h(b)
      h = bigb(astar, b)
      end
    bstar = argmax(h, bstar, n)
    g = biga(a, bstar)
    end
  f = argmax(g, astar, n)-astar
  end
astar = root(f, astar, n)
end
```

Continuous-Strategy Two-Person Nonzero-Sum Game

VLAD/STALIN∇

```
(define (argmax f x0 n)
 (define (fprime x) (deriv1 f x))
 (root fprime x0 n))

(define (eqlbrm biga bigb astar bstar n)
 (define (f astar)
  (define (g a)
   (define (h b) (bigb astar b))
   (biga a (argmax h bstar n)))
  (- (argmax g astar n) astar))
 (let ((astar (root f astar n)))
  (define (h b) (bigb astar b))
  (let ((bstar (argmax h bstar n)))
   (cons astar bstar))))
```

Continuous-Strategy Two-Person Nonzero-Sum Game

Farfel/Farfallen

```
function gmbiga(a, b)
price = 20-0.1*a-0.1*b
costs = a*(10-0.05*a)
gmbiga = a*price-costs
end

function gmbigb(a, b)
price = 20-0.1*b-0.0999*a
costs = b*(10.005-0.05*b)
gmbigb = b*price-costs
end

program main
read *, astar
read *, bstar
read *, n
call eqlbrm(gmbiga, gmbigb, astar, bstar, n)
print *, astar, bstar
end
```

Continuous-Strategy Two-Person Nonzero-Sum Game

VLAD/STALIN∇

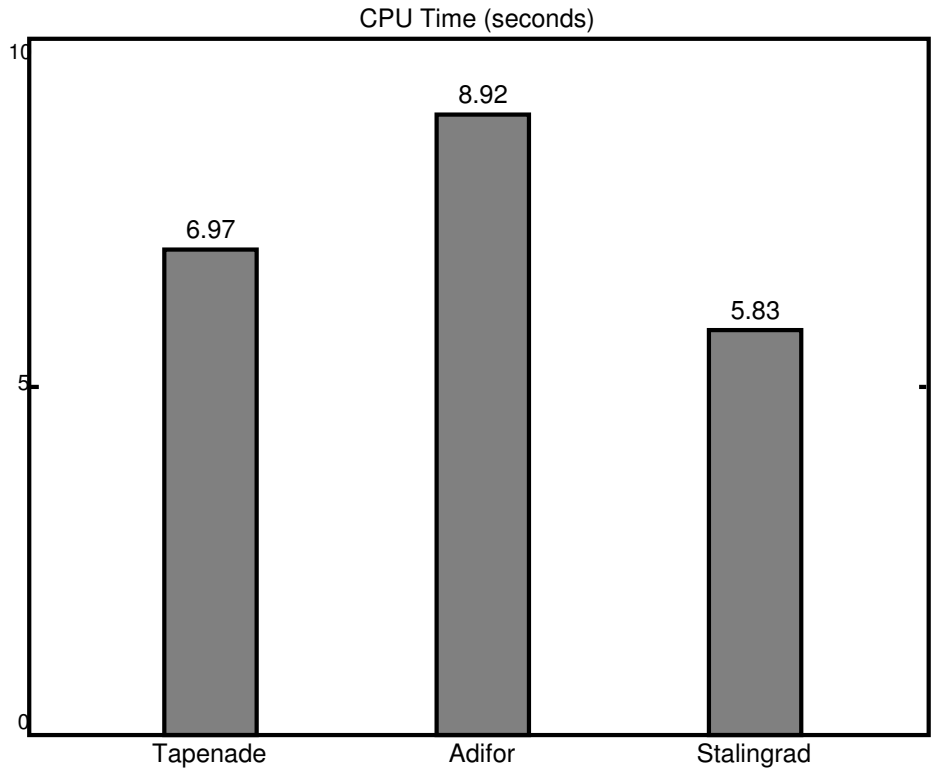
```
(define (gmbiga a b)
 (let ((price (- (- 20 (* 0.1 a)) (* 0.1 b)))
       (costs (* a (- 10 (* 0.05 a)))))
  (- (* a price) costs)))

(define (gmbigb a b)
 (let ((price (- (- 20 (* 0.1 b)) (* 0.0999 a)))
       (costs (* b (- 10.005 (* 0.05 b)))))
  (- (* b price) costs)))

(let (((cons astar bstar) (eqlbrm gmbiga gmbigb 0 0 (real 500))))
 (cons (write-real astar) (write-real bstar)))
```

The full EQLBRM example

```
      FUNCTION GMBIGB(A, B)
      PRICE = 20-0.1*B-0.0999*A
      COSTS = B*(10.005-0.05*B)
      GMBIGB = B*PRICE-COSTS
      END

      FUNCTION EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, B)
      EQLBRM_GMBIGA_GMBIGB_F_G_H = GMBIGB(ASTAR, B)
      END

      SUBROUTINE DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H_ADF(ASTAR, X, Y)
      Y = EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, X)
      END

      FUNCTION DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, X)
      X_G1 = 1.0
      ASTAR_G1 = 0.0
      Y_G1 = 0.0
      CALL DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H_ADF_G1(ASTAR, ASTAR_G1, X,
     +X_G1, Y, Y_G1)
      DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_H = Y_G1
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME = DERIV1_EQLBRM_GMBIGA_GM
     +BIGB_F_G_H(ASTAR, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME_ADF(AST
     +AR, X, Y)
      Y = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR,
     +X, Y, YPRIME)
      X_G2 = 1.0
      ASTAR_G2 = 0.0
      Y_G2 = 0.0
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME_ADF_G2(ASTAR,
     + ASTAR_G2, X, X_G2, Y, Y_G2)
      YPRIME = Y_G2
      END

      FUNCTION ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X0,
     +N)
      X = X0
      DO 1669 I = 1, N
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME(ASTAR, X, Y,
     +YPRIME)
 1669 X = X-Y/YPRIME
      ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H_FPRIME = X
      END

      FUNCTION GMBIGA(A, B)
      PRICE = 20-0.1*A-0.1*B
      COSTS = A*(10-0.05*A)
      GMBIGA = A*PRICE-COSTS
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, X0, N)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H = ROOT_ARGMAX_EQLBRM_GMBIGA_GMBI
     +GB_F_G_H_FPRIME(ASTAR, X0, N)
      END

      FUNCTION EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N, A)
      BSTAR = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_H(ASTAR, BSTAR, N)
      EQLBRM_GMBIGA_GMBIGB_F_G = GMBIGA(A, BSTAR)
      END

      SUBROUTINE DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G_ADF(ASTAR, BSTAR, N, X,
     + Y)
      Y = EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N, X)
      END

      FUNCTION DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N, X)
      X_G3 = 1.0
      ASTAR_G3 = 0.0
      BSTAR_G3 = 0.0
     +, BSTAR_G3, N, N_G3, X, X_G3, Y, Y_G3)
      DERIV1_EQLBRM_GMBIGA_GMBIGB_F_G = Y_G3
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR, N, X
     +)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME = DERIV1_EQLBRM_GMBIGA_GMBI
     +GB_F_G(ASTAR, BSTAR, N, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME_ADF(ASTAR
     +, BSTAR, N, X, Y)
      Y = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR, N, X)
      END

      SUBROUTINE DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BS
     +TAR, N, X, Y, YPRIME)
      X_G4 = 1.0
      ASTAR_G4 = 0.0
      BSTAR_G4 = 0.0
      N_G4 = 0.0
      Y_G4 = 0.0
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME_ADF_G4(ASTAR, A
     +STAR_G4, BSTAR, BSTAR_G4, N, N_G4, X, X_G4, Y, Y_G4)
      YPRIME = Y_G4
      END

      FUNCTION ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR,
     + N)
      X = ASTAR
      DO 1669 I = 1, N
      CALL DERIV2_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME(ASTAR, BSTAR, N
     +, X, Y, YPRIME)
 1669 X = X-Y/YPRIME
      ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G_FPRIME = X
      END

      FUNCTION ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BSTAR, N)
      ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G = ROOT_ARGMAX_EQLBRM_GMBIGA_GMBIGB
     +_F_G_FPRIME(ASTAR, BSTAR, N)
      END

      FUNCTION EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, ASTAR)
      EQLBRM_GMBIGA_GMBIGB_F = ARGMAX_EQLBRM_GMBIGA_GMBIGB_F_G(ASTAR, BS
     +TAR, N)-ASTAR
      END

      SUBROUTINE DERIV2_EQLBRM_GMBIGA_GMBIGB_F_ADF(BSTAR, N, X, Y)
      Y = EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X)
      END

      SUBROUTINE DERIV2_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X, Y, YPRIME)
      X_G5 = 1.0
      BSTAR_G5 = 0.0
      N_G5 = 0.0
      Y_G5 = 0.0
      CALL DERIV2_EQLBRM_GMBIGA_GMBIGB_F_ADF_G5(BSTAR, BSTAR_G5, N, N_G5
     +, X, X_G5, Y, Y_G5)
      YPRIME = Y_G5
      END

      FUNCTION ROOT_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X0)
      X = X0
      DO 1669 I = 1, N
      CALL DERIV2_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, X, Y, YPRIME)
 1669 X = X-Y/YPRIME
      ROOT_EQLBRM_GMBIGA_GMBIGB_F = X
      END

C     ASTAR & BSTAR: GUESSES IN, OPTIMIZED VALUES OUT
      SUBROUTINE EQLBRM_GMBIGA_GMBIGB(ASTAR, BSTAR, N)
      ASTAR = ROOT_EQLBRM_GMBIGA_GMBIGB_F(BSTAR, N, ASTAR)
      END

      PROGRAM MAIN
      READ *, ASTAR
      READ *, BSTAR
      READ *, N
```

```
#! /bin/bash

tapenade -root deriv1_2_adf2 -d -o eqlbrm42      \
    -diffvarname "_g2" -diffuncname "_g2"        \
    eqlbrm42.f

tapenade -root deriv2_1_2_adf4 -d -o eqlbrm42    \
    -diffvarname "_g4" -diffuncname "_g4"        \
    eqlbrm42{,_g2}.f

tapenade -root deriv1_1_adf1 -d -o eqlbrm42      \
    -diffvarname "_g1" -diffuncname "_g1"        \
    eqlbrm42{,_g2,_g4}.f

tapenade -root deriv2_1_1_adf3 -d -o eqlbrm42    \
    -diffvarname "_g3" -diffuncname "_g3"        \
    eqlbrm42{,_g2,_g4,_g1}.f

tapenade -root deriv2_2_adf5 -d -o eqlbrm42      \
    -diffvarname "_g5" -diffuncname "_g5"        \
    eqlbrm42{,_g2,_g4,_g1,_g3}.f
```

Performance Comparison



Probabilistic Lambda Calculus

$P =$ **if** x_0 **then** 0 **else if** x_1 **then** 1 **else** 2

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$\Pr(x_0 \mapsto \textbf{true}) = p_0$ $\quad$ $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$

$\Pr(x_1 \mapsto \textbf{true}) = p_1$ $\quad$ $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P =$ **if** x_0 **then** 0 **else if** x_1 **then** 1 **else** 2

$\Pr(x_0 \mapsto \textbf{true}) = p_0$ $\quad$ $\Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$

$\Pr(x_1 \mapsto \textbf{true}) = p_1$ $\quad$ $\Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$

$$\Pr(\mathcal{E}(P) = 0 | p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1 | p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2 | p_0, p_1) = (1 - p_0)(1 - p_1)$$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P =$ **if** x_0 **then** 0 **else if** x_1 **then** 1 **else** 2

$$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$$
$$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$$

$$\Pr(\mathcal{E}(P) = 0 | p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1 | p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2 | p_0, p_1) = (1 - p_0)(1 - p_1)$$

$$\prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2$$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Lambda Calculus

$P = \textbf{if } x_0 \textbf{ then } 0 \textbf{ else if } x_1 \textbf{ then } 1 \textbf{ else } 2$

$\Pr(x_0 \mapsto \textbf{true}) = p_0 \qquad \Pr(x_0 \mapsto \textbf{false}) = 1 - p_0$
$\Pr(x_1 \mapsto \textbf{true}) = p_1 \qquad \Pr(x_1 \mapsto \textbf{false}) = 1 - p_1$

$$\Pr(\mathcal{E}(P) = 0 | p_0, p_1) = p_0$$
$$\Pr(\mathcal{E}(P) = 1 | p_0, p_1) = (1 - p_0)p_1$$
$$\Pr(\mathcal{E}(P) = 2 | p_0, p_1) = (1 - p_0)(1 - p_1)$$

$$\prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = p_0(1 - p_0)^3 p_1(1 - p_1)^2$$

$$\operatorname*{argmax}_{p_0, p_1} \prod_{v \in \{0,1,2,2\}} \Pr(\mathcal{E}(P) = v | p_0, p_1) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle$$

Koller, D., McAllester, D. , and Pfeffer, A. (1997). *Effective Bayesian Inference for Stochastic Programs*. Proceedings of the 14th National Conference on Artificial Intelligence (AAAI), pp. 740–7.

Probabilistic Prolog

```
p(0).
p(X):-q(X).
q(1).
q(2).
```

Probabilistic Prolog

$\Pr(\texttt{p(0).}) = p_0$
$\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
$\Pr(\texttt{q(1).}) = p_1$
$\Pr(\texttt{q(2).}) = 1 - p_1$

Probabilistic Prolog

$\Pr(\texttt{p(0).}) = p_0$
$\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
$\Pr(\texttt{q(1).}) = p_1$
$\Pr(\texttt{q(2).}) = 1 - p_1$

$\Pr(\texttt{?-p(0).}) = p_0$
$\Pr(\texttt{?-p(1).}) = (1 - p_0)p_1$
$\Pr(\texttt{?-p(2).}) = (1 - p_0)(1 - p_1)$

Probabilistic Prolog

$\Pr(\texttt{p(0).}) = p_0$
$\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$
$\Pr(\texttt{q(1).}) = p_1$
$\Pr(\texttt{q(2).}) = 1 - p_1$

$\Pr(\texttt{?-p(0).}) = p_0$
$\Pr(\texttt{?-p(1).}) = (1 - p_0)p_1$
$\Pr(\texttt{?-p(2).}) = (1 - p_0)(1 - p_1)$

$$\prod_{q \in \{\texttt{p(0)},\texttt{p(1)},\texttt{p(2)},\texttt{p(2)}\}} \Pr(\texttt{?-}q\texttt{.}) = p_0(1 - p_0)^3 p_1(1 - p_1)^2$$

Probabilistic Prolog

$$\Pr(\texttt{p(0).}) = p_0$$
$$\Pr(\texttt{p(X):-q(X).}) = 1 - p_0$$
$$\Pr(\texttt{q(1).}) = p_1$$
$$\Pr(\texttt{q(2).}) = 1 - p_1$$

$$\Pr(\texttt{?-p(0).}) = p_0$$
$$\Pr(\texttt{?-p(1).}) = (1 - p_0)p_1$$
$$\Pr(\texttt{?-p(2).}) = (1 - p_0)(1 - p_1)$$

$$\prod_{q \in \{\texttt{p(0)},\texttt{p(1)},\texttt{p(2)},\texttt{p(2)}\}} \Pr(\texttt{?-}q\texttt{.}) = p_0(1 - p_0)^3 p_1(1 - p_1)^2$$

$$\operatorname*{argmax}_{p_0,p_1} \prod_{q \in \{\texttt{p(0)},\texttt{p(1)},\texttt{p(2)},\texttt{p(2)}\}} \Pr(\texttt{?-}q\texttt{.}) = \left\langle \frac{1}{4}, \frac{1}{3} \right\rangle$$

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment)))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(define (evaluate expression environment)
 (cond
  ((constant-expression? expression)
   (singleton-tagged-distribution
    (constant-expression-value expression)))
  ((variable-access-expression? expression)
   (lookup-value
    (variable-access-expression-variable expression) environment))
  ((lambda-expression? expression)
   (singleton-tagged-distribution
    (lambda (tagged-distribution)
     (evaluate
      (lambda-expression-body expression)
      (cons (make-binding (lambda-expression-variable expression)
                          tagged-distribution)
            environment)))))
  (else (let ((tagged-distribution
               (evaluate (application-argument expression)
                         environment)))
          (map-tagged-distribution
           (lambda (value) (value tagged-distribution))
           (evaluate (application-callee expression) environment))))))
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0  Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1  Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Lambda Calculus

```
(gradient-ascent
 (lambda (p)
  (let ((tagged-distribution
         (evaluate if x0 then 0 else if x1 then 1 else 2
                   (list Pr(x0 ↦ true) = p0   Pr(x0 ↦ false) = 1 − p0
                         Pr(x1 ↦ true) = p1   Pr(x1 ↦ false) = 1 − p1
                         ...))))
    (map-reduce
     *
     1.0
     (lambda (value)
      (likelihood value tagged-distribution))
     '(0 1 2 2))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses))))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses))))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(define (proof-distribution term clauses)
 (let ((offset ...))
  (map-reduce
   append
   '()
   (lambda (clause)
    (let ((clause (alpha-rename clause offset)))
     (let loop ((p (clause-p clause))
                (substitution (unify term (clause-term clause)))
                (terms (clause-terms clause)))
      (if (boolean? substitution)
          '()
          (if (null? terms)
              (list (make-double p substitution))
              (map-reduce
               append
               '()
               (lambda (double)
                (loop (* p (double-p double))
                      (append substitution (double-substitution double))
                      (rest terms)))
               (proof-distribution
                (apply-substitution substitution (first terms)) clauses)))))))
   clauses)))
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p_0
                       Pr(p(X):-q(X).) = 1 − p_0
                       Pr(q(1).) = p_1
                       Pr(q(2).) = 1 − p_1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p_0
                       Pr(p(X):-q(X).) = 1 − p_0
                       Pr(q(1).) = p_1
                       Pr(q(2).) = 1 − p_1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p0
                       Pr(p(X):-q(X).) = 1 − p0
                       Pr(q(1).) = p1
                       Pr(q(2).) = 1 − p1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p0
                       Pr(p(X):-q(X).) = 1 − p0
                       Pr(q(1).) = p1
                       Pr(q(2).) = 1 − p1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p_0
                       Pr(p(X):-q(X).) = 1 − p_0
                       Pr(q(1).) = p_1
                       Pr(q(2).) = 1 − p_1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p0
                       Pr(p(X):-q(X).) = 1 − p0
                       Pr(q(1).) = p1
                       Pr(q(2).) = 1 − p1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
 '(0.5 0.5)
 1000.0
 0.1)
```

Probabilistic Prolog

```
(gradient-ascent
 (lambda (p)
  (let ((clauses (list Pr(p(0).) = p0
                       Pr(p(X):-q(X).) = 1 − p0
                       Pr(q(1).) = p1
                       Pr(q(2).) = 1 − p1)))
   (map-reduce
    *
    1.0
    (lambda (query)
     (likelihood (proof-distribution query clauses)))
    '(p(0) p(1) p(2) p(2)))))
'(0.5 0.5)
1000.0
0.1)
```

Generated Code

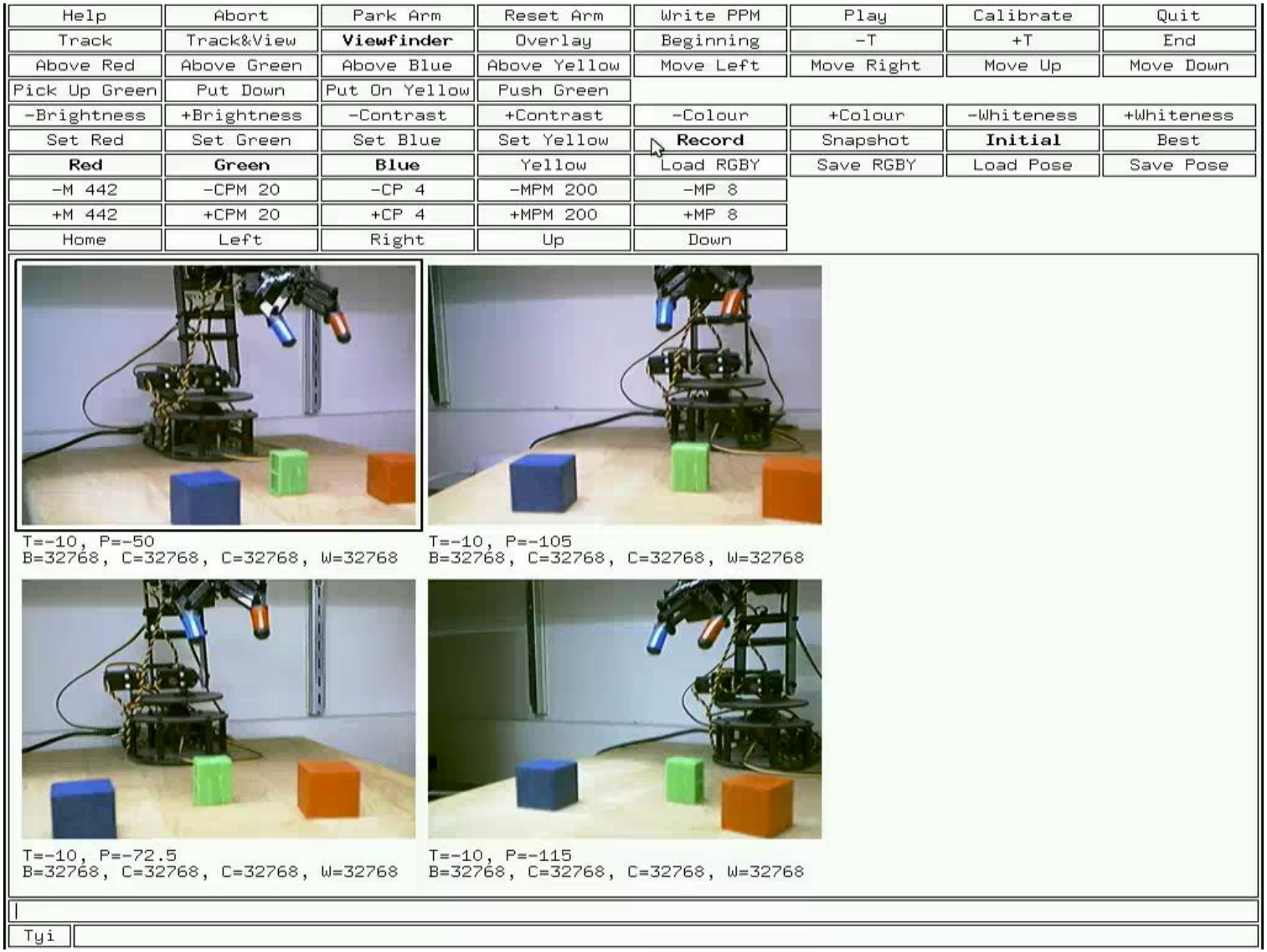
```
static void f2679(double a_f2679_0,double a_f2679_1,double a_f2679_2,double a_f2679_3){
  int t272381=((a_f2679_2==0.)?0:1);
  double t272406;
  double t272405;
  double t272404;
  double t272403;
  double t272402;
  if((t272381==0)){
    double t272480=(1.-a_f2679_0);
    double t272572=(1.-a_f2679_1);
    double t273043=(a_f2679_0+0.);
    double t274185=(t272480*a_f2679_1);
    double t274426=(t274185+0.);
    double t275653=(t272480*t272572);
    double t275894=(t275653+0.);
    double t277121=(t272480*t272572);
    double t277362=(t277121+0.);
    double t277431=(t277362*1.);
    double t277436=(t275894*t277431);
    double t277441=(t274426*t277436);
    double t277446=(t273043*t277441);
    ...
    double t1777107=(t1774696+t1715394);
    double t1777194=(0.-t1745420);
    double t1778533=(t1777194+t1419700);
    t272406=a_f2679_0;
    t272405=a_f2679_1;
    t272404=t277446;
    t272403=t1778533;
    t272402=t1777107;}
  else {...}
  r_f2679_0=t272406;
  r_f2679_1=t272405;
  r_f2679_2=t272404;
  r_f2679_3=t272403;
  r_f2679_4=t272402;}
```

Performance Comparison

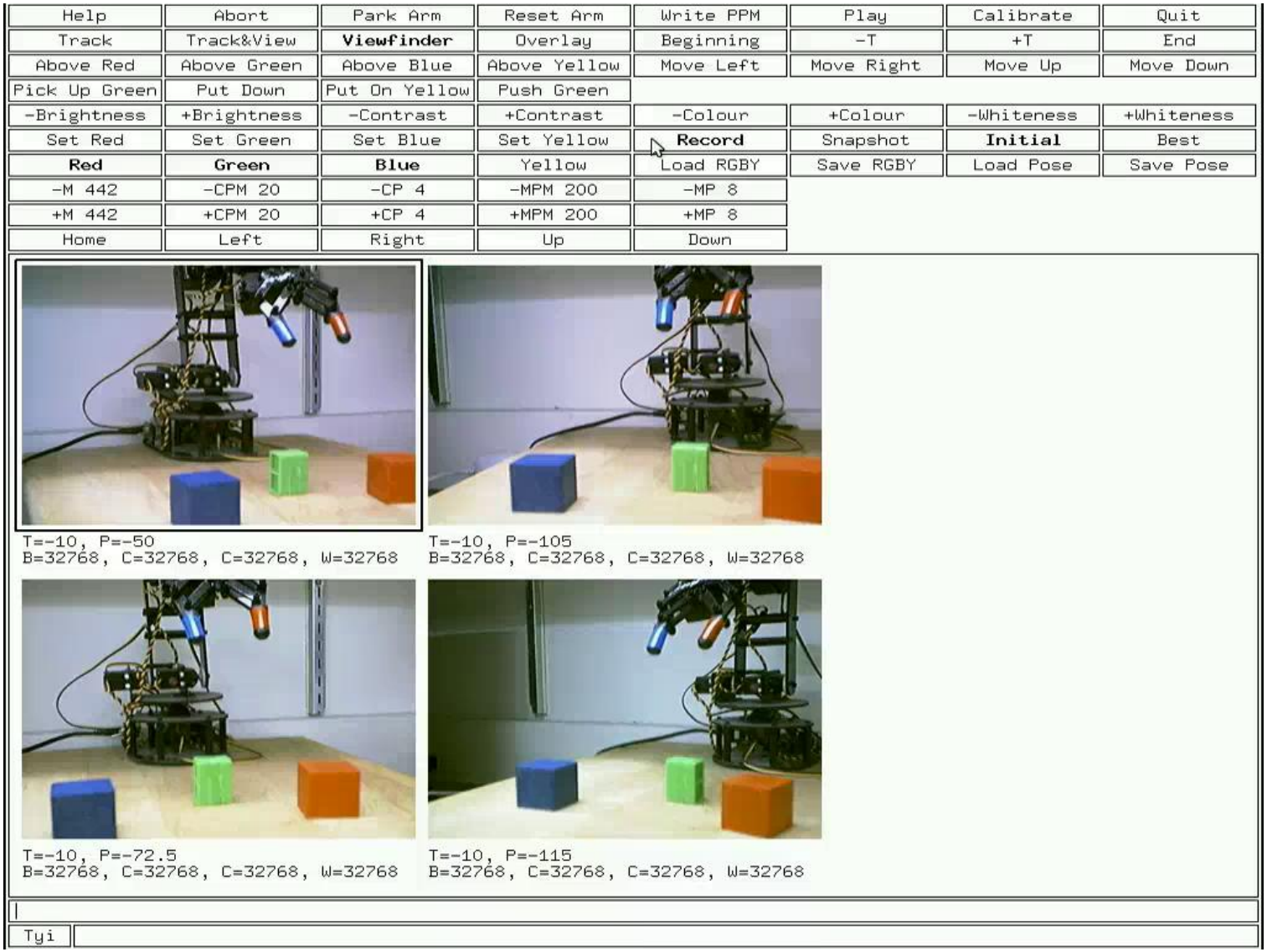
	probabilistic-lambda-calculus		probabilistic-prolog	
	F	R	F	R
STALIN∇	1.00	1.00	1.00	1.00
MLTON	106.45	124.95	789.41	483.47
OCAML	215.73	538.68	1207.13	1534.61
SML/NJ	197.75	272.45	2448.02	1471.94
GHC	■	■	■	■
BIGLOO	832.92	1048.11	14422.16	8286.06
CHICKEN	2305.98	3283.00	66948.70	37792.84
GAMBIT	879.88	1153.86	24316.03	13649.81
IKARUS	437.46	531.10	8242.92	4845.86
LARCENY	1651.01	1673.22	25589.62	14833.53
MIT SCHEME	3491.10	4130.19	85819.57	48335.38
MZC	5289.17	5929.14	154206.95	83480.27
MZSCHEME	6235.78	7134.71	166129.12	91630.70
SCHEME->C	682.15	794.31	10530.66	5980.27
SCMUTILS	6456.99	■	80100.23	■
STALIN	1240.73	1137.41	22511.79	10986.43

- ■ not implemented but could implement, including FORTRAN, C, and C++
- ■ not implemented in existing tool
- ■ can't implement

Demo of Inverting Motor Programs



Demo of Inverting Motor Programs



How It Works

How It Works

z	world coordinates of executor's end effector

How It Works

z	world coordinates of executor's end effector
a	arm configuration

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing

x_o

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing

$$p^{-1}(x_o, c_o)$$

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing

$$p(p^{-1}(x_o, c_o), c_e)$$

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing

$$m(p(p^{-1}(x_o, c_o), c_e))$$

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing

$$f(m(p(p^{-1}(x_o, c_o), c_e)), b)$$

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing
$\hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o)$	imagination capacity

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing
$\hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o)$	imagination capacity
$\text{argmin}_m \|x_o - \mathbf{lsh}(\hat{x}_o)\|$	classification

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing
$\hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o)$	imagination capacity
$\text{argmin}_m \|x_o - \textbf{lsh}(\hat{x}_o)\|$	classification

How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing
$\hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o)$	imagination capacity
$\text{argmin}_m \|x_o - \mathbf{lsh}(\hat{x}_o)\|$	classification
$\text{argmin}_m \min_{b,c_o,c_e} \|x_o - \mathbf{lsh}(\hat{x}_o)\|$	pose estimation by gradient descent

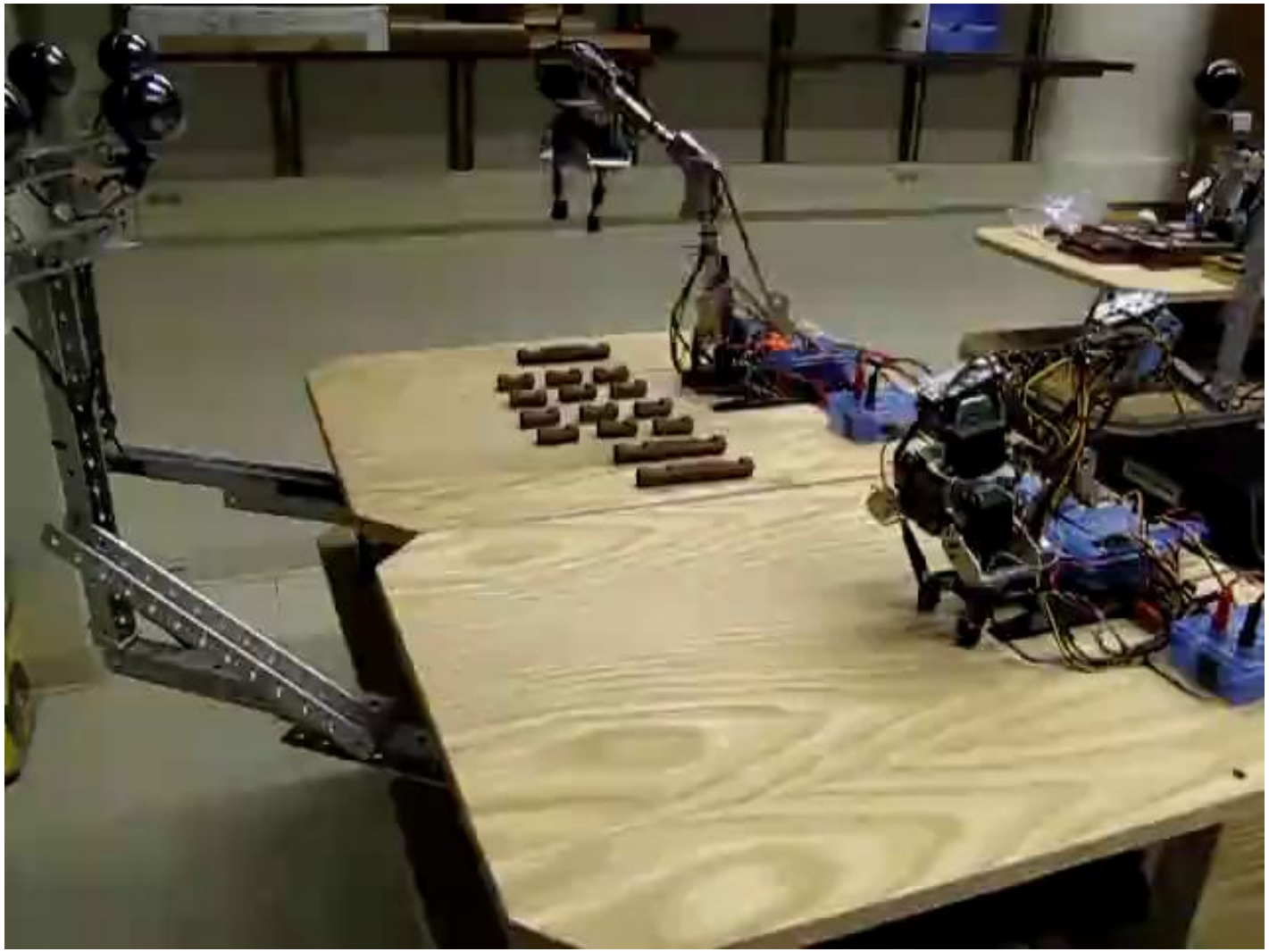
How It Works

z	world coordinates of executor's end effector
a	arm configuration
b	robot pose
$c_e = (c_{el}, c_{er}), c_o = (c_{ol}, c_{or})$	camera poses
$x_e = (x_{el}, x_{er}), x_o = (x_{ol}, x_{or})$	image coordinates of executor's end effector
$z = f(a, b)$	forward kinematics
$x = p(z, c)$	forward optics
$z = p^{-1}(x_l, x_r, c_l, c_r)$	inverse optics
	binocular vision
	not stereo
	robustness in the face of occlusion
	visual servoing along all axes
$a = m(x_e)$	motor program
	inverse kinematics
	closed-loop visual servoing
$\hat{x}_o = p(f(m(p(p^{-1}(x_o, c_o), c_e)), b), c_o)$	imagination capacity
$\text{argmin}_m \|x_o - \textbf{lsh}(\hat{x}_o)\|$	classification
$\text{argmin}_m \ \min_{b,c_o,c_e} \|x_o - \textbf{lsh}(\hat{x}_o)\|$	pose estimation by gradient descent
$\nabla_{b,c_o,c_e} \|x_o - \textbf{lsh}(\hat{x}_o)\|$	calculated using AD

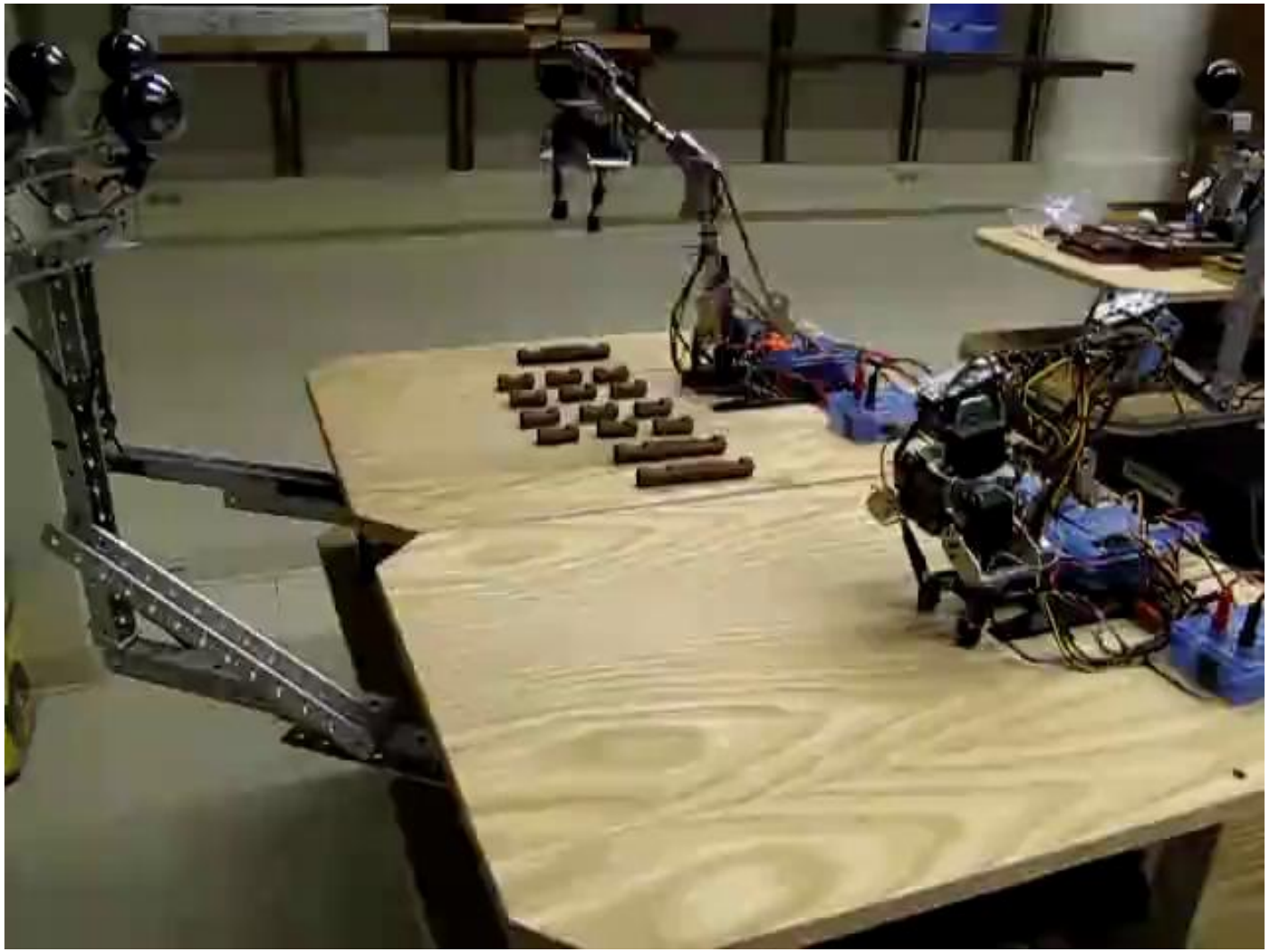
Robot Collaboration

Robot Collaboration

Robot Collaboration



Robot Collaboration



Forward kinematics

$$\mathbf{x} = f(\theta, \mathbf{p})$$

How It Works

Forward kinematics

$$\mathbf{x} = f(\theta, \mathbf{p})$$

Estimating the parameters of the kinematic chain

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmin}} \sum_i \|\mathbf{x}_i - f(\theta_i, \mathbf{p})\|^2$$

Forward kinematics

$$\mathbf{x} = f(\theta, \mathbf{p})$$

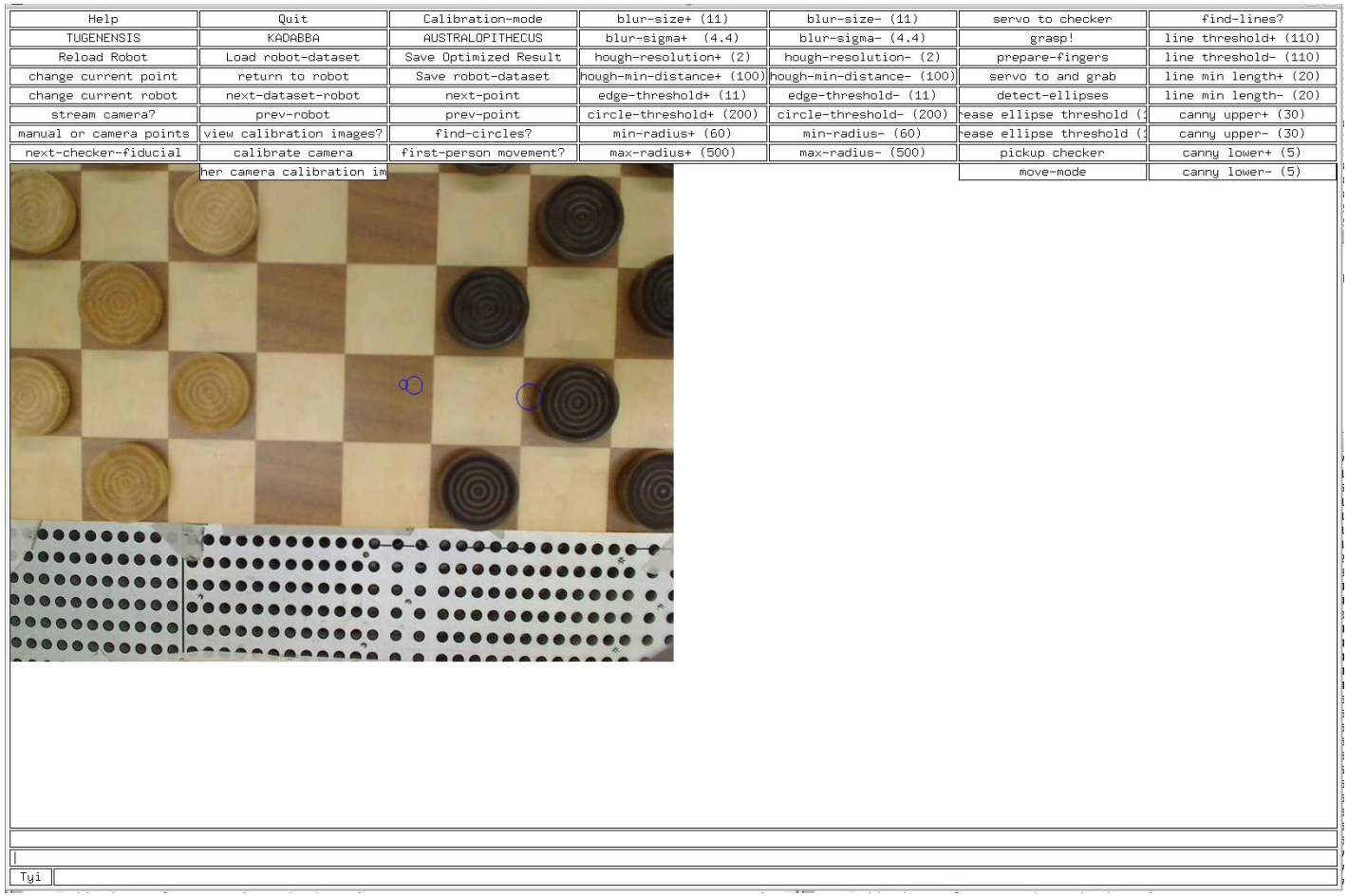
Estimating the parameters of the kinematic chain

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmin}} \sum_i \|\mathbf{x}_i - f(\theta_i, \mathbf{p})\|^2$$

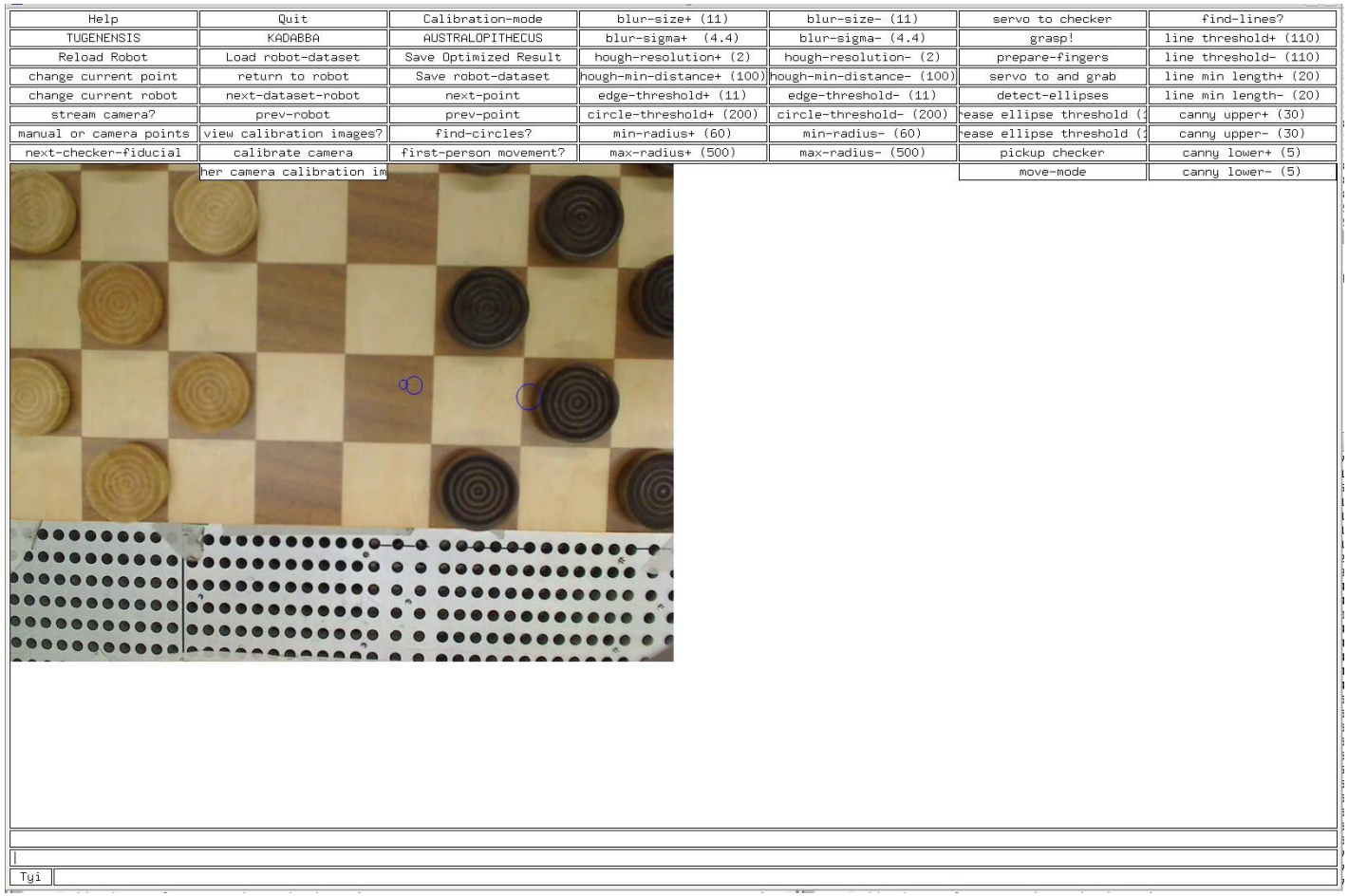
Inverse kinematics

$$\begin{aligned} \theta &= f^{-1}(\mathbf{x}, \mathbf{p}) \\ &= \underset{\theta}{\operatorname{argmin}} \|\mathbf{x} - f(\theta, \mathbf{p})\|^2 \end{aligned}$$

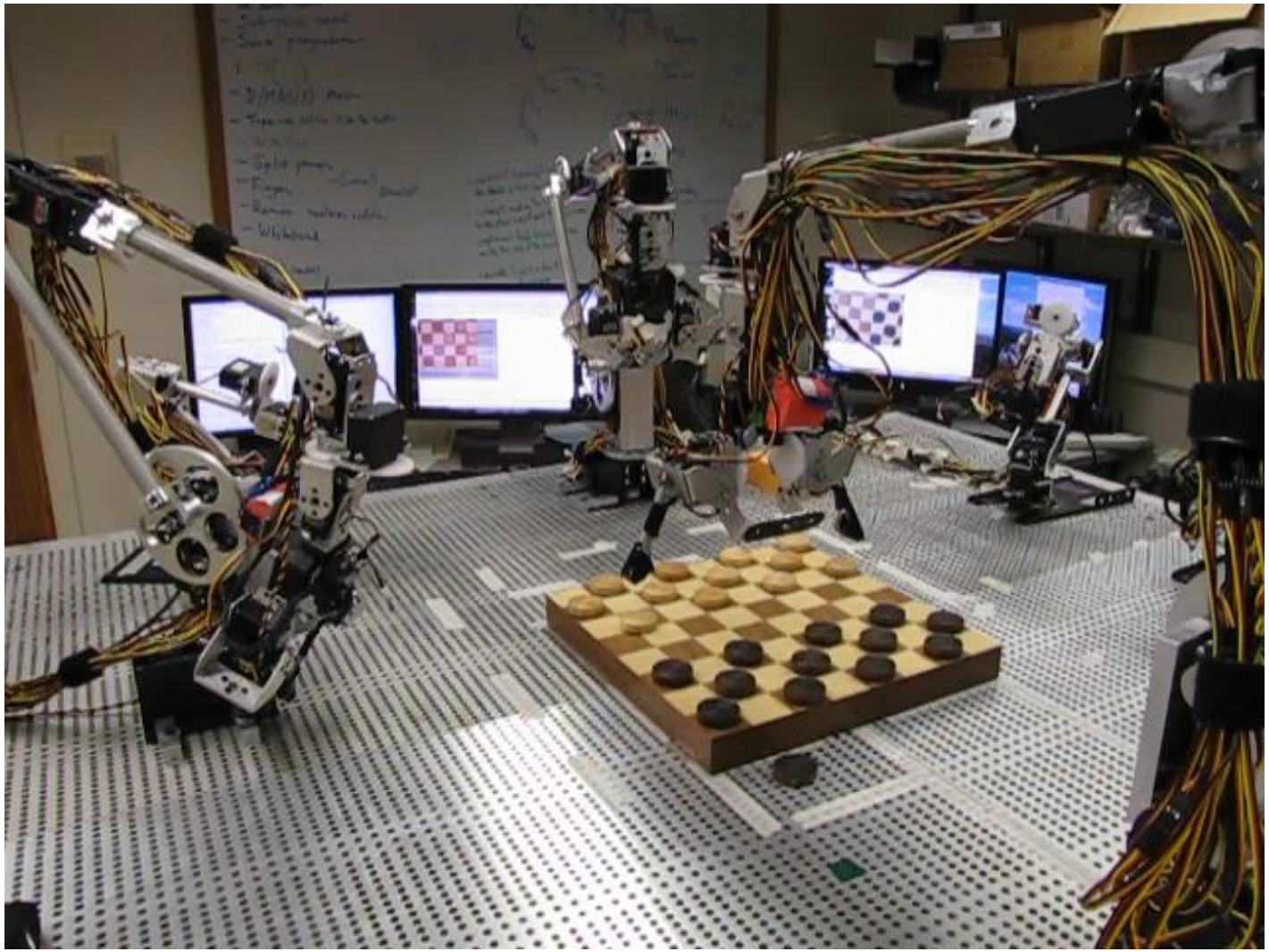
Examples


Help
Quit
Calibration-mode
blur-size+ (11)
blur-size- (11)
servo to checker
find-lines?
TUGENENSIS
KADABBA
AUSTRALOPITHECUS
blur-sigma+ (4.4)
blur-sigma- (4.4)
grasp!
line threshold+ (110)
Reload Robot
Load robot-dataset
Save Optimized Result
hough-resolution+ (2)
hough-resolution- (2)
prepare-fingers
line threshold- (110)
change current point
return to robot
Save robot-dataset
hough-min-distance+ (100)
hough-min-distance- (100)
servo to and grab
line min length+ (20)
change current robot
next-dataset-robot
next-point
edge-threshold+ (11)
edge-threshold- (11)
detect-ellipses
line min length- (20)
stream camera?
prev-robot
prev-point
circle-threshold+ (200)
circle-threshold- (200)
canny upper+ (30)
manual or camera points
view calibration images?
find-circles?
min-radius+ (60)
min-radius- (60)
canny upper- (30)
next-checker-fiducial
calibrate camera
first-person movement?
max-radius+ (500)
max-radius- (500)
pickup checker
canny lower+ (5)
move-mode
canny lower- (5)
Tyi

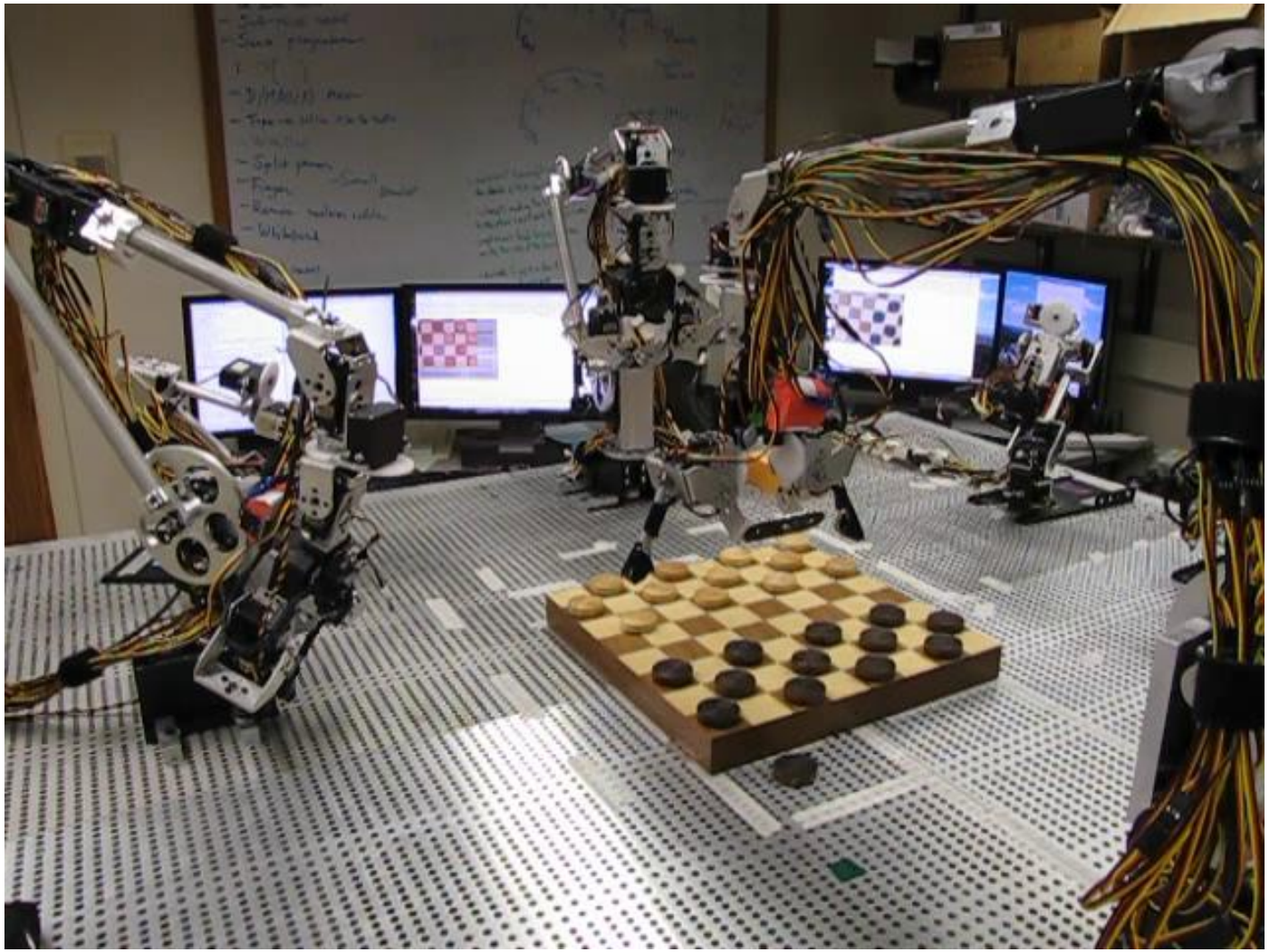
Examples


Help
Quit
Calibration-mode
blur-size+ (11)
blur-size- (11)
servo to checker
find-lines?
TUGENENSIS
KADABBA
AUSTRALOPITHECUS
blur-sigma+ (4.4)
blur-sigma- (4.4)
grasp!
line threshold+ (110)
Reload Robot
Load robot-dataset
Save Optimized Result
hough-resolution+ (2)
hough-resolution- (2)
prepare-fingers
line threshold- (110)
change current point
return to robot
Save robot-dataset
hough-min-distance+ (100)
hough-min-distance- (100)
servo to and grab
line min length+ (20)
change current robot
next-dataset-robot
next-point
edge-threshold+ (11)
edge-threshold- (11)
detect-ellipses
line min length- (20)
stream camera?
prev-robot
prev-point
circle-threshold+ (200)
circle-threshold- (200)
canny upper+ (30)
manual or camera points
view calibration images?
find-circles?
min-radius+ (60)
min-radius- (60)
canny upper- (30)
next-checker-fiducial
calibrate camera
first-person movement?
max-radius+ (500)
max-radius- (500)
pickup checker
canny lower+ (5)
move-mode
canny lower- (5)
Tyi

Examples



Examples



`(flip` p`)`

`(flip` p`)`

`(probability` e`)`

`(flip` p`)`

`(probability` e`)`

`(argmax` f x_0`)`

Reduced Gradient

Wolfe (1962, 1967)

$$\underset{\mathbf{x}}{\operatorname{argmax}} f(\mathbf{x})$$

Reduced Gradient

Wolfe (1962, 1967)

$$\underset{\mathbf{x}}{\operatorname{argmax}} f(\mathbf{x})$$

$$\underset{\substack{\mathbf{x} \\ \mathbf{x} \geq 0 \\ \mathbf{Ax}=\mathbf{b}}}{\operatorname{argmax}} f(\mathbf{x})$$

Reduced Gradient

Wolfe (1962, 1967)

$$\underset{\mathbf{x}}{\operatorname{argmax}} f(\mathbf{x})$$

$$\underset{\substack{\mathbf{x} \\ \mathbf{x} \geq 0 \\ \mathbf{A}\mathbf{x} = \mathbf{b}}}{\operatorname{argmax}} f(\mathbf{x})$$

$$\underset{\substack{\mathbf{x} \\ \mathbf{x} \geq 0 \\ \sum \mathbf{x} = 1}}{\operatorname{argmax}} f(\mathbf{x})$$

Reduced Gradient

Wolfe (1962, 1967)

$$\underset{\mathbf{x}}{\operatorname{argmax}} f(\mathbf{x})$$

$$\underset{\substack{\mathbf{x} \\ \mathbf{x} \geq 0 \\ \mathbf{A}\mathbf{x} = \mathbf{b}}}{\operatorname{argmax}} f(\mathbf{x})$$

$$\underset{\substack{\mathbf{x} \\ \mathbf{x} \geq 0 \\ \sum \mathbf{x} = 1}}{\operatorname{argmax}} f(\mathbf{x})$$

```
(argmax f x0)
```

```
(define (lexicon game-state)
 (let ((things
        (append
         (map-n (lambda (position) (list 'position position)) 9)
         (map-n (lambda (position)
                  (list 'position-state
                        position
                        (list-ref game-state position)))
                9))))
  (list
   (cons 'the ⟨meaning for the⟩)
   (cons 'x ⟨meaning for x⟩)
   (cons 'is-on ⟨meaning for is on⟩)
   (cons 'center ⟨meaning for center⟩))))
```

```
(define (draw distribution)
 (let loop ((p 1)
            (pairs
             (remove-if
              (lambda (pair) (zero? (cdr pair)))
              distribution)))
  (if (or (null? (rest pairs))
          (flip (/ (cdr (first pairs)) p)))
      (car (first pairs))
      (loop (- p (cdr (first pairs)))
            (rest pairs)))))
```

```
(define (position-state-draw distribution)
 (draw (map cons
            '(empty x o)
            distribution)))
```

```
(define (word-draw distribution)
 (draw (map cons
            '(the x is-on center)
            distribution)))
```

```
(define (interpret words game-state)
 (if (= (length words) 1)
     (cdr (assq (first words) (lexicon game-state)))
     (let* ((i (+ (draw
                   (map
                    cons
                    (enumerate (- (length words) 1))
                    (uniform (- (length words) 1))))
                  1))
            (left (interpret
                   (sublist words 0 i)
                   game-state))
            (right (interpret
                    (sublist words i (length words))
                    game-state)))
       (if (flip 0.5) (left right) (right left)))))
```

```
(argmax
 (lambda (distributions)
  (probability
   (interpret
    (map word-draw distributions)
    '(empty empty empty
      empty x     empty
      empty empty empty))))
 (map-n (lambda (i) (uniform 4)) 5))
```

```
(argmax
 (lambda (distributions)
  (probability
   (interpret
    (map word-draw distributions)
    '(empty empty empty
      empty x     empty
      empty empty empty))))
 (map-n (lambda (i) (uniform 4)) 5))

 #(#(0 41 0 59)                ;x/center
   #(100 0 0 0)                ;the
   #(0 0 100 0)                ;is-on
   #(100 0 0 0)                ;the
   #(0 41 0 59))               ;x/center
```

```
(argmax
 (lambda (distributions)
  (probability
   (interpret
    '(the x is-on the center)
    (map position-state-draw distributions))))
 (map-n (lambda (i) (uniform 3)) 9))
```

```
(argmax
 (lambda (distributions)
  (probability
   (interpret
    '(the x is-on the center)
    (map position-state-draw distributions))))
 (map-n (lambda (i) (uniform 3)) 9))
```

```
#(#(67 0 33) #(67 0 33) #(67 0 33)   ;empty/o empty/o empty/o
  #(67 0 33) #(0 100 0) #(67 0 33)   ;empty/o x       empty/o
  #(67 0 33) #(67 0 33) #(67 0 33))  ;empty/o empty/o empty/o
```

Recognition by Sequences of Appearance and Motion

Given a set of labeled training videos with bounding boxes

Recognition by Sequences of Appearance and Motion

Given a set of labeled training videos with bounding boxes

Learn the characteristic sequence of appearance and motion of each action.

Recognition by Sequences of Appearance and Motion

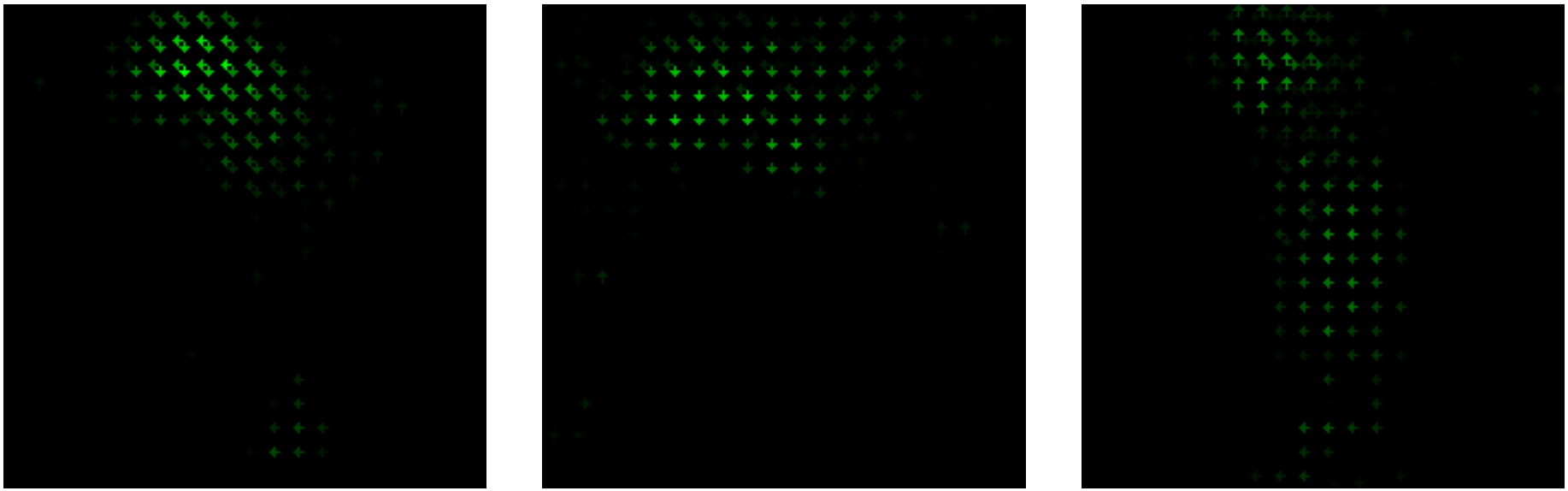
Given a set of labeled training videos with bounding boxes

Learn the characteristic sequence of appearance and motion of each action.

Recognition by Sequences of Appearance and Motion

Given a set of labeled training videos with bounding boxes

Learn the characteristic sequence of appearance and motion of each action.



Representing Appearance

Representing Appearance

- Histograms of Oriented Gradients (HOG) (Dalal & Triggs 2005a)

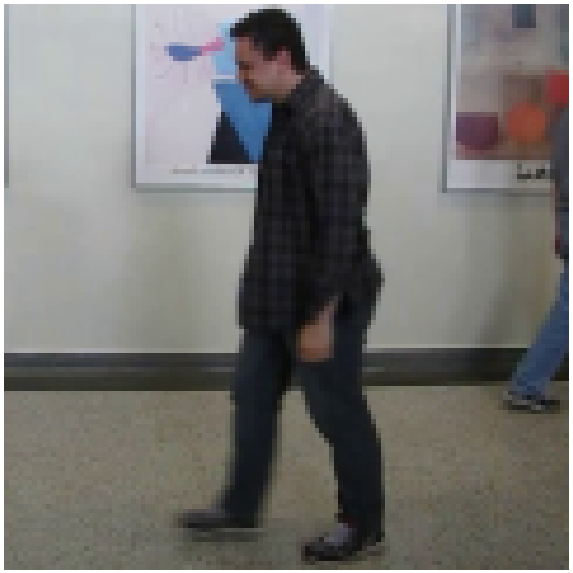
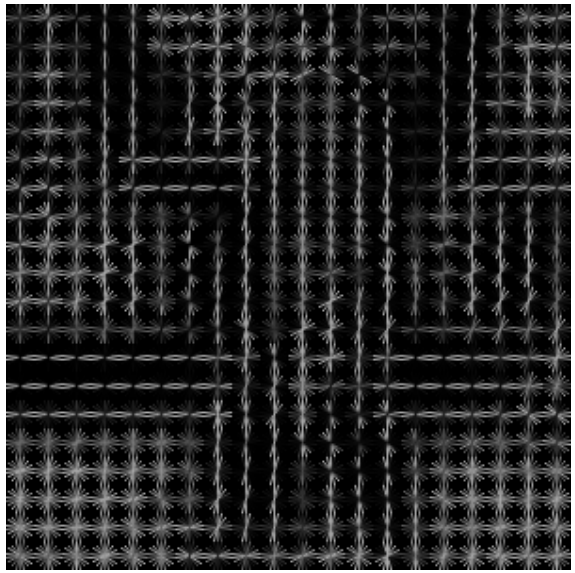
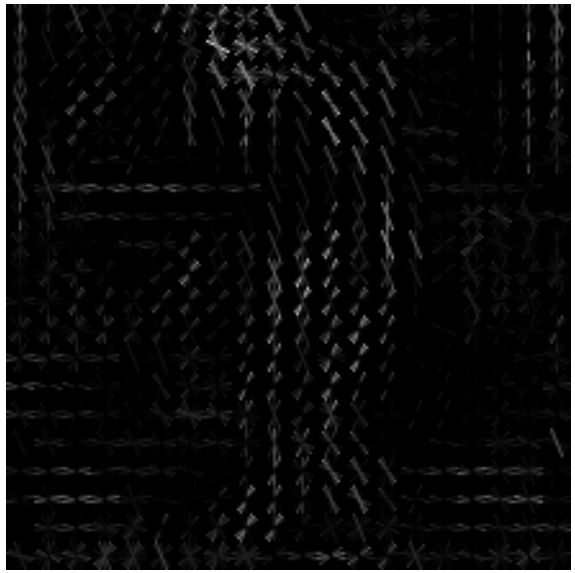


Image within box



HOG Feature



Learned HOG model

Representing Appearance

- Histograms of Oriented Gradients (HOG) (Dalal & Triggs 2005a)

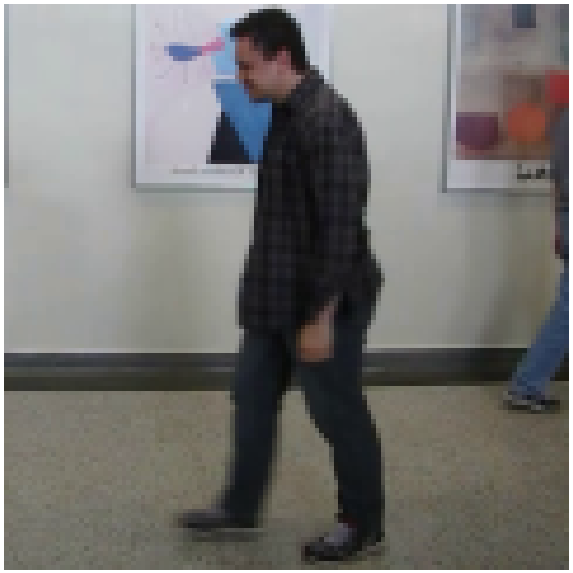
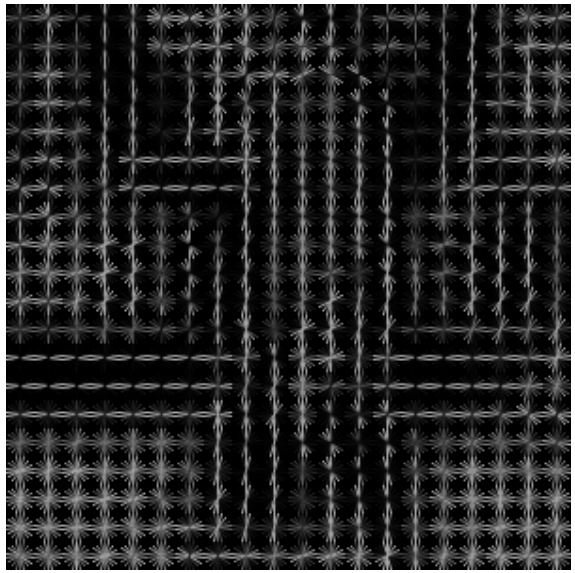
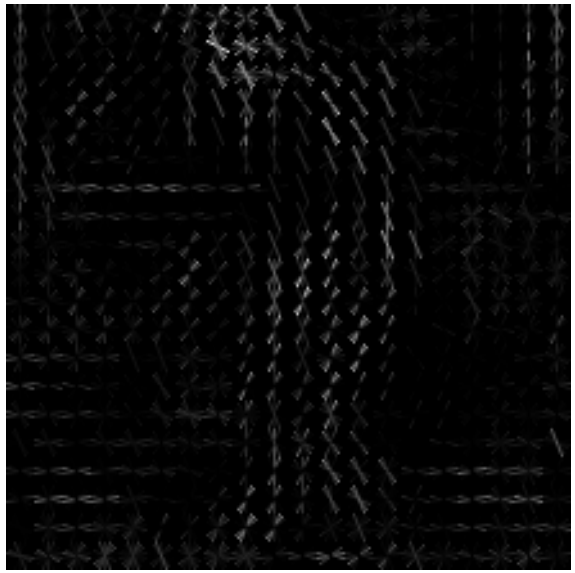


Image within box



HOG Feature



Learned HOG model

- Matching score computed with a dot product

Representing Motion Patterns

- Histograms of Oriented Flow (HOF) (Dalal & Triggs 2005b)

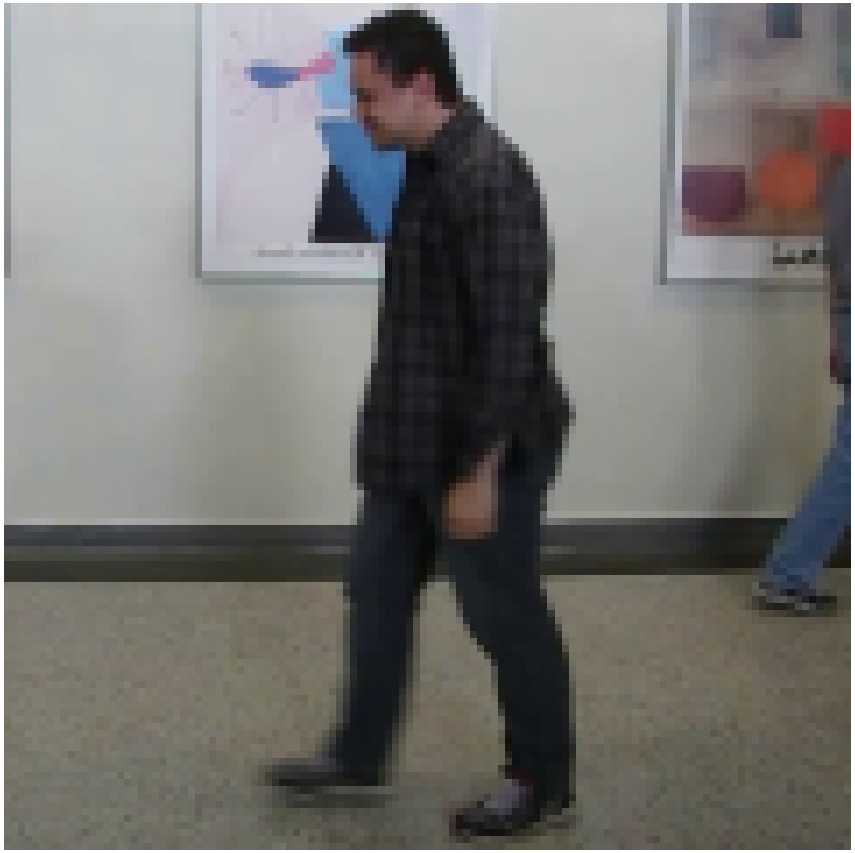
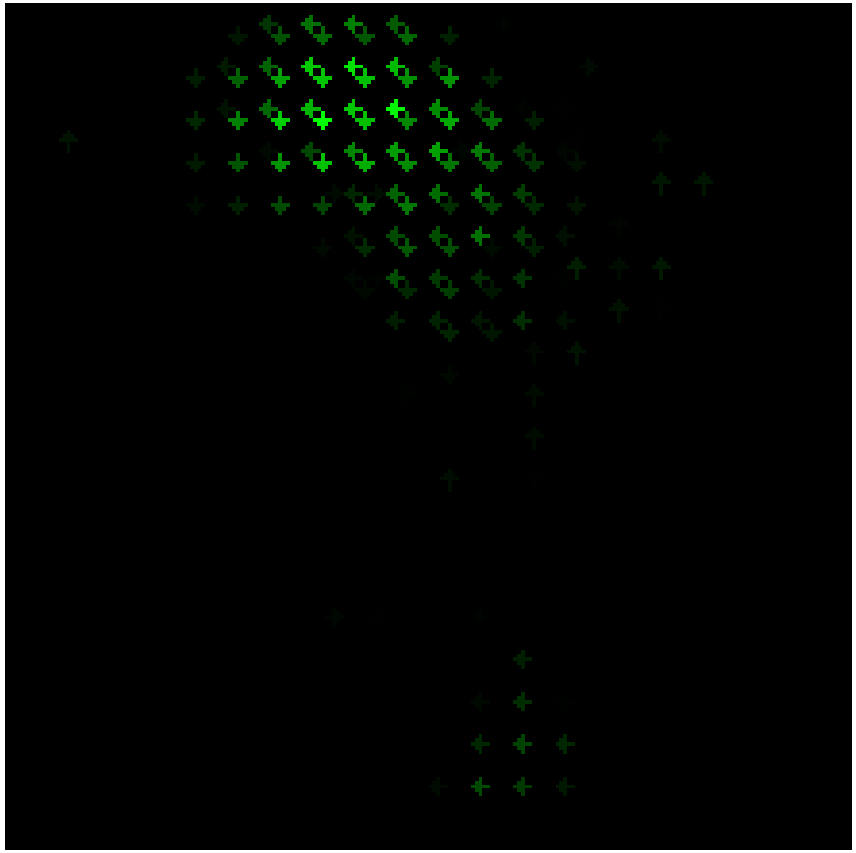


Image within box



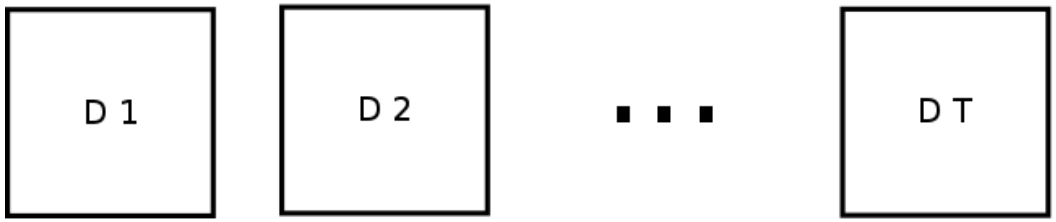
Learned Flow model

Representing the Sequence of an Action

Hidden Markov Models (HMM) (Baum & Petrie 1966)

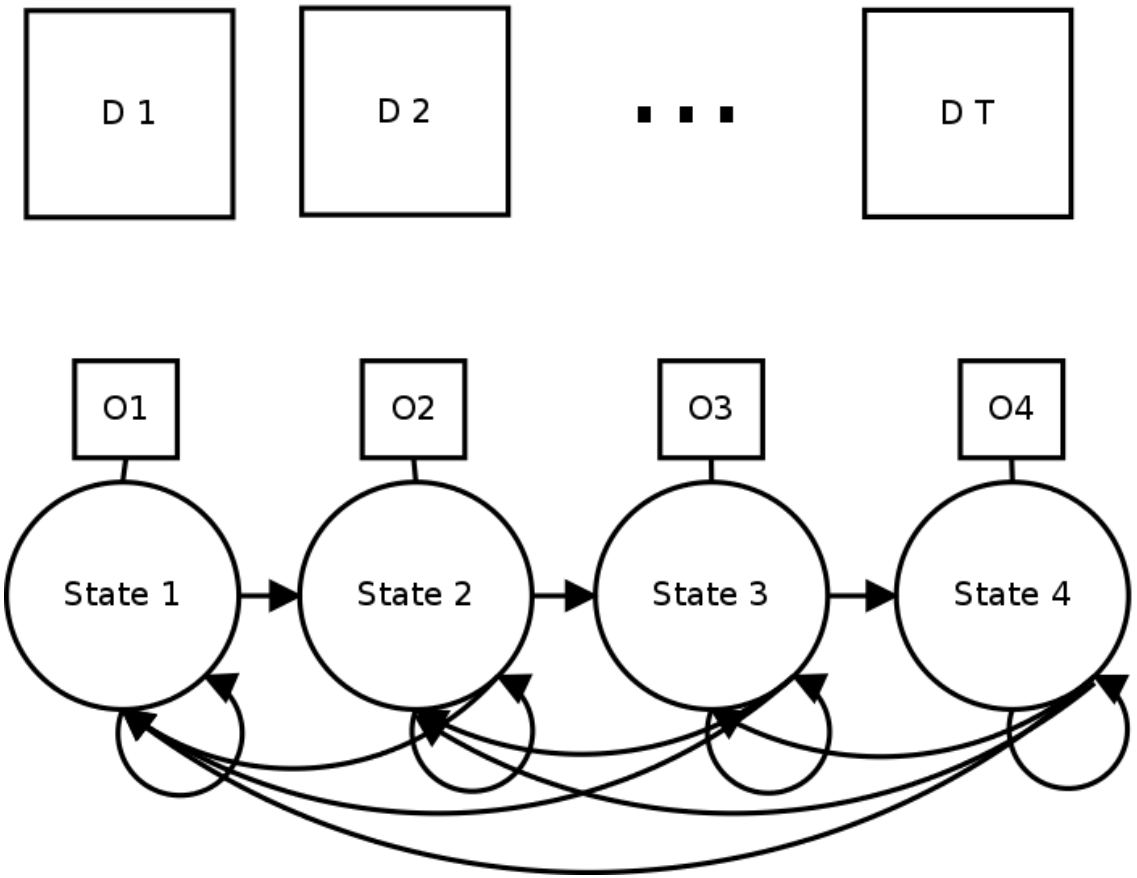
Representing the Sequence of an Action

Hidden Markov Models (HMM) (Baum & Petrie 1966)



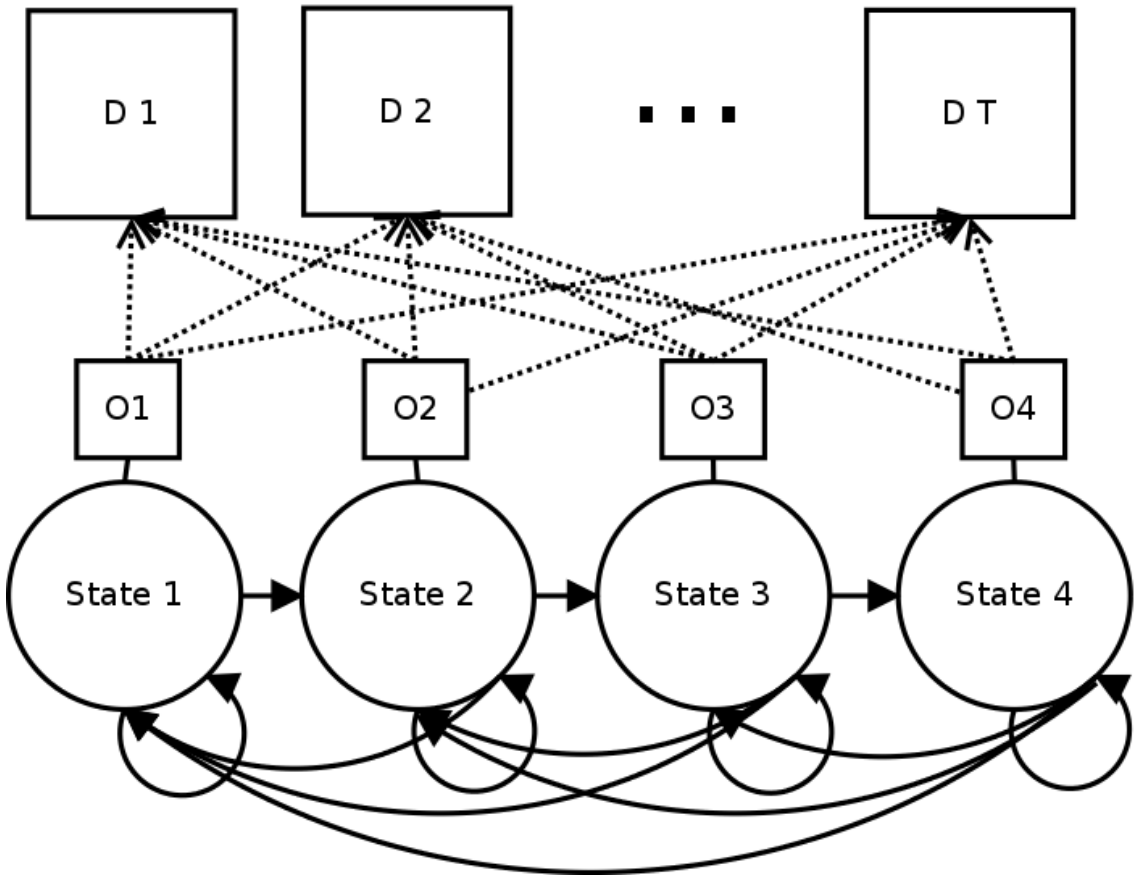
Representing the Sequence of an Action

Hidden Markov Models (HMM) (Baum & Petrie 1966)



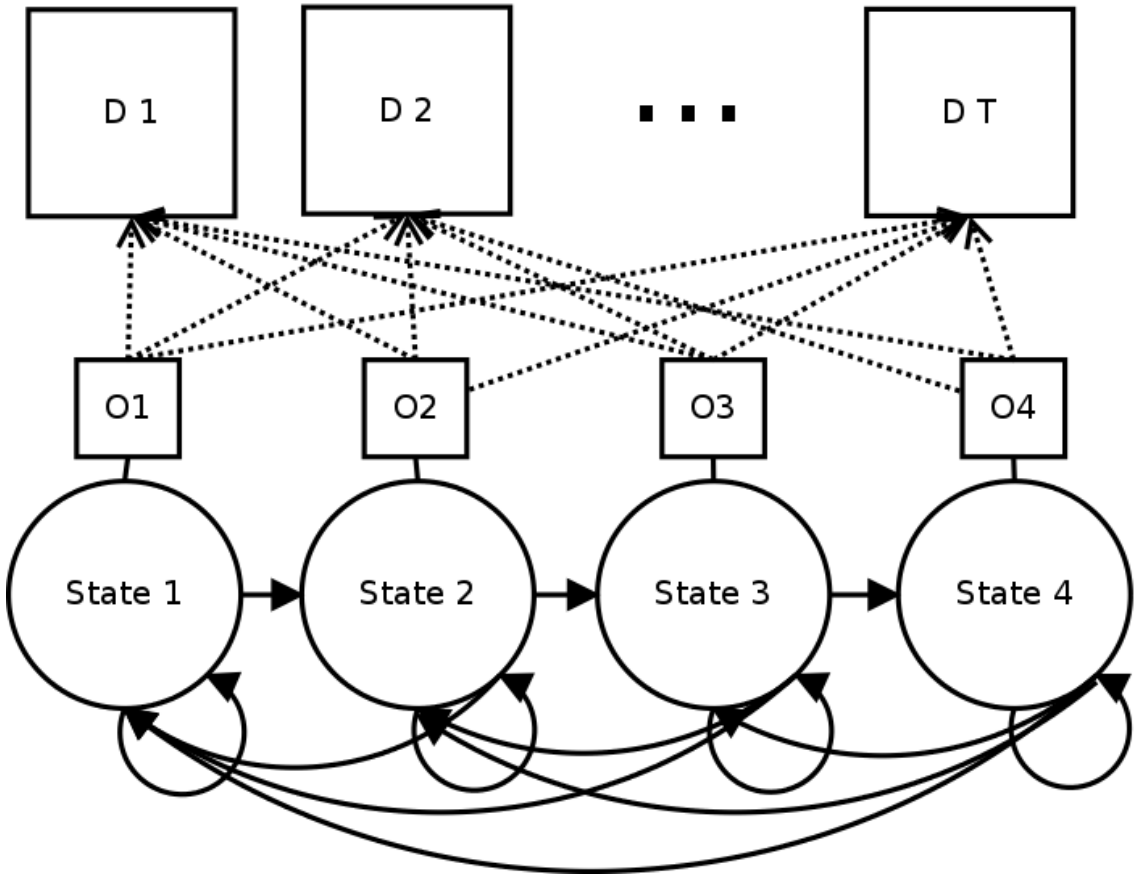
Representing the Sequence of an Action

Hidden Markov Models (HMM) (Baum & Petrie 1966)



Representing the Sequence of an Action

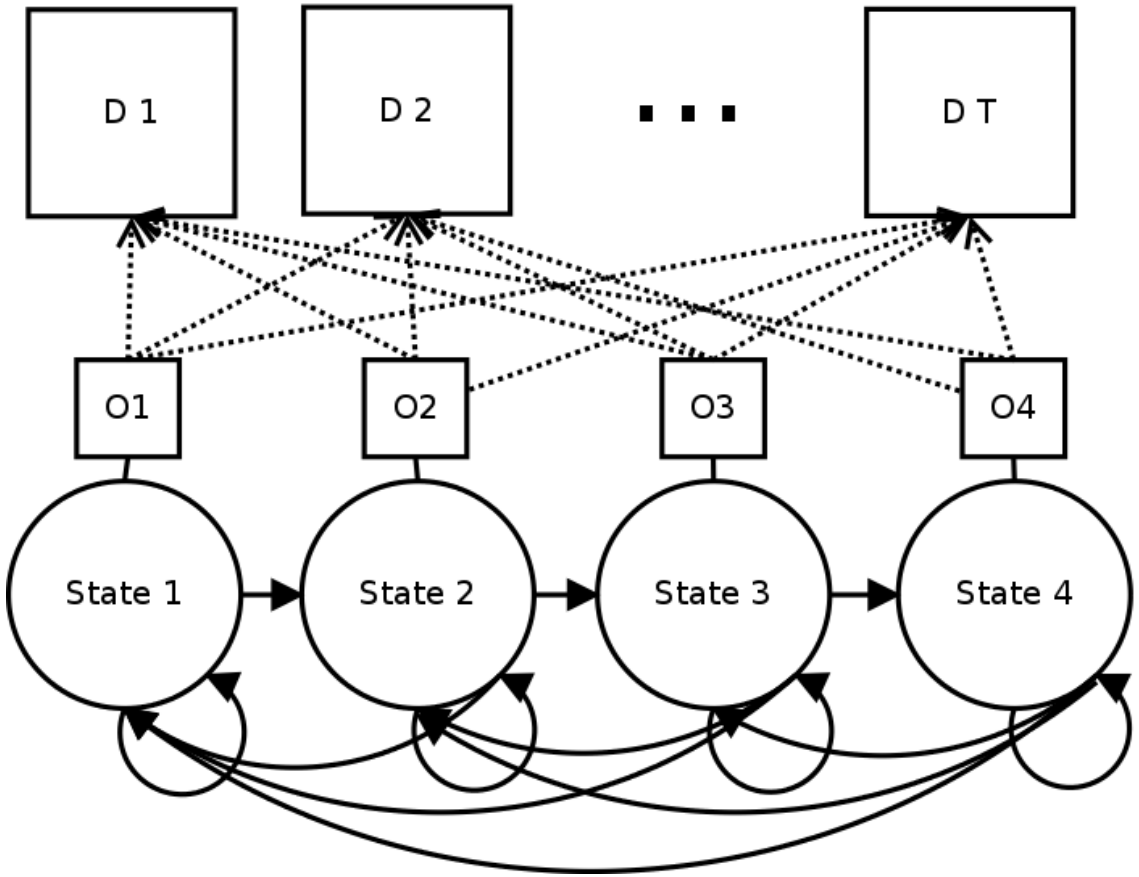
Hidden Markov Models (HMM) (Baum & Petrie 1966)



Likelihood of sequence computed with the Forward Algorithm

Representing the Sequence of an Action

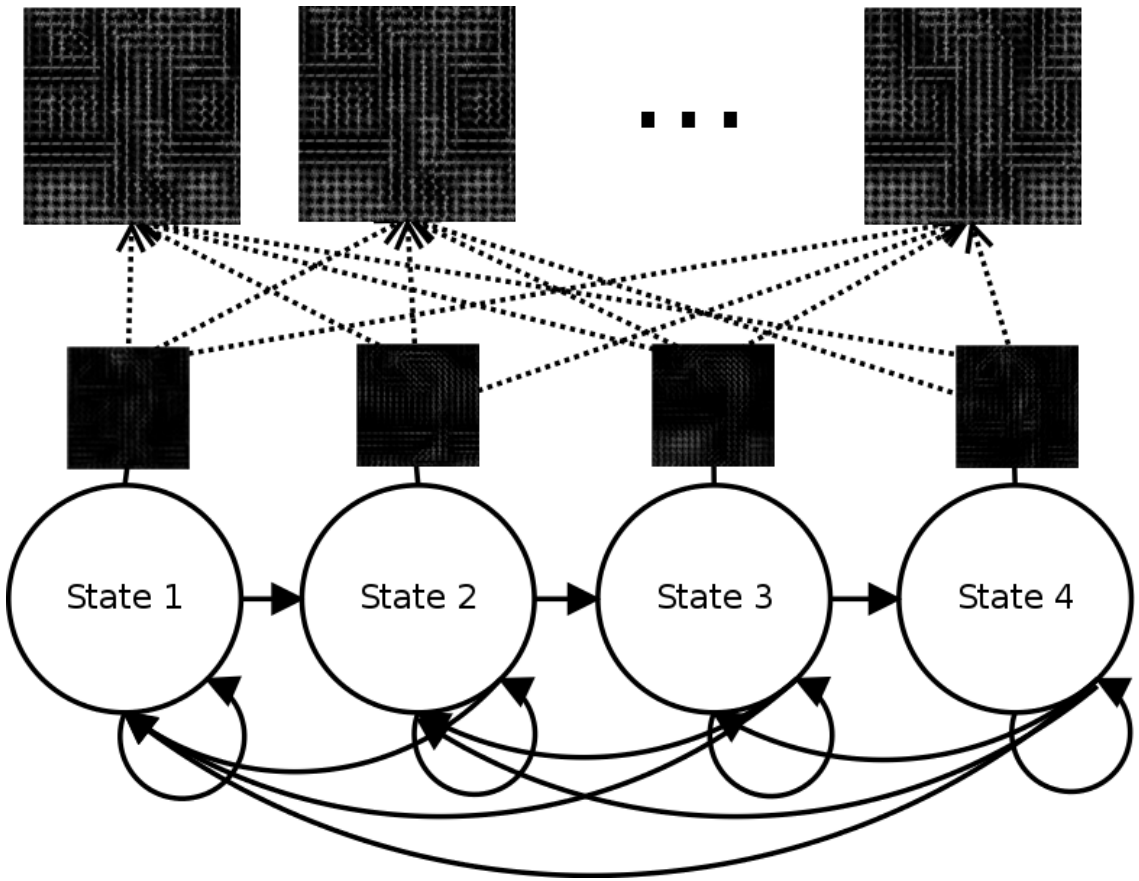
Hidden Markov Models (HMM) (Baum & Petrie 1966)



Likelihood of sequence computed with the Forward Algorithm
MAP computed with the Viterbi Algorithm

Representing the Sequence of an Action

Hidden Markov Models (HMM) (Baum & Petrie 1966)



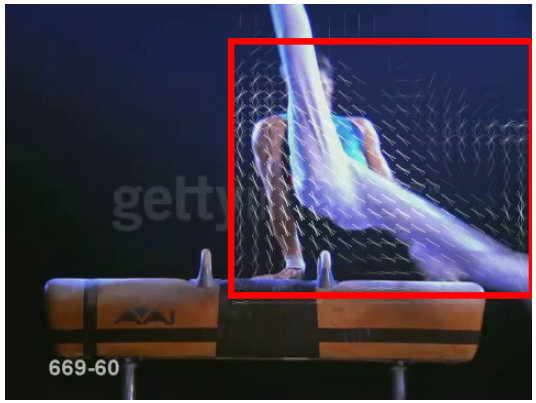
Modification to the Event tracker

Modification to the Event tracker

Use the HMM state output models as our object detectors.

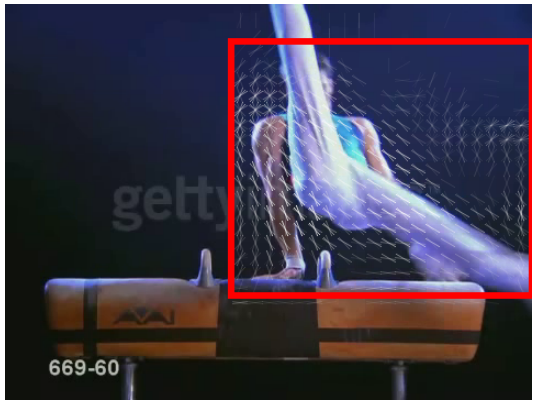
Modification to the Event tracker

Use the HMM state output models as our object detectors.
Detect unusual poses



Modification to the Event tracker

Use the HMM state output models as our object detectors.
Detect unusual poses



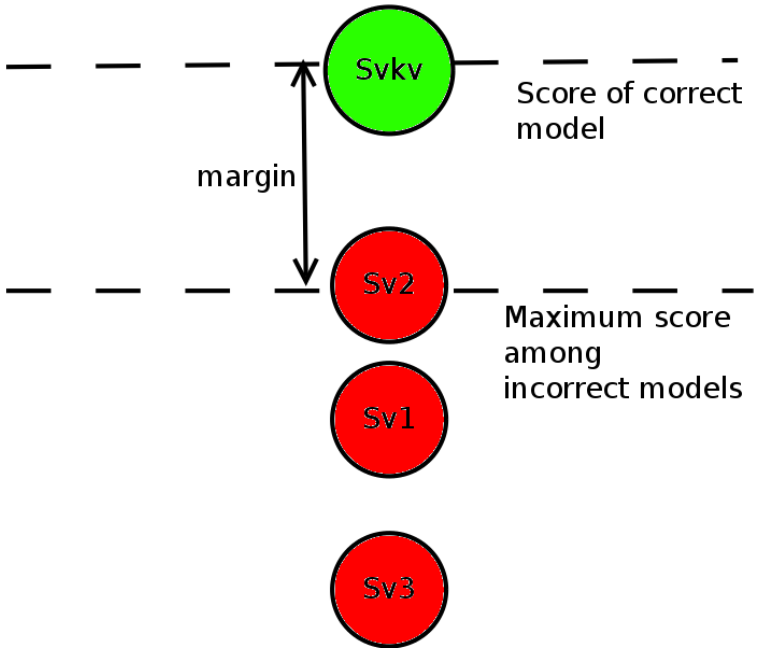
Detect non-human actions



Learning the Sequence of Appearance and Motion

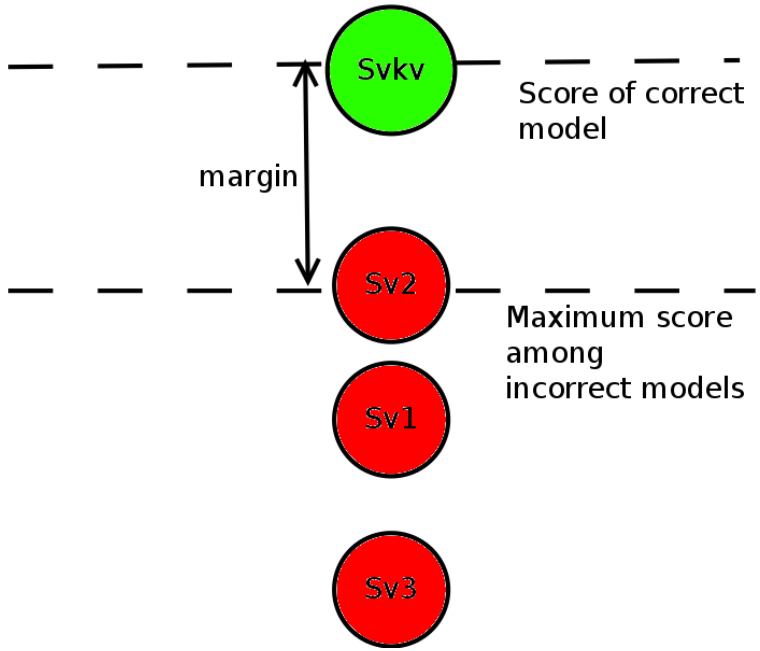
Learning the Sequence of Appearance and Motion

Maximize the classification margin on a set of training videos.



Learning the Sequence of Appearance and Motion

Maximize the classification margin on a set of training videos.

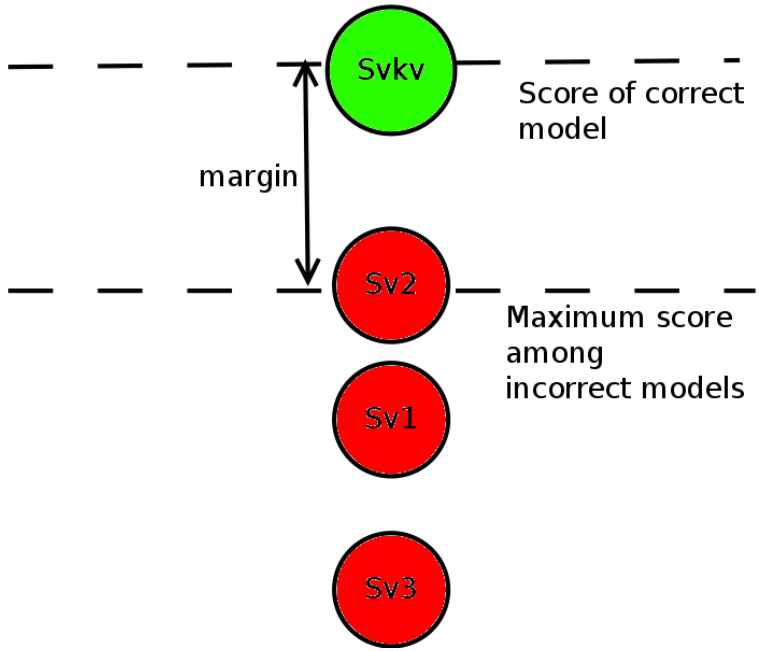


- Minimize a cost function O, the sum of individual video costs O_v.

$$O_v = \max(0, 1 + \textsc{SoftMax}(\{s_{vj} | j \neq k_v\}) - (s_{vk_v}))^2$$

Learning the Sequence of Appearance and Motion

Maximize the classification margin on a set of training videos.



- Minimize a cost function O, the sum of individual video costs O_v.

$$O_v = \max(0, 1 + \text{SOFTMAX}(\{s_{vj} | j \neq k_v\}) - (s_{vk_v}))^2$$

- Optimized via gradient descent.

Computing the Gradient

- To optimize, need $\frac{\partial O}{\partial x_{jl}}$ for each parameter l of each class j.

$$\frac{\partial O}{\partial x_{jl}} = \sum_{v} 2A\sqrt{O_v}\frac{\partial s_{vj}}{\partial x_{jl}}$$

where $A = -1$ if $j = k_v$, otherwise

$$A = \frac{s_{vj}\exp(s_{vj})}{\sum\limits_{\substack{i=1,\ldots,C \\ i \neq k_v}} \exp\left(\frac{s_{vi}}{t}\right)}$$

- Use techniques of Automatic Differentiation to compute $\frac{\partial s_{vj}}{\partial x_{jl}}$ through the HMM forward algorithm.
  - transition models
  - output models

Need for additional negative samples

Given a video with boxes around the actor:

return a label:

Need for additional negative samples

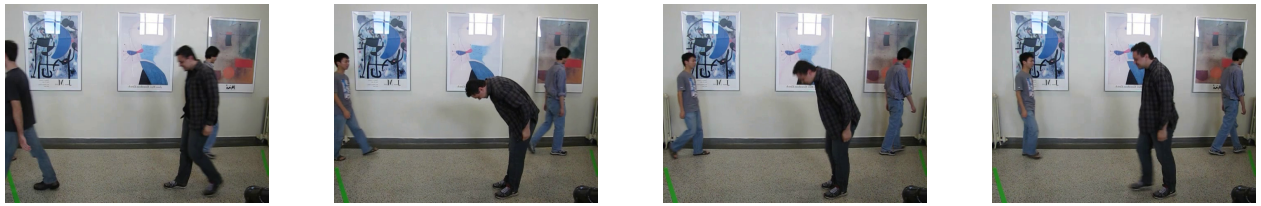
Given a video with boxes around the actor:

return a label: *bend*

Need for additional negative samples

Given a video with boxes around the actor:

return a label: *bend*
Given a video without boxes:



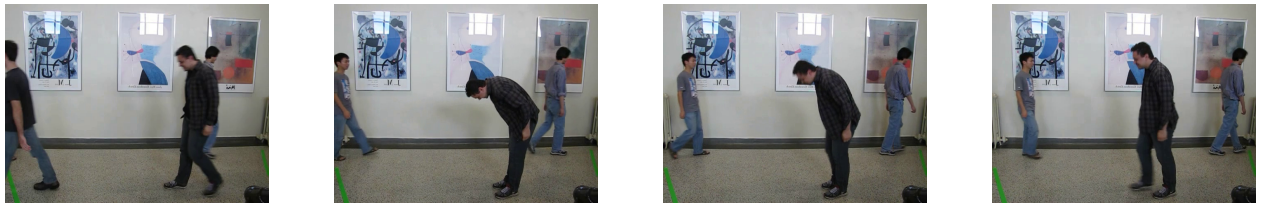
Return a set of boxes and a label:

Need for additional negative samples

Given a video with boxes around the actor:

return a label: *bend*

Given a video without boxes:

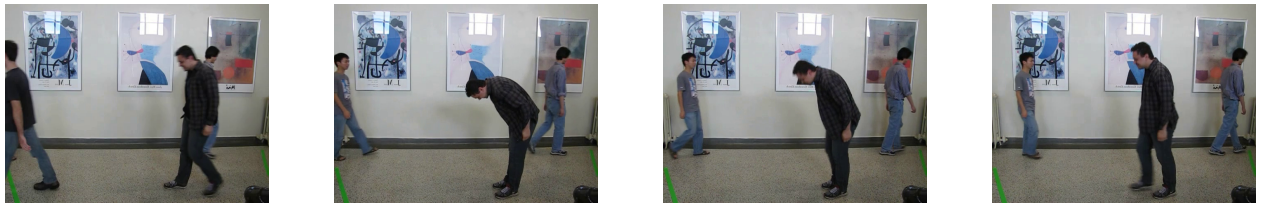


Return a set of boxes and a label: *kick*

Need for additional negative samples

Given a video with boxes around the actor:

return a label: *bend*
Given a video without boxes:



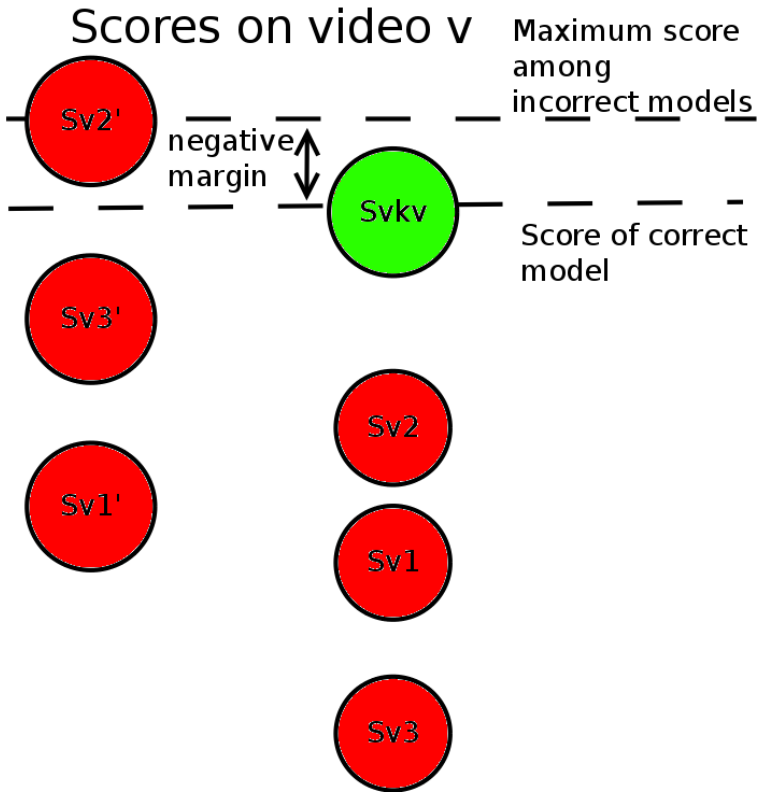
Return a set of boxes and a label: *kick*

Near chance performance!

Using tracker to produce additional negative samples

Obtain optimal tracks on training videos for all models.

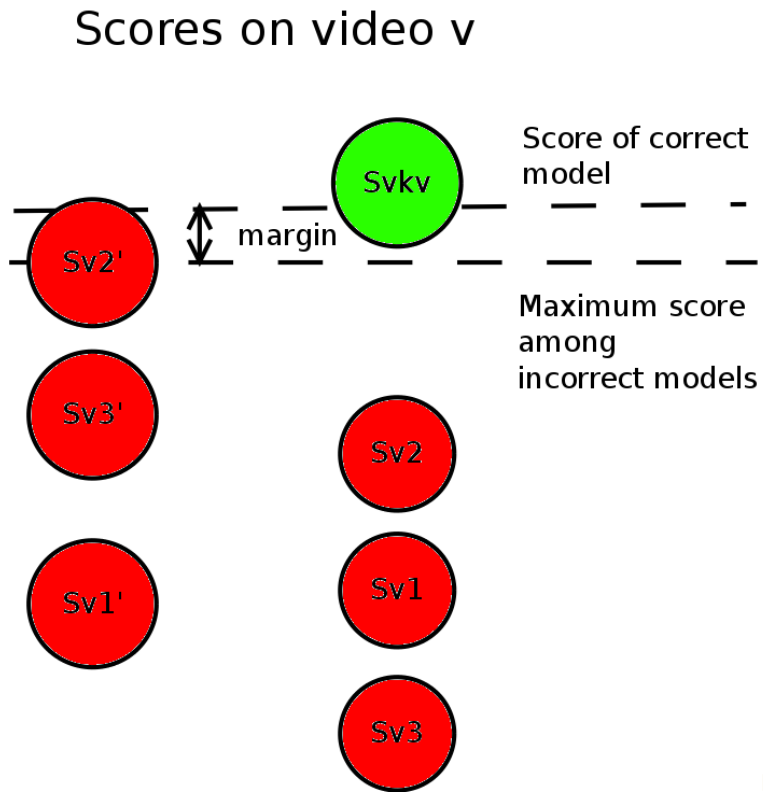
Add resulting tracks to training set.



Using tracker to produce additional negative samples

Obtain optimal tracks on training videos for all models.

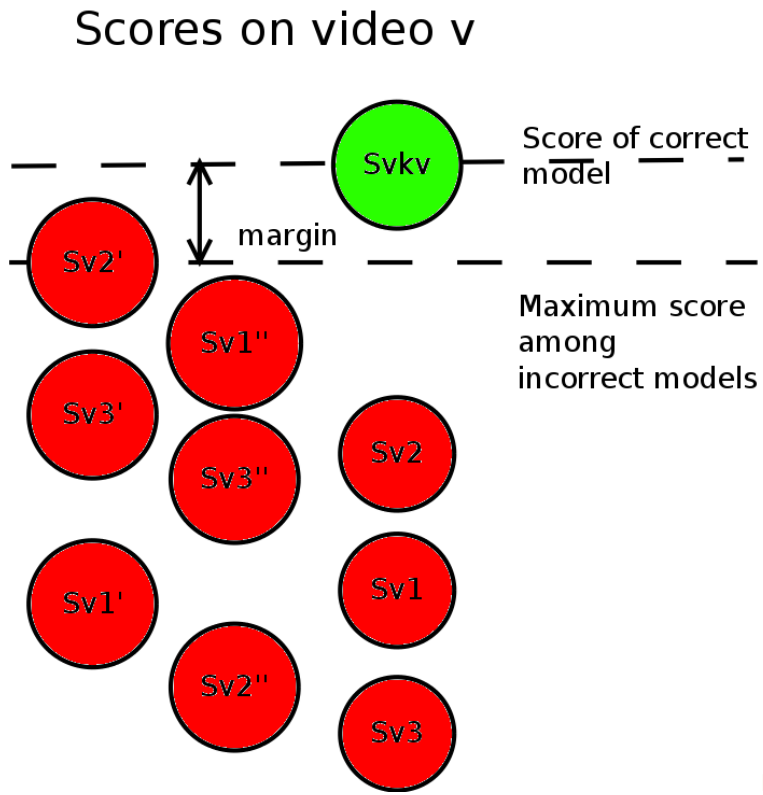
Add resulting tracks to training set.



Using tracker to produce additional negative samples

Obtain optimal tracks on training videos for all models.

Add resulting tracks to training set.



Examples

Examples