1. AD for $\frac{x^3}{8-x}$ at $x=3$

(a) Forward mode

$x = \langle 3, 1 \rangle$

$x^2 = \langle 3,1\rangle \cdot \langle 3,1\rangle$

$= \langle 9, 1\cdot 3 + 3\cdot 1\rangle$ (product rule)

$= \langle 9, 6 \rangle$

$x^3 = x^2 \cdot x = \langle 3,1\rangle \cdot \langle 9,6\rangle$

$= \langle 27, 1\cdot 9 + 3\cdot 6\rangle$ (product rule)

$= \langle 27, 27\rangle$

$8 = \langle 8, 0\rangle$

$8 - x = \langle 8,0\rangle - \langle 3,1\rangle$

$= \langle 5, -1\rangle$

$$\frac{x^3}{8-x} = \frac{\langle 27,27\rangle}{\langle 5,-1\rangle}$$

$$= \left\langle \frac{27}{5}, \frac{27\cdot 5 - 27\cdot(-1)}{(5)^2} \right\rangle \quad \text{(quotient rule)}$$

$$= \left\langle \frac{27}{5}, \frac{6\cdot 27}{25} \right\rangle$$

$$= \left\langle \frac{27}{5}, \frac{162}{25} \right\rangle$$

Revised version of problem:

$x = \langle 3,1\rangle$

$x^2 = \langle 9, 1\cdot 3+3\cdot 1\rangle = \langle 9,6\rangle$

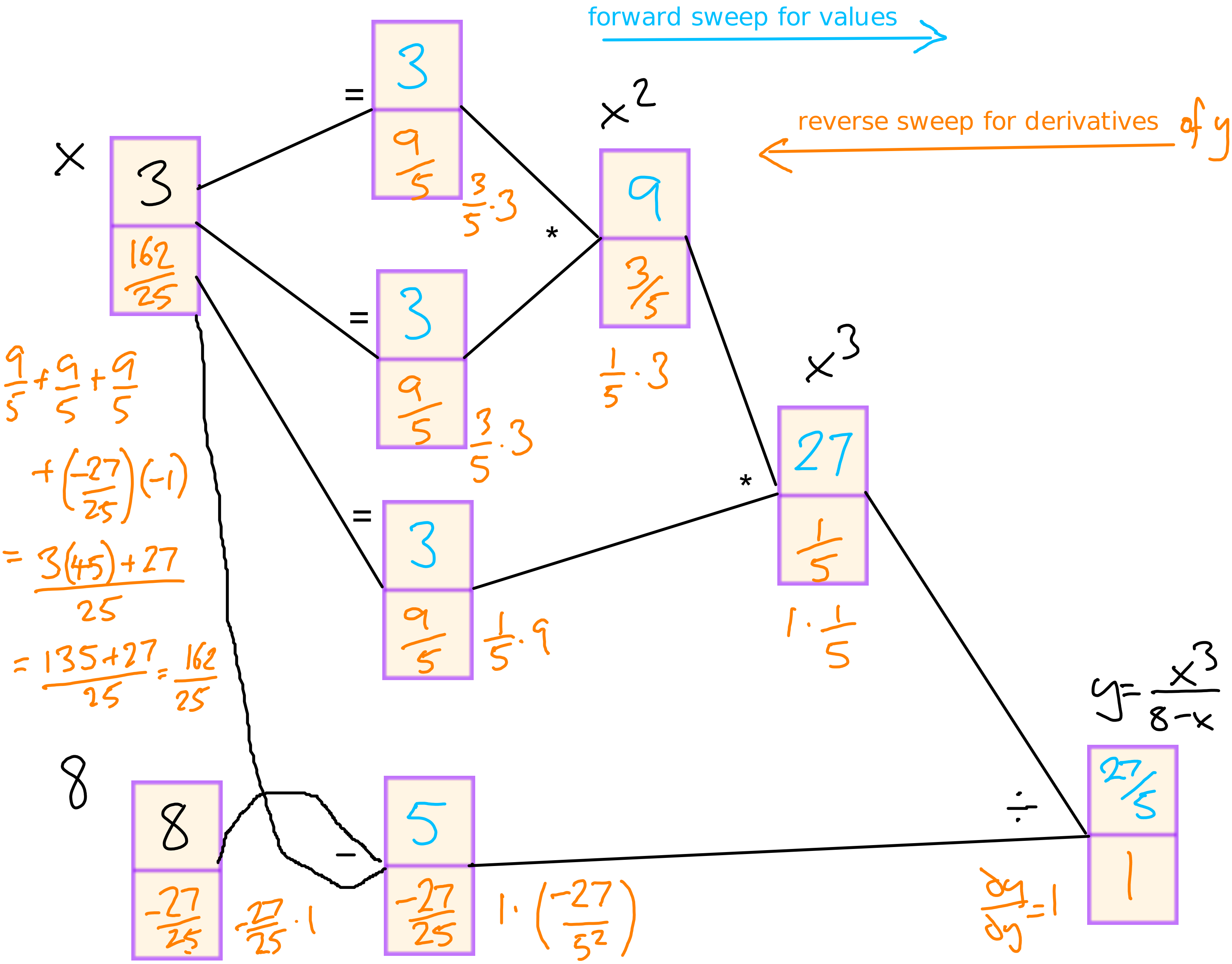
$x^3 = x^2\cdot x = \langle 9,6\rangle\langle 3,1\rangle = \langle 27, 6\cdot 3 + 9\cdot 1\rangle = \langle 27,27\rangle$

$10 - x = \langle 10,0\rangle - \langle 3,1\rangle = \langle 7,-1\rangle$

$$\frac{x^3}{10-x} = \frac{\langle 27,27\rangle}{\langle 7,-1\rangle} = \left\langle \frac{27}{7}, \frac{27\cdot 7 - 27\cdot(-1)}{(7)^2} \right\rangle = \left\langle \frac{27}{7}, \frac{27\cdot 8}{49}\right\rangle = \left\langle \frac{27}{7}, \frac{216}{49}\right\rangle$$

.

(b) reverse mode, original version of problem $y = \frac{x^3}{8-x}$.



Check results by symbolic differentiation:

```
import sympy as sp
x,w = sp.symbols('x,w')
y = x**3/(w-x)
dydx = sp.diff(y,x)
dydw = sp.diff(y,w)
xval,wval = 3,10
print(f'Partial derivatives of y at x={xval}, w={wval}')
print('wrt x:')
display(dydx.subs({x:xval,w:wval}))
print('wrt w:')
display(dydw.subs({x:xval,w:wval}))
```

```
Partial derivatives of y at x=3, w=8
wrt x:
```

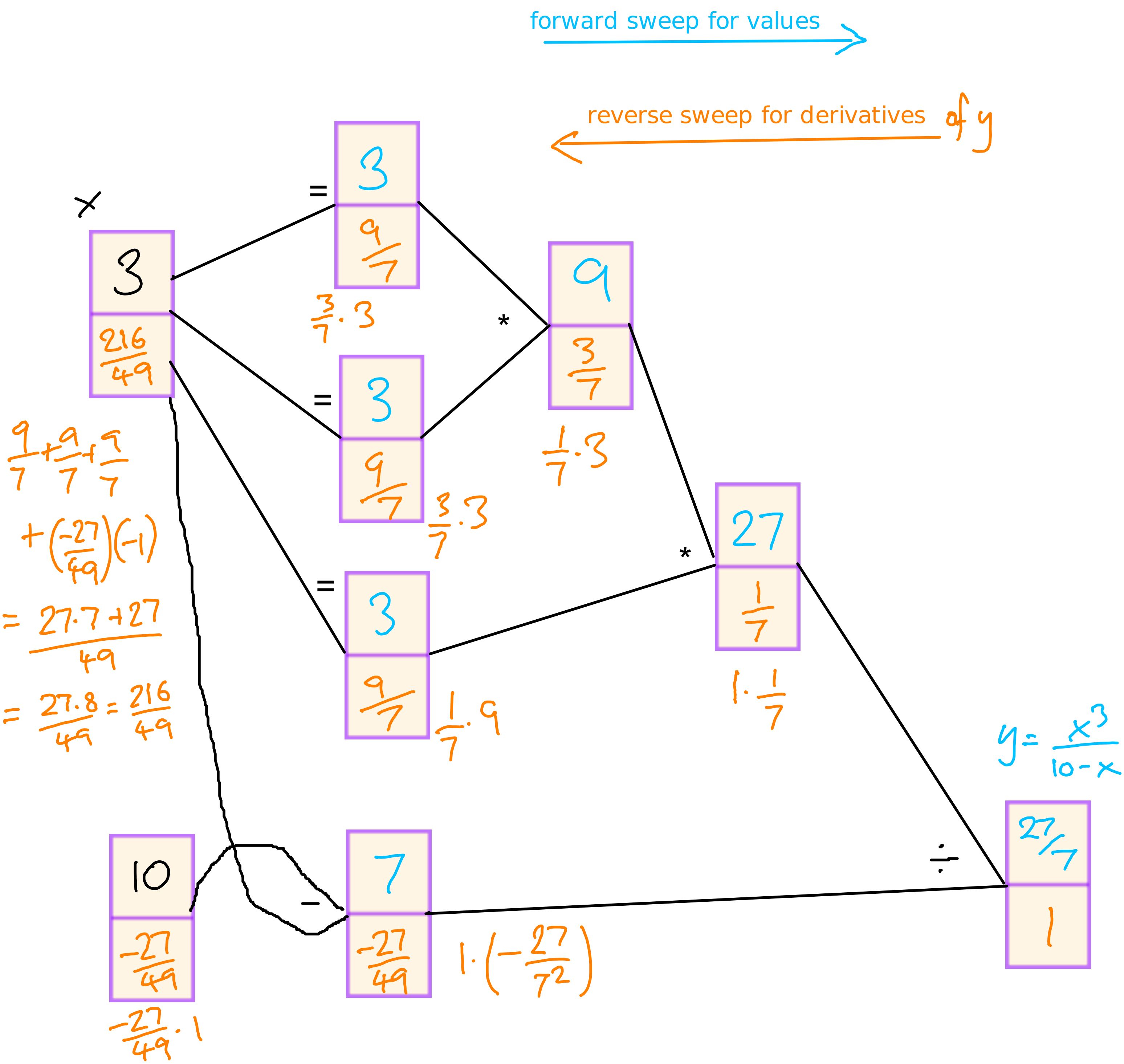
$$\frac{162}{25}$$ ✓

agreement

```
wrt w:
```

$$-\frac{27}{25}$$ ✓

(b) reverse mode: modified version of problem $y = \frac{x^3}{10-x}$



Check results by symbolic differentiation:

```
import sympy as sp
x,w = sp.symbols('x,w')
y = x**3/(w-x)
dydx = sp.diff(y,x)
dydw = sp.diff(y,w)
xval,wval = 3,10
print(f'Partial derivatives of y at x={xval}, w={wval}')
print('wrt x:')
display(dydx.subs({x:xval,w:wval}))
print('wrt w:')
display(dydw.subs({x:xval,w:wval}))
```

```
Partial derivatives of y at x=3, w=10
wrt x:
```

$\frac{216}{49}$ ✓

agreement!

```
wrt w:
```

$-\frac{27}{49}$ ✓

2. Forward-mode implementation of exponentiation

Let $h = f^g$. We want to know h'.

$(\log h)' = \frac{1}{h} h'$ (chain rule), so $h' = h (\log h)'$.

This is useful because $\log h = g \log f$, hence

$$(\log h)' = g' \log f + g \cdot \frac{1}{f} f'$$ by product & chain rules.

Thus $$\boxed{(f^g)' = f^g \cdot \left(g' \log f + g \frac{f'}{f} \right)}$$

And the rule for our AD objects is

$$\langle f, f' \rangle^{\langle g, g' \rangle} = \langle f^g, f^g \cdot \left(g' \log f + g \frac{f'}{f} \right) \rangle.$$

My implementation:

```python

    def __pow__(self,other):
        f,g = self,other
        if isinstance(g,ad): # handle case of exponent being just a number, not an ad object
            return ad( f.val**g.val, f.val**g.val*( g.der*math.log(f.val) + g.val*f.der/f.val  ))
        else:
            gad = ad(g)
            return ad( f.val**gad.val, f.val**gad.val*( gad.der*math.log(f.val) + gad.val*f.der/f.val ))

    def __rpow__(self,other): # to handle case when left operand is not an ad object
        f,g = other,self
        if isinstance(f,ad):
            return ad( f.val**g.val, f.val**g.val*( g.der*math.log(f.val) + g.val*f.der/f.val  ))
        else:
            fad = ad(f)
            return ad( fad.val**g.val, fad.val**g.val*( g.der*math.log(fad.val) + g.val*fad.der/fad.val ))

```

Testing it:

```python

def check_with_sympy(expr,var,val):
    print( 'sympy check:',expr.subs({var:val}), sp.diff(expr,var).subs({var:val}) ,'\n')

xsym = sp.symbols('x')

x = ad(3.,1.)

expr = xsym**2
y    = x**2
display(expr)
print('at x =',x.val)
print(y)
check_with_sympy(expr,xsym,x.val)


expr = 2.**xsym
y    = 2.**x
display(expr)
print('at x =',x.val)
print(y)
check_with_sympy(expr,xsym,x.val)


expr = (xsym+2.)**sp.sin(xsym)
y    = (x+2.)**sin(x)
display(expr)
print('at x =',x.val)
print(y)
check_with_sympy(expr,xsym,x.val)



```

x^2

```
at x = 3.0
< 9.0, 6.0 >
sympy check: 9.00000000000000 6.00000000000000
```

2.0^x

```
at x = 3.0
< 8.0, 5.545177444479562 >
sympy check: 8.00000000000000 5.54517744447956
```

$(x + 2.0)^{\sin(x)}$

```
at x = 3.0
< 1.2549853399170905, -1.9641869119747395 >
sympy check: 1.25498533991709 -1.96418691197474
```

AD resuts agree with sympy.