Introduction to Algorithmic Differentiation (AD)

First-Order Adjoint AD by Overloading ($F : \boldsymbol{R}^n \rightarrow \boldsymbol{R}^m$)

Uwe Naumann

Informatik 12:
Software and Tools for Computational Engineering (STCE)

RWTH Aachen

Contents

Outline

The first-order adjoint

$$x_{(1)} = y_{(1)} \cdot F' \equiv y_{(1)} \cdot \frac{dF}{dx}(\tilde{x})$$

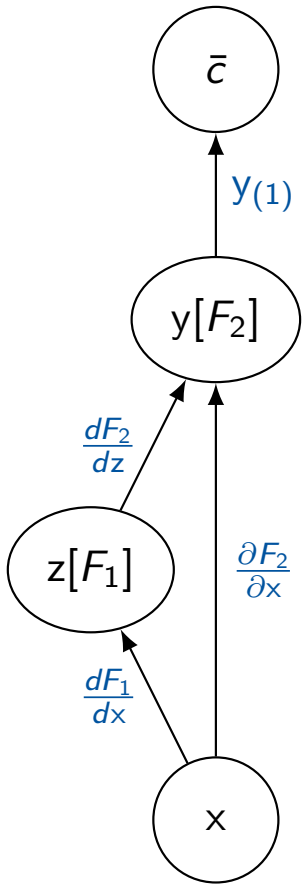
of $y = F(x)$ at $\tilde{x}$ in direction $y_{(1)}$ can be represented as the (total) derivative of a (output or adjoint) context $\bar{c} = \bar{c}(y(x)) \in \boldsymbol{R}$ with respect to x at $\tilde{x}$ such that

$$\frac{d\bar{c}}{dy} = y_{(1)} \in \boldsymbol{R}^{1 \times m} .$$

The chain rule yields

$$x_{(1)} \equiv \frac{d\bar{c}}{dx}(\tilde{x}) = \frac{d\bar{c}}{dy} \cdot \frac{dF}{dx}(\tilde{x}) = y_{(1)} \cdot \frac{dF}{dx}(\tilde{x}) .$$

The adjoint DAG is induced.



Adjoint DAG

Think of $\bar{c} = y_{(1)} \cdot y$.

The adjoint of F in direction $y_{(1)}$ is equal to the vector-matrix product

$$x_{(1)} \equiv \left(x_{(1)_i}\right)_{i=0,\ldots,n-1} = \left(\sum_{j=0}^{m-1} y_{(1)_j} \cdot F'_{j,i}\right)_{i=0,\ldots,n-1} = y_{(1)} \cdot F' \,.$$

Adjoints are row vectors, i.e, $x_{(1)} \in \boldsymbol{R}^{1\times n}$, $y_{(1)} \in \boldsymbol{R}^{1\times m}$.

The $\sum$ symbol is omitted in index notation yielding

$$x_{(1)_i} = y_{(1)_j} \cdot F'_{j,i}$$

and implying summation over the common (here j) index on the right-hand side. This expression is uniquely determined for known m and n.

Index notation will become particularly useful in the context of second- and higher-order tangents and adjoints.

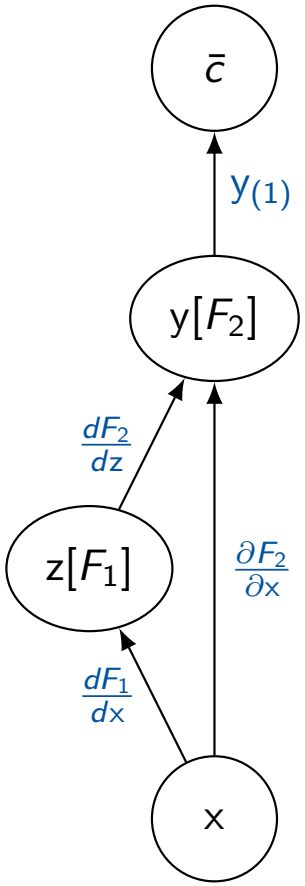
The DAG $G = G(\mathrm{x})$ of F induces a linear mapping (matrix-free (row) vector × Jacobian Product)

$$G_{(1)} : \boldsymbol{R}^m \times \boldsymbol{R}^n \to \boldsymbol{R}^m : \quad \mathrm{x}_{(1)} = \mathrm{y}_{(1)} \cdot G$$

defined by the chain rule applied to $\bar{c}(F(\mathrm{x}))$ at $\mathrm{x} = \tilde{\mathrm{x}}$.
This vector × DAG product is evaluated in adjoint AD as

$$v_{i(1)} = v_{i(1)} + v_{j(1)} \cdot \frac{d\varphi_j(v_k)_{k \prec j}}{dv_i}$$

for $j = n + q - 1, \ldots, n$ and $i \prec j$ and where $v_{i(1)} = x_{i(1)}$, $i = 0, \ldots, n-1$, and $y_{k(1)} = v_{n+p+k(1)}$, $k = 0, \ldots, m-1$.
Note potential dependencies among active results.



Adjoint DAG

Outline

First-order adjoints of

$$F : \boldsymbol{R}^n \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}^m \times \boldsymbol{R}^{m^*} : \quad (\mathrm{y}, \mathrm{y}^*) = F(\mathrm{x}, \mathrm{x}^*) \,,$$

can be evaluated in (scalar) adjoint [reverse] mode AD as

$$\mathrm{x}_{(1)} = \mathrm{x}_{(1)} + F_{(1)}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*, \mathrm{y}_{(1)}) \equiv \mathrm{x}_{(1)} + \mathrm{y}_{(1)} \cdot \frac{\partial F}{\partial \mathrm{x}}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*)$$

for active $\mathrm{y}_{(1)} \in \boldsymbol{R}^{1 \times m}$, $\mathrm{x}_{(1)} \in \boldsymbol{R}^{1 \times n}$ and passive $\mathrm{x}^*, \mathrm{y}^*$.

$\mathrm{x}_{(1)}$ can be nonzero due to prior adjoint computation with different activity patterns or as the result of uses of x outside of F. Incrementation of $\mathrm{x}_{(1)}$ ensures correctness according to the chain rule.

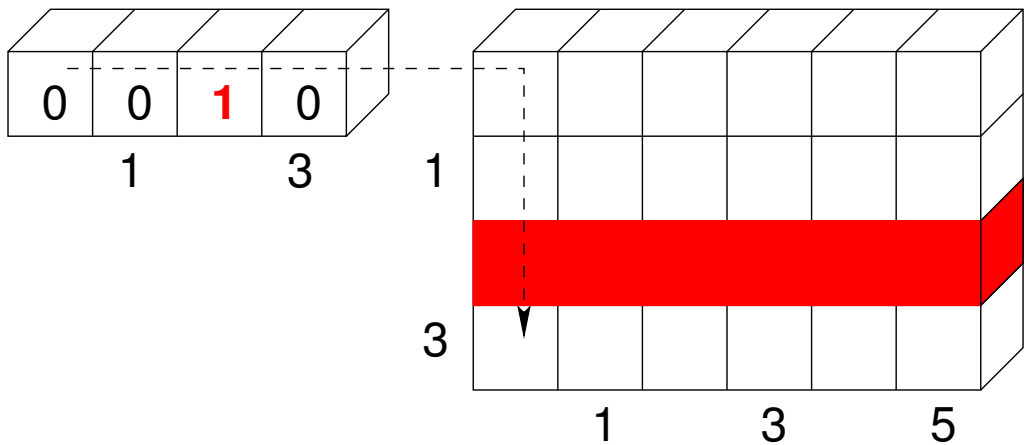
Typically, $\mathrm{x}_{(1)} = 0$ initially.

The partial Jacobian $F'_x \equiv \frac{\partial F}{\partial x}$ can be accumulated at $(\tilde{x}, \tilde{x}^*)$ by m evaluations of $F_{(1)}$ with $y_{(1)}$ ranging over the Cartesian basis vectors in $\mathbb{R}^m$.[1] Hence,

$$\text{Cost}(F'_x) = O(m) \cdot \text{Cost}(F_{(1)}) = O(m) \cdot \text{Cost}(F) \ .$$

The second identity follows immediately from the fact that adjoints are implemented as vector $\times$ DAG product.
If $n^* = m^* = 0$ then $y = F(x)$ and

$$\begin{aligned} x_{(1)} &= x_{(1)} + F_{(1)}(\tilde{x}, x^{(1)}) \\ &\equiv x_{(1)} + y_{(1)} \cdot \frac{dF}{dx}(\tilde{x}) \ . \end{aligned}$$



[1] This number can be decreased by detecting and exploiting potential sparsity of F'_x.

Augmented primal (forward section)

$$
\begin{aligned}
i = 0, \ldots, n-1 : &\qquad v_i = x_i \\
j = n, \ldots, n+q-1 : &\qquad v_j = \varphi_j(v_k)_{k \prec j}; \quad d_{j,i} = \frac{d\varphi_j(v_k)_{k \prec j}}{dv_i} \quad \forall i \prec j \\
k = 0, \ldots, m-1 : &\qquad y_k = v_{n+p+k}
\end{aligned}
$$

Adjoint (reverse section)

$$
\begin{aligned}
j = 0, \ldots, n+q-1 : &\qquad v_{j_{(1)}} = 0 \\
k = 0, \ldots, m-1 : &\qquad v_{n+p+k_{(1)}} = y_{k_{(1)}} \\
j = n+q-1, \ldots, n : &\qquad v_{i_{(1)}} = v_{i_{(1)}} + v_{j_{(1)}} \cdot d_{j,i} \quad \forall i \prec j \\
i = 0, \ldots, n-1 : &\qquad x_{i_{(1)}} = x_{i_{(1)}} + v_{i_{(1)}}
\end{aligned}
$$

The augmented primal lends itself for implementation by overloading. The primal is extended by a tape containing all information necessary for propagation of adjoints. The latter are computed by interpretation of the tape.

Adjoint AD

Example: $y = f(x) = e^{\sin\left(x_0^2 + x_1^2\right)}$

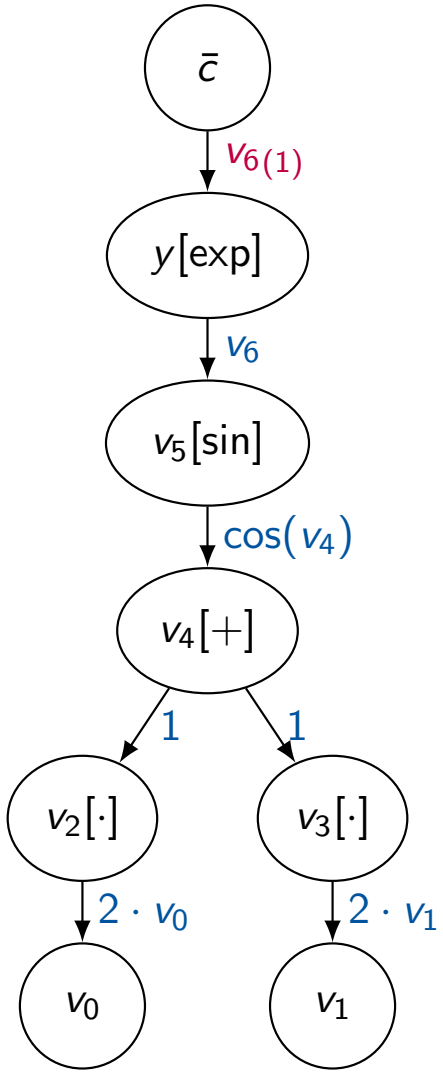
Seeding $v_{6(1)} = y_{(1)} = 1$ and running the primal SAC

$$v_2 = v_0^2;\ \ v_3 = v_1^2;\ \ v_4 = v_2 + v_3;\ \ v_5 = \sin(v_4);\ \ v_6 = e^{v_5}$$

followed by the adjoint SAC

$$v_{5(1)} \mathrel{+}= v_6 \cdot v_{6(1)}$$
$$v_{4(1)} \mathrel{+}= \cos(v_4) \cdot v_{5(1)}$$
$$v_{3(1)} \mathrel{+}= v_{4(1)}$$
$$v_{2(1)} \mathrel{+}= v_{4(1)}$$
$$v_{1(1)} \mathrel{+}= 2 \cdot v_1 \cdot v_{3(1)}$$
$$v_{0(1)} \mathrel{+}= 2 \cdot v_0 \cdot v_{2(1)}$$

yields the corresponding row of the Jacobian (here the entire gradient) in $x_{(1)} = \left(x_{0(1)}, x_{1(1)}\right) = \left(v_{0(1)}, v_{1(1)}\right)$.



Wake Up!

Derive adjoint SAC and adjoint DAG for the product reduction

$$y = \prod_{i=0}^{n-1} x_i$$

for $n = 3$.

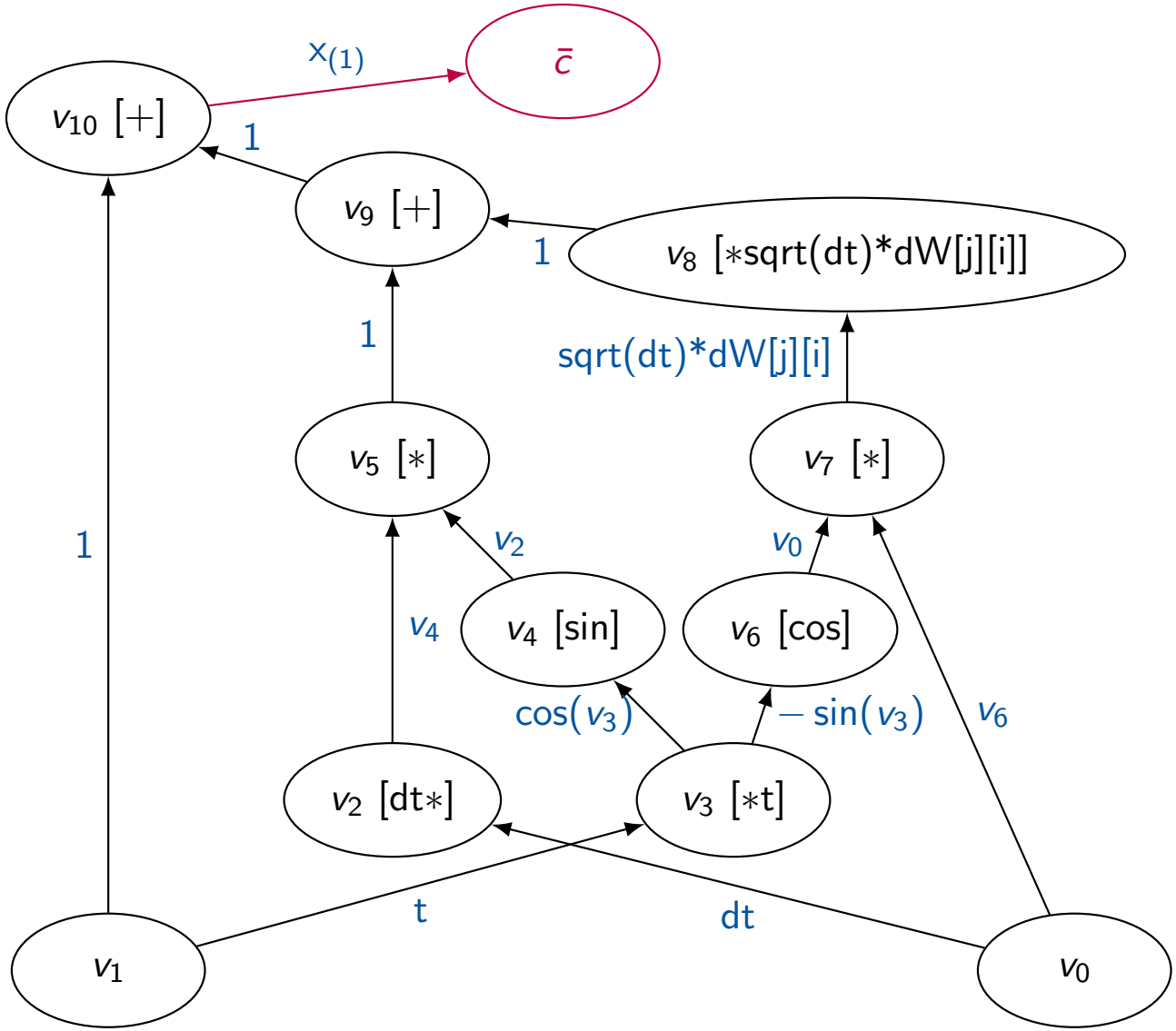
Hello AD World: Adjoint SAC

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];

$$
\begin{aligned}
v_0 &= \text{p[i]} & & & \text{p[i]}_{(1)} &\mathrel{+}= v_{0(1)} \\
v_1 &= \text{x} & & \uparrow & \text{x}_{(1)} &\mathrel{+}= v_{1(1)} \\
v_2 &= \text{dt} * v_0 & & & v_{0(1)} &\mathrel{+}= \text{dt} * v_{2(1)} \\
v_3 &= v_1 * \text{t} & & \uparrow & v_{1(1)} &\mathrel{+}= v_{3(1)} * \text{t} \\
v_4 &= \sin(v_3) & & & v_{3(1)} &\mathrel{+}= \cos(v_3) * v_{4(1)} \\
v_5 &= v_2 * v_4 & & \uparrow & v_{4(1)} &\mathrel{+}= v_{5(1)} * v_2;\ v_{2(1)} \mathrel{+}= v_{5(1)} * v_4 \\
v_6 &= \cos(v_3) & & & v_{3(1)} &\mathrel{+}= -\sin(v_3) * v_{6(1)} \\
v_7 &= v_0 * v_6 & & \uparrow & v_{0(1)} &\mathrel{+}= v_{7(1)} * v_6;\ v_{6(1)} \mathrel{+}= v_0 * v_{7(1)} \\
v_8 &= v_7 * \text{sqrt(dt)*dW[j][i]} & & & v_{7(1)} &\mathrel{+}= v_{8(1)} * \text{sqrt(dt)*dW[j][i]} \\
v_9 &= v_5 + v_8 & & \uparrow & v_{5(1)} &\mathrel{+}= v_{9(1)};\ v_{8(1)} \mathrel{+}= v_{9(1)} \\
v_{10} &= v_1 + v_9 & & & v_{1(1)} &\mathrel{+}= v_{10(1)};\ v_{9(1)} \mathrel{+}= v_{10(1)} \\
\text{x} &= v_{10} & & \uparrow & v_{10(1)} &\mathrel{+}= \text{x}_{(1)} \\
& & & & v_{i(1)} &= 0\ (i = 0, \ldots, 10)
\end{aligned}
$$

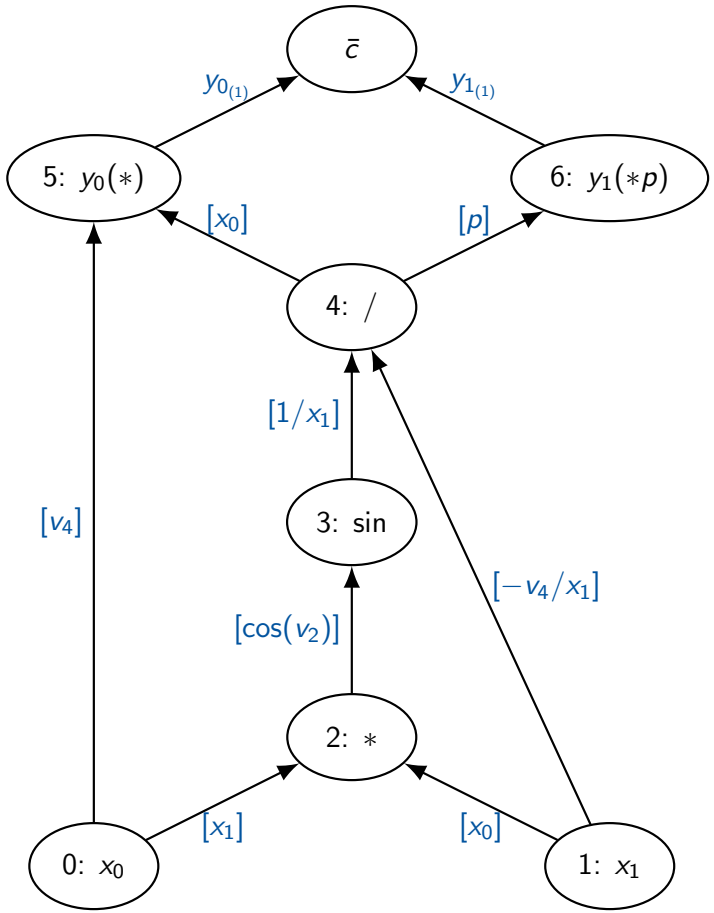
Hello AD World: Adjoint DAG

x+=dt*p[i]*sin(x*t)+p[i]*cos(x*t)*sqrt(dt)*dW[j][i];



Adjoint AD by Overloading (Movie)

Chain Rule on Adjoint DAG (Tape)



Adjoint DAG

We consider

$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} x_0 * \sin(x_0 * x_1)/x_1 \\ \sin(x_0 * x_1)/x_1 * p \end{pmatrix}$$

implemented as

$$t = \sin(x_0 * x_1)/x_1$$
$$y_0 = x_0 * t$$
$$y_1 = t * p$$

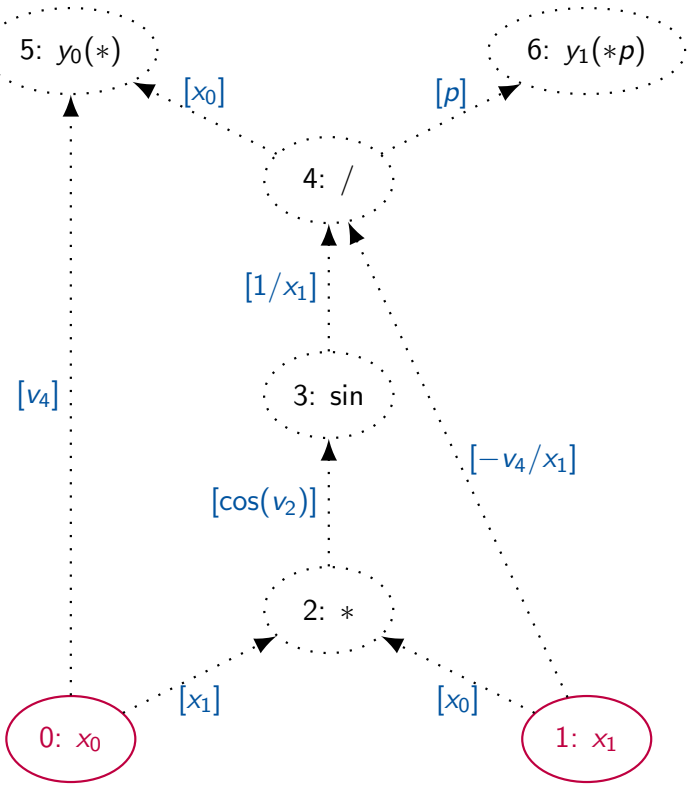
yielding SAC

$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_3/x_1$$
$$y_0 = x_0 * v_4$$
$$y_1 = v_4 * p$$

for some passive value p.

Adjoint AD by Overloading (Movie)

Register (Independent Inputs with Tape)

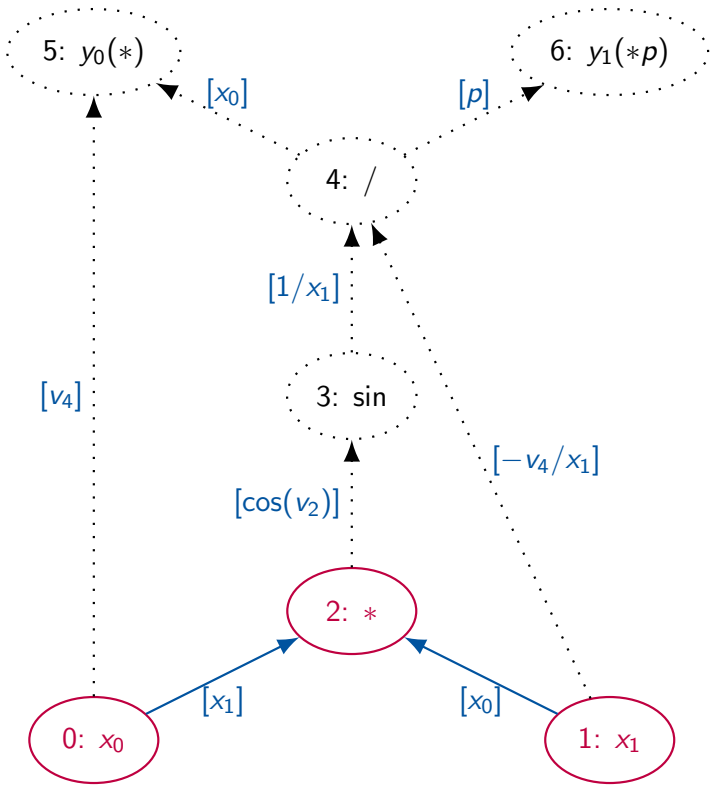


$x_0 = ?$

$x_1 = ?$

Adjoint AD by Overloading (Movie)

Record (Tape)

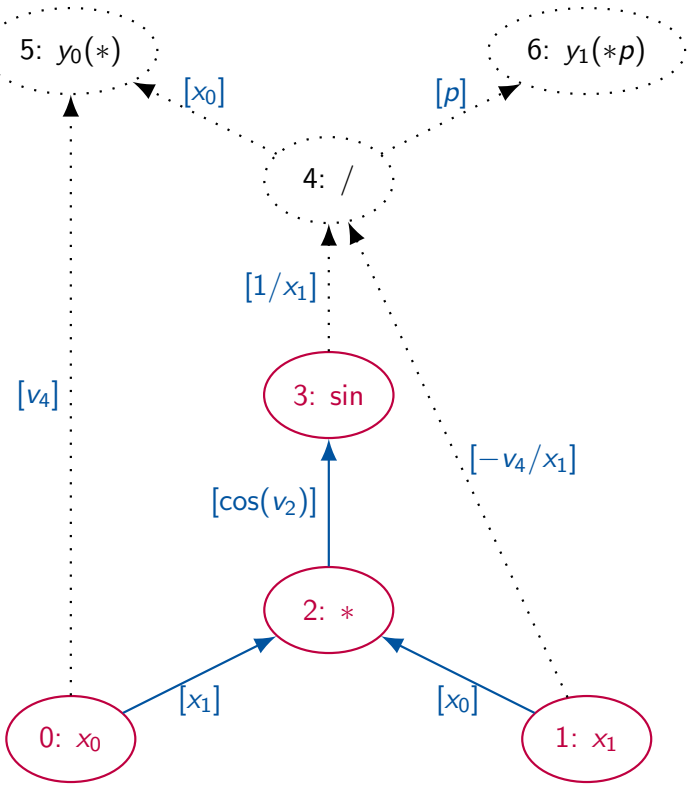


// overloaded $*$

$v_2 = x_0 * x_1$

Adjoint AD by Overloading (Movie)

Record (Tape)



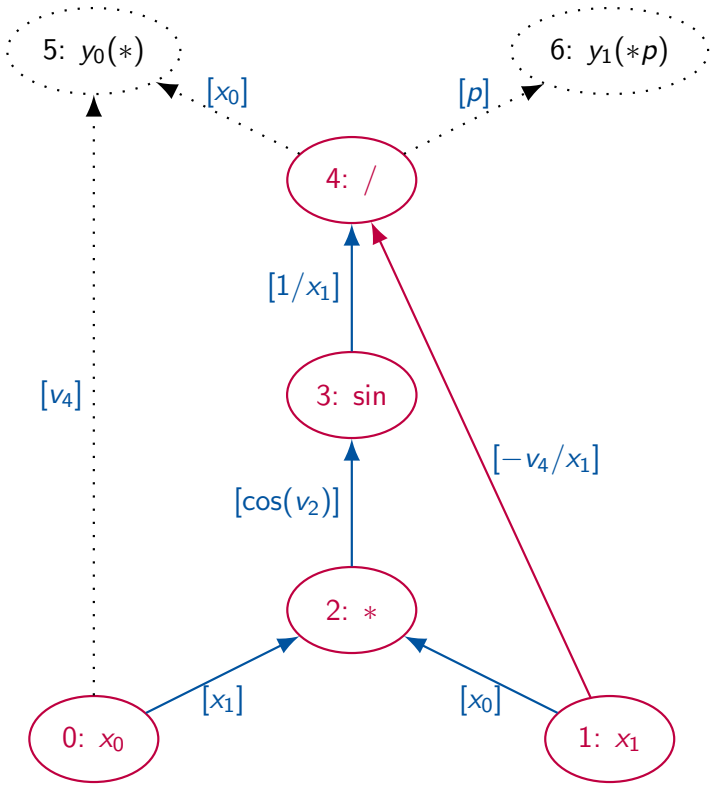
$v_2 = x_0 * x_1$

// overloaded sin

$v_3 = \sin(v_2)$

Adjoint AD by Overloading (Movie)

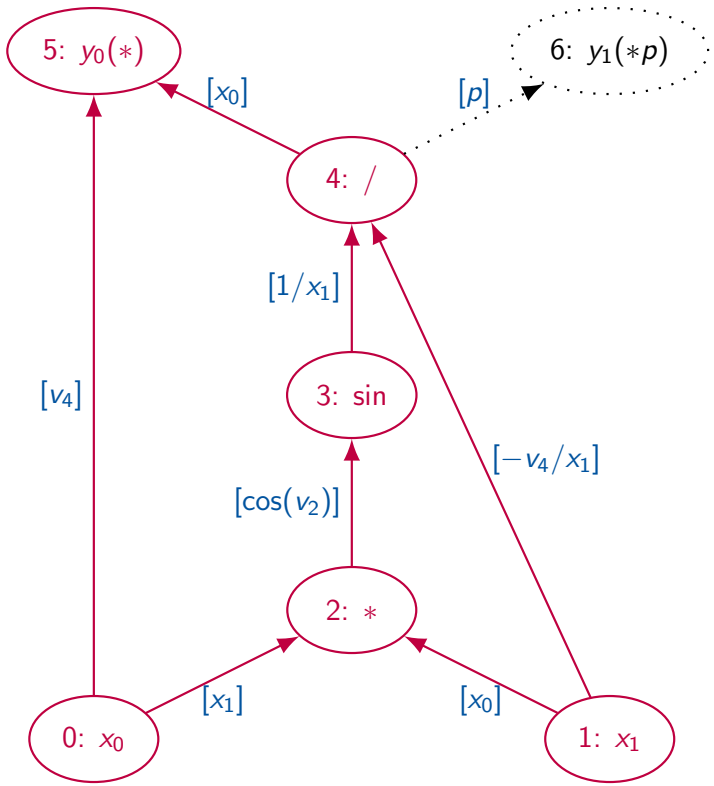
Record (Tape)



$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
// overloaded /
$$v_4 = v_3 / x_1$$

Adjoint AD by Overloading (Movie)

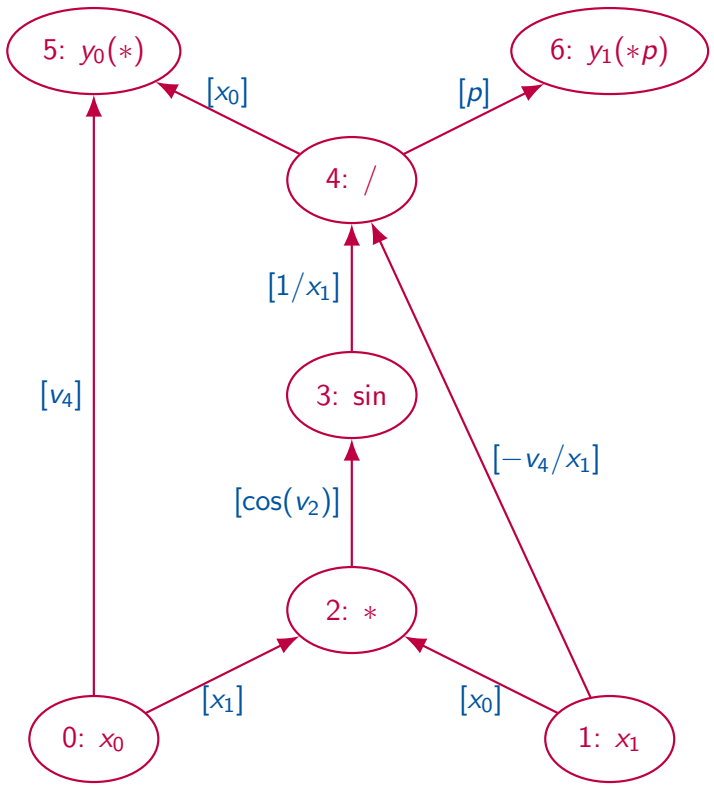
Record (Tape)



$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_3 / x_1$$
// overloaded *
$$y_0 = x_0 * v_4$$

Adjoint AD by Overloading (Movie)

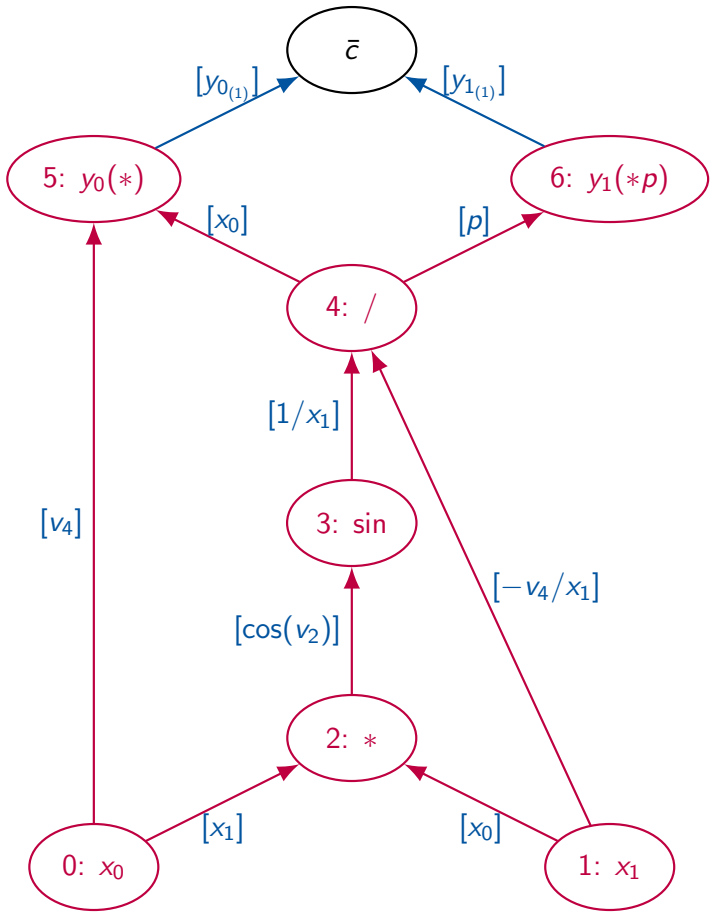
Record (Tape)



$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_3 / x_1$$
$$y_0 = x_0 * v_4$$
// overloaded $*$
$$y_1 = v_4 * p$$

Adjoint AD by Overloading (Movie)

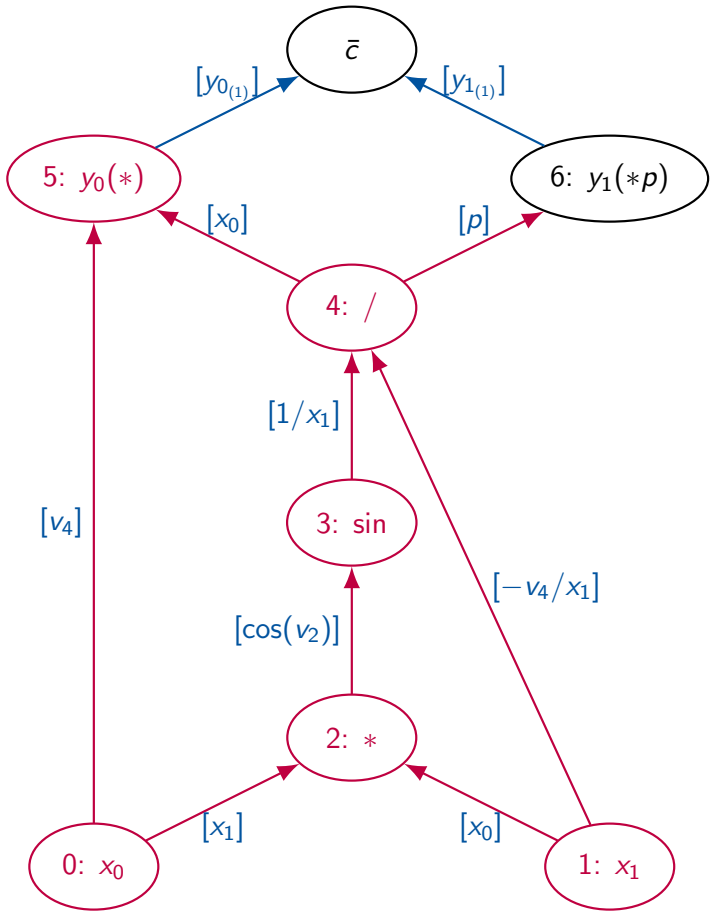
Seed



$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_3/x_1$$
$$y_0 = x_0 * v_4$$
$$y_1 = v_4 * p$$
$$y0_{(1)} = ?$$
$$y1_{(1)} = ?$$
$$x0_{(1)} = ?$$
$$x1_{(1)} = ?$$
$$v2_{(1)} = 0$$
$$v3_{(1)} = 0$$
$$v4_{(1)} = 0$$

Adjoint AD by Overloading (Movie)

Interpret (Tape)



$$v_2 = x_0 * x_1$$
$$v_3 = \sin(v_2)$$
$$v_4 = v_3 / x_1$$
$$y_0 = x_0 * v_4$$
$$y_1 = v_4 * p$$
$$v_{4_{(1)}} \mathrel{+}= p * y_{1_{(1)}}$$

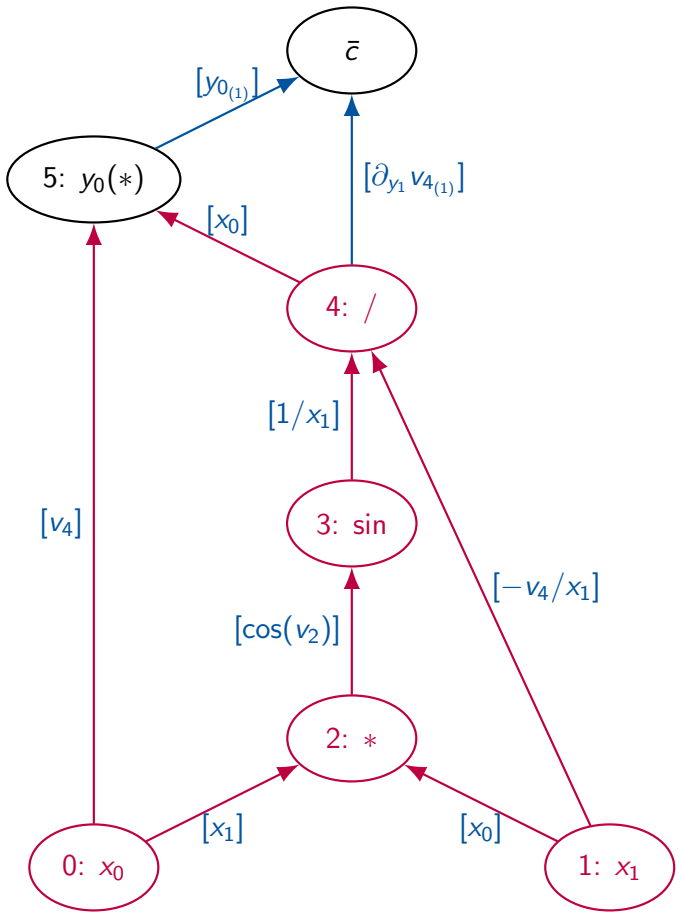
$$v_{4_{(1)}} \mathrel{+}= p * y_{1_{(1)}}$$

$$\Leftrightarrow$$

$$v_{4_{(1)}} = v_{4_{(1)}} + p * y_{1_{(1)}}.$$

Adjoint AD by Overloading (Movie)

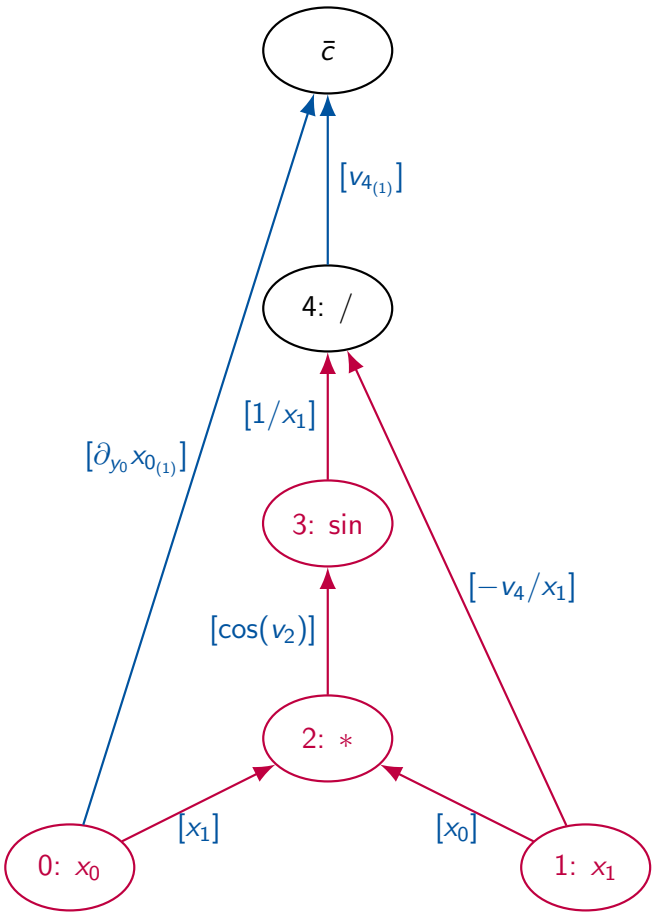
Interpret (Tape)



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_3 &= \sin(v_2) \\
v_4 &= v_3 / x_1 \\
y_0 &= x_0 * v_4 \\
y_1 &= v_4 * p \\
v_{4_{(1)}} &\mathrel{+}= p * y_{1_{(1)}} \\
v_{4_{(1)}} &\mathrel{+}= x_0 * y_{0_{(1)}} \\
x_{0_{(1)}} &\mathrel{+}= v_4 * y_{0_{(1)}}
\end{aligned}
$$

Adjoint AD by Overloading (Movie)

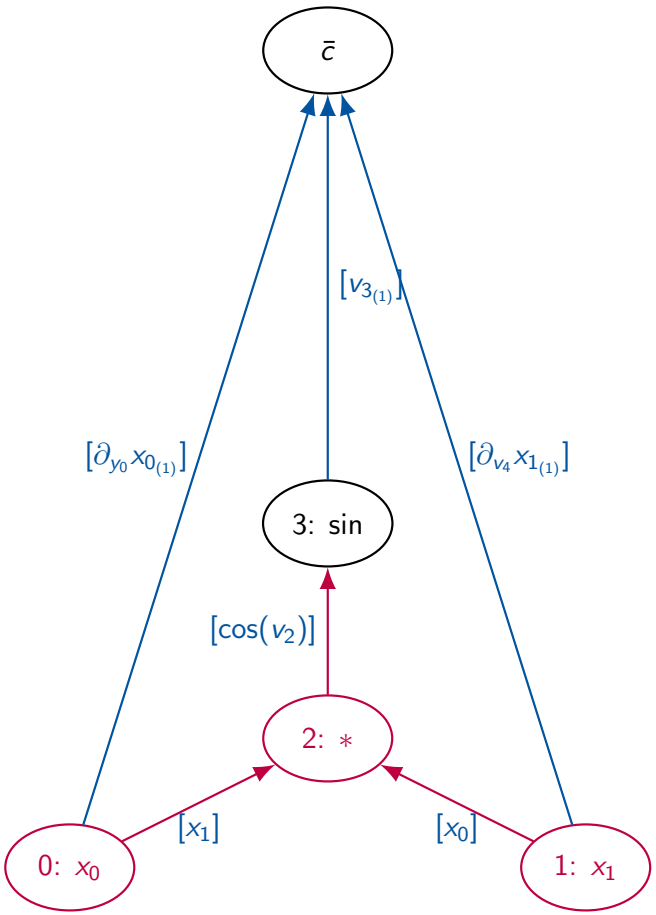
Interpret (Tape)



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_3 &= \sin(v_2) \\
v_4 &= v_3 / x_1 \\
y_0 &= x_0 * v_4 \\
y_1 &= v_4 * p \\
v_{4_{(1)}} &\mathrel{+}= p * y_{1_{(1)}} \\
v_{4_{(1)}} &\mathrel{+}= x_0 * y_{0_{(1)}} \\
x_{0_{(1)}} &\mathrel{+}= v_4 * y_{0_{(1)}} \\
u &= 1/x_1 \\
v_{3_{(1)}} &\mathrel{+}= u * v_{4_{(1)}} \\
x_{1_{(1)}} &\mathrel{-}= v_4 * u * v_{4_{(1)}}
\end{aligned}
$$

Adjoint AD by Overloading (Movie)

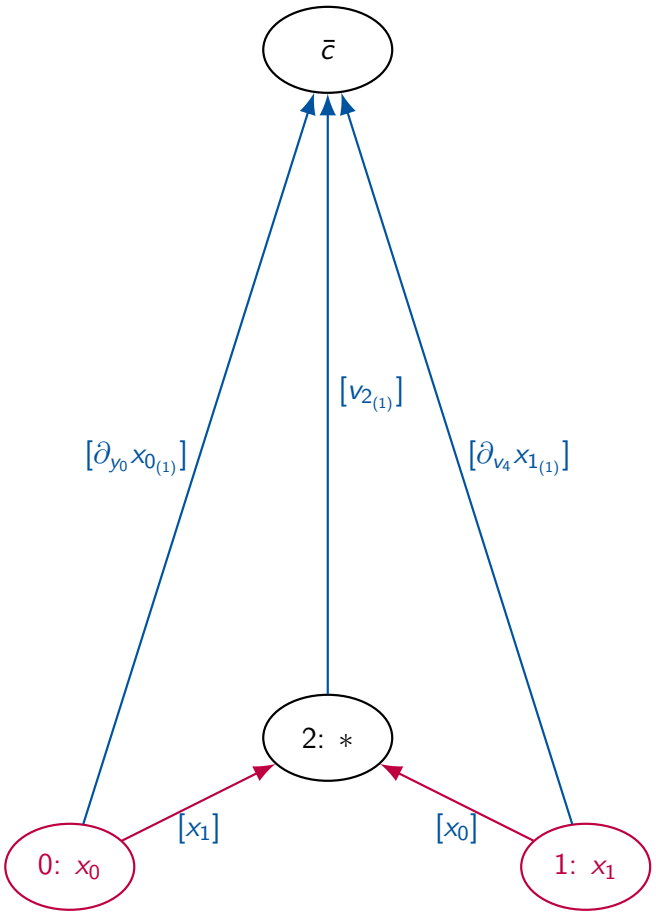
Interpret (Tape)



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_3 &= \sin(v_2) \\
v_4 &= v_3 / x_1 \\
y_0 &= x_0 * v_4 \\
y_1 &= v_4 * p \\
v_{4_{(1)}} &\mathrel{+}= p * y_{1_{(1)}} \\
v_{4_{(1)}} &\mathrel{+}= x_0 * y_{0_{(1)}} \\
x_{0_{(1)}} &\mathrel{+}= v_4 * y_{0_{(1)}} \\
u &= 1 / x_1 \\
v_{3_{(1)}} &\mathrel{+}= u * v_{4_{(1)}} \\
x_{1_{(1)}} &\mathrel{-}= v_4 * u * v_{4_{(1)}} \\
v_{2_{(1)}} &\mathrel{+}= \cos(x_2) * v_{3_{(1)}}
\end{aligned}
$$

Adjoint AD by Overloading (Movie)

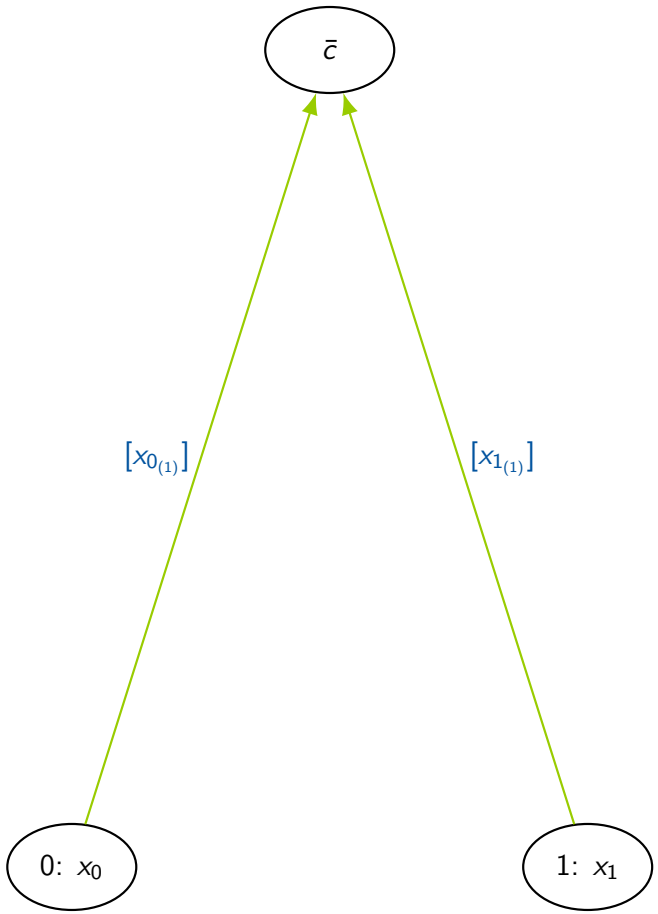
Interpret (Tape)



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_3 &= \sin(v_2) \\
v_4 &= v_3 / x_1 \\
y_0 &= x_0 * v_4 \\
y_1 &= v_4 * p \\
v_{4_{(1)}} &\mathrel{+}= p * y_{1_{(1)}} \\
v_{4_{(1)}} &\mathrel{+}= x_0 * y_{0_{(1)}} \\
x_{0_{(1)}} &\mathrel{+}= v_4 * y_{0_{(1)}} \\
u &= 1 / x_1 \\
v_{3_{(1)}} &\mathrel{+}= u * v_{4_{(1)}} \\
x_{1_{(1)}} &\mathrel{-}= v_4 * u * v_{4_{(1)}} \\
v_{2_{(1)}} &\mathrel{+}= \cos(x_2) * v_{3_{(1)}} \\
x_{0_{(1)}} &\mathrel{+}= x_1 * v_{2_{(1)}} \\
x_{1_{(1)}} &\mathrel{+}= x_0 * v_{2_{(1)}}
\end{aligned}
$$

Adjoint AD by Overloading (Movie)

Harvest



$$
\begin{aligned}
v_2 &= x_0 * x_1 \\
v_3 &= \sin(v_2) \\
v_4 &= v_3 / x_1 \\
y_0 &= x_0 * v_4 \\
y_1 &= v_4 * p \\
v_{4_{(1)}} &+= p * y_{1_{(1)}} \\
v_{4_{(1)}} &+= x_0 * y_{0_{(1)}} \\
x_{0_{(1)}} &+= v_4 * y_{0_{(1)}} \\
u &= 1 / x_1 \\
v_{3_{(1)}} &+= u * v_{4_{(1)}} \\
x_{1_{(1)}} &-= v_4 * u * v_{4_{(1)}} \\
v_{2_{(1)}} &+= \cos(x_2) * v_{3_{(1)}} \\
x_{0_{(1)}} &+= x_1 * v_{2_{(1)}} \\
x_{1_{(1)}} &+= x_0 * v_{2_{(1)}}
\end{aligned}
$$

```
1  template<typename T, typename PT>
2  void f(size_t ncp, size_t ncs, const PT &eps, T& x, const std::vector<T>& p, const
       std::vector<std::vector<PT>>& dW) {
3    newton(x,p[0],eps);
4    T s=0;
5    size_t m=dW.size();
6    for (size_t j=0;j<m;j+=ncp) paths(ncs,j,j+ncp,x,p,dW,s);
7    x=s/m;
8  }
```

We are looking for the gradient of the final `x` with respect to the read-only parameters `p`.

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9
10
11
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10
11
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14   x_v=dco::value(x); // get primal function value
15
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14   x_v=dco::value(x); // get primal function value
15   DCO_M::global_tape->register_output_variable(x); // record active output
16
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14   x_v=dco::value(x); // get primal function value
15   DCO_M::global_tape->register_output_variable(x); // record active output
16   dco::derivative(x)=1; // seed
17
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14   x_v=dco::value(x); // get primal function value
15   DCO_M::global_tape->register_output_variable(x); // record active output
16   dco::derivative(x)=1; // seed
17   DCO_M::global_tape->interpret_adjoint(); // interpret tape
18
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14   x_v=dco::value(x); // get primal function value
15   DCO_M::global_tape->register_output_variable(x); // record active output
16   dco::derivative(x)=1; // seed
17   DCO_M::global_tape->interpret_adjoint(); // interpret tape
18   for (size_t i=0;i<n;i++) dxdp[i]=dco::derivative(p[i]); // harvest
19
20 }
```

Adjoint AD

dco/c++


```
1  #include "f.hpp" // primal source
2  #include "dco.hpp" // dco/c++
3
4  template<typename T, typename PT> // adjoint driver
5  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
6    using DCO_M=typename dco::ga1s<T>; // adjoint mode
7    using DCO_T=typename DCO_M::type; // adjoint type
8    using DCO_TT=typename DCO_M::tape_t; // tape type
9    size_t n=p_v.size(); // size of gradient
10   DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
11   DCO_M::global_tape=DCO_TT::create(); // "touch" tape
12   DCO_M::global_tape->register_variable(p); // record active inputs
13   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
14   x_v=dco::value(x); // get primal function value
15   DCO_M::global_tape->register_output_variable(x); // record active output
16   dco::derivative(x)=1; // seed
17   DCO_M::global_tape->interpret_adjoint(); // interpret tape
18   for (size_t i=0;i<n;i++) dxdp[i]=dco::derivative(p[i]); // harvest
19   DCO_TT::remove(DCO_M::global_tape); // deallocate tape
20 }
```

- read-only active input: for access to correct input adjoints
- blob tape: default; grabs nearly all available memory on Windows unless specified otherwise; see below
- chunk tape: tape grows dynamically; see below
- multiple tapes: `dco::ga1sm<T>` allows for multiple tapes to be created; potential inter-tape communication manually
- multiple vectors of adjoints: multithreaded interpretation of shared tape
- file tape: extends available tape memory to disc
- modulo tape: reduces size of vector of adjoints

Adjoint AD with dco/c++

Blob Tape

```
1    using DCO_M=typename dco::ga1s<T>;
2    using DCO_TT=typename DCO_M::tape_t;
3    DCO_T x=x_v,y;
4
5    dco::tape_options o;
6    // grabs better part of mem available unless specified
7    o.set_blob_size_in_mbyte(16);
8    DCO_M::global_tape=DCO_TT::create(o);
```

Adjoint AD with dco/c++

Chunk Tape

Requires `$(CPPC) ... -DDCO_CHUNK_TAPE`

```
1  ...
2    using DCO_M=typename dco::ga1s<T>;
3    using DCO_TT=typename DCO_M::tape_t;
4
5    dco::tape_options o;
6    // default chunk size is 128mb
7    o.set_chunk_size_in_mbyte(4);
8    DCO_M::global_tape=DCO_TT::create(o);
9  ...
```

Vectors (also: ensembles) of k first-order adjoints of

$$F : \boldsymbol{R}^n \times \boldsymbol{R}^{n^*} \to \boldsymbol{R}^m \times \boldsymbol{R}^{m^*} : \quad (\mathrm{y}, \mathrm{y}^*) = F(\mathrm{x}, \mathrm{x}^*) \ ,$$

can be evaluated in vector adjoint AD mode with machine accuracy as

$$X_{(1)} = X_{(1)} + F_{(1)}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*, Y_{(1)}) \equiv X_{(1)} + Y_{(1)} \cdot \frac{\partial F}{\partial \mathrm{x}}(\tilde{\mathrm{x}}, \tilde{\mathrm{x}}^*)$$

for $Y_{(1)} \in \boldsymbol{R}^{k \times m}$ and $X_{(1)} \in \boldsymbol{R}^{k \times n}$.

Performance of vector adjoint mode typically exceeds that of scalar adjoint mode even without the highly desirable explicit parallelization / vectorization. The memory requirement is increased.

Vector Adjoint AD

dco/c++


```
1  template<int N=1, typename T, typename PT> // vector adjoint driver
2  void gradient(size_t ncp, size_t ncs, const PT &eps, T& x_v, std::vector<T>& p_v,
       const std::vector<std::vector<PT>> &dW, std::vector<T> &dxdp) {
3    using DCO_M=typename dco::ga1v<T,N>; // adjoint mode
4    using DCO_T=typename DCO_M::type; // adjoint type
5    using DCO_TT=typename DCO_M::tape_t; // tape type
6    size_t n=p_v.size(); // size of gradient
7    DCO_T x=x_v; std::vector<DCO_T> p(n); dco::value(p)=p_v; // activate
8    DCO_M::global_tape=DCO_TT::create(); // "touch" tape
9    DCO_M::global_tape->register_variable(p); // record active inputs
10   f(ncp,ncs,eps,x,p,dW); // compute augmented primal
11   x_v=dco::value(x); // get primal function value
12   DCO_M::global_tape->register_output_variable(x); // record active output
13   dco::derivative(x)[0]=1; // seed
14   DCO_M::global_tape->interpret_adjoint(); // interpret tape
15   for (size_t i=0;i<n;i++) dxdp[i]=dco::derivative(p[i])[0]; // harvest
16   DCO_TT::remove(DCO_M::global_tape); // deallocate tape
17 }
```

Outline

$$x_{(1)} \cdot x^{(1)} = (y_{(1)} \cdot F'(x)) \cdot x^{(1)} = y_{(1)} \cdot (F'(x) \cdot x^{(1)}) = y_{(1)} \cdot y^{(1)}$$

Note use of differential invariant for

- validation of adjoints against tangents
- validation of adjoints against approximate tangents
- debugging of derivative code.

Wake Up!

Illustrate the first-order differential invariant for the product reduction

$$y = \prod_{i=0}^{n-1} x_i$$

for $n = 3$.

Outline

Let $y = F(x) = F_2(F_1(x))$ with $F_1 : \boldsymbol{R}^n \to \boldsymbol{R}^k$ and $F_2 : \boldsymbol{R}^k \to \boldsymbol{R}^m$ with DAGs G_1 and G_2, respectively. There are eight alternatives for computing F':

- $F' = G_2 \cdot F_1' = G_2 \cdot (G_1 \cdot I_n)$ (homogeneous tangent)
- $F' = G_2 \cdot F_1' = G_2 \cdot (I_k \cdot G_1)$
- $F' = F_2' \cdot G_1 = (I_m \cdot G_2) \cdot G_1$ (homogeneous adjoint)
- $F' = F_2' \cdot G_1 = (G_2 \cdot I_k) \cdot G_1$
- $F' = F_2' \cdot F_1' = (G_2 \cdot I_k) \cdot (I_k \cdot G_1)$
- $F' = F_2' \cdot F_1' = (I_m \cdot G_2) \cdot (I_k \cdot G_1)$
- $F' = F_2' \cdot F_1' = (I_m \cdot G_2) \cdot (G_1 \cdot I_n)$
- $F' = F_2' \cdot F_1' = (G_2 \cdot I_k) \cdot (G_1 \cdot I_n)$

where $I_j \in \boldsymbol{R}^{j \times j}$ denotes the identity in $\boldsymbol{R}^j$.

The corresponding operations counts (OC) can vary significantly, e.g, for $n = 4$, $k = 2$, $m = 32$, $|E_1| = |E_2| = 100$

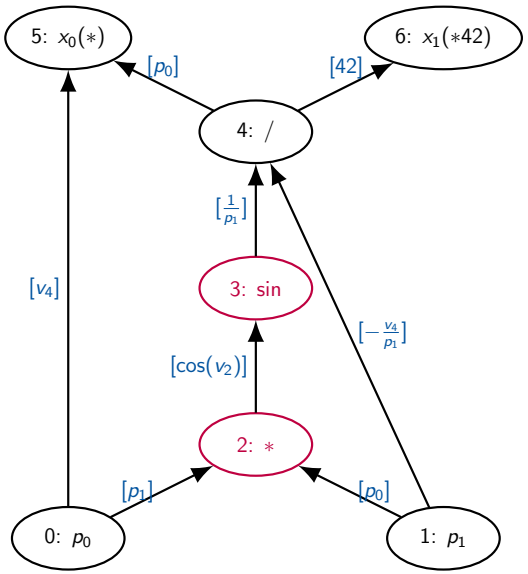
$$\begin{aligned}
\mathrm{OC}\,(G_2 \cdot (G_1 \cdot I_n)) &= 800 \\
\mathrm{OC}\,(G_2 \cdot (I_k \cdot G_1)) &= 600 \\
\mathrm{OC}\,((I_m \cdot G_2) \cdot G_1) &= 6400 \\
\mathrm{OC}\,((G_2 \cdot I_k) \cdot G_1) &= 3400 \\
\mathrm{OC}\,((G_2 \cdot I_k) \cdot (I_k \cdot G_1)) &= 656 \\
\mathrm{OC}\,((I_m \cdot G_2) \cdot (I_k \cdot G_1)) &= 3656 \\
\mathrm{OC}\,((I_m \cdot G_2) \cdot (G_1 \cdot I_n)) &= 3856 \\
\mathrm{OC}\,((G_2 \cdot I_k) \cdot (G_1 \cdot I_n)) &= 856.
\end{aligned}$$

Wake Up!

Modify n, k and m such that $\mathrm{OC}\left(\left(I_m \cdot G_2\right) \cdot G_1\right)$ becomes minimal.

In general, preaccumulation is employed to save memory and operations, e.g, $x = F(p) = F_2(F_1(p))$ s.t. $F_1 : p \mapsto v_4$, $F_2 : (p_0, v_4) \mapsto x$ yields

$I_2 \cdot G$

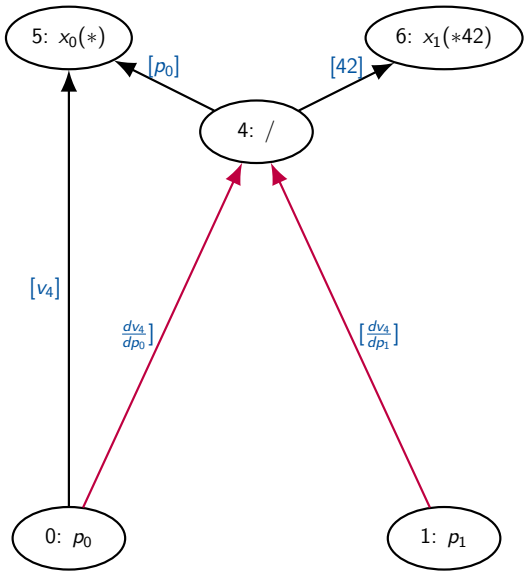


MEM=7+8=15

OC=5

$OC_{(1)} = 2 \cdot 8 = 16$

$I_1 \cdot G_1$

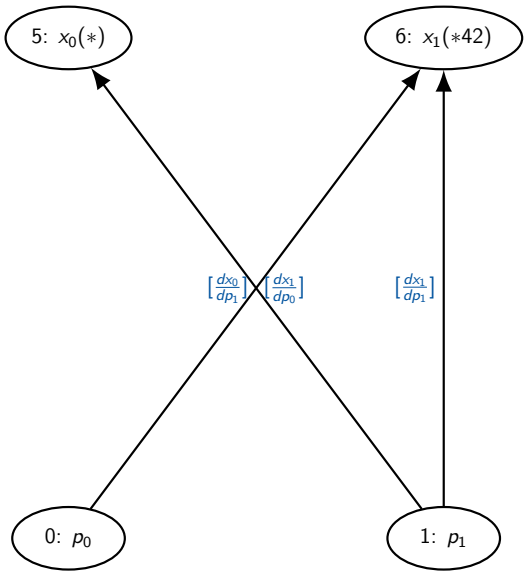


MEM=$5 + 5 + \ldots$

OC=$5 + \ldots$

$OC_{(1)} = 1 \cdot 5 + \ldots$

$(I_2 \cdot G_2) \cdot G_1$



$\ldots + 0 = 10$

$\ldots + 0 = 5$

$\ldots + 2 \cdot 5 = 15$

Outline

Summary