On the Practical Exploitation of Scarsity

Andrew Lyons[1] Jean Utke[1,2]

[1]University of Chicago

[2]Argonne National Laboratory

The 5th International Conference on Automatic Differentiation
(AD 2008)
August 14, 2008

Outline

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ *plus* a collection of p Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \ldots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ *plus* a collection of p Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \ldots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

or a collection of p Jacobian-transpose-vector products

$$F'(\mathbf{x})^T\bar{\mathbf{y}}^1, F'(\mathbf{x})^T\bar{\mathbf{y}}^2, \ldots, F'(\mathbf{x})^T\bar{\mathbf{y}}^p$$

Context

Given program for $\mathbf{y} = F(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^m$

Want a program $F^+(\mathbf{x})$ that computes $F(\mathbf{x})$ *plus* a collection of p Jacobian-vector products

$$F'(\mathbf{x})\dot{\mathbf{x}}^1, F'(\mathbf{x})\dot{\mathbf{x}}^2, \ldots, F'(\mathbf{x})\dot{\mathbf{x}}^p$$

or a collection of p Jacobian-transpose-vector products

$$F'(\mathbf{x})^T\bar{\mathbf{y}}^1, F'(\mathbf{x})^T\bar{\mathbf{y}}^2, \ldots, F'(\mathbf{x})^T\bar{\mathbf{y}}^p$$

As needed in Newton Krylov methods, etc.

Evaluation Procedures

$$y_1 = x_1 * x_1 * x_2, \quad y_2 = \sin(x_1 * x_1 * x_2)$$

$v_{-1} = x_1$

$v_0 = x_2$

$v_1 = v_{-1} * v_0$

$v_2 = v_{-1} * v_1$

$v_3 = v_2 + 1$

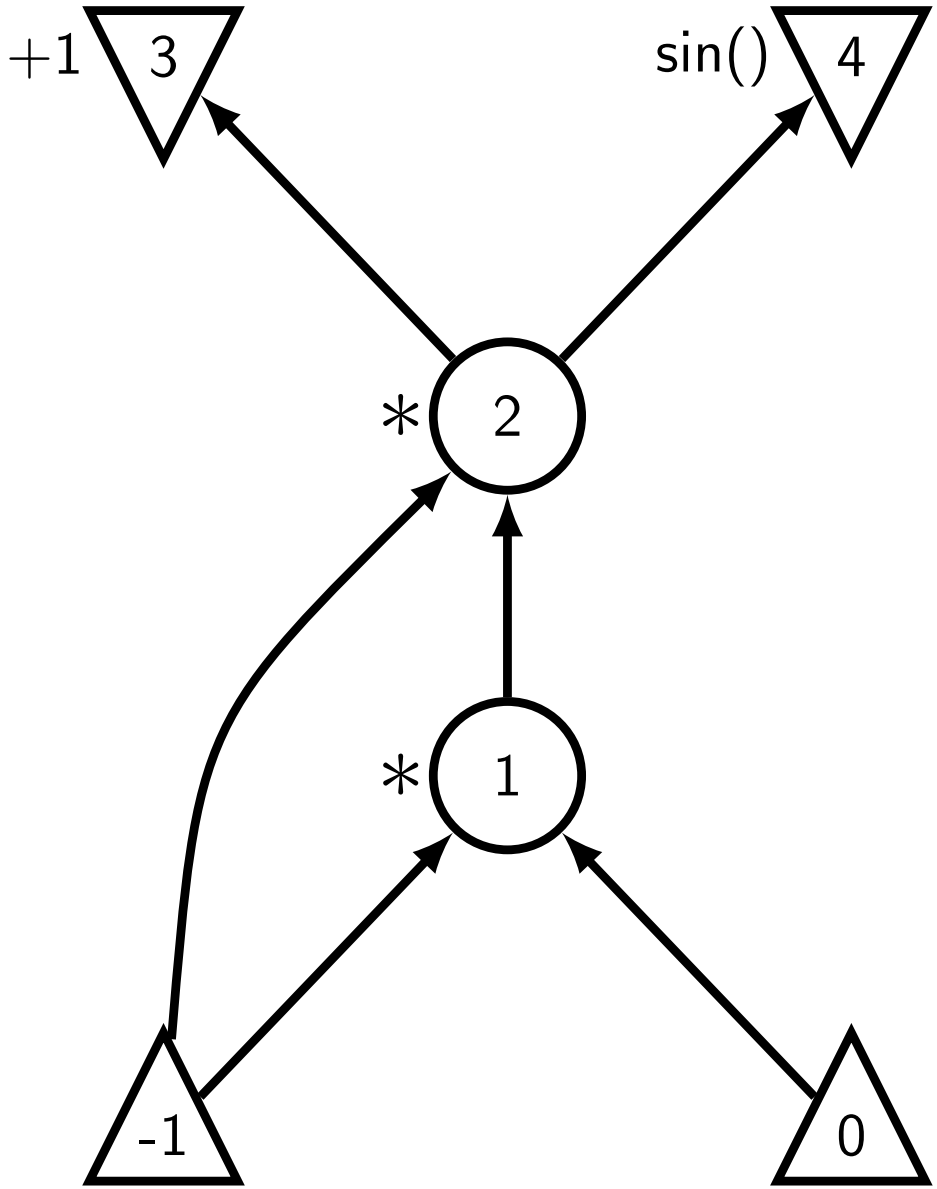
$v_4 = \sin(v_2)$

$y_1 = v_3$

$y_2 = v_4$

Evaluation Procedures and **Computational Graphs**

$$y_1 = x_1 * x_1 * x_2, \quad y_2 = \sin(x_1 * x_1 * x_2)$$



$v_{-1} = x_1$

$v_0 = x_2$

$v_1 = v_{-1} * v_0$

$v_2 = v_{-1} * v_1$

$v_3 = v_2 + 1$

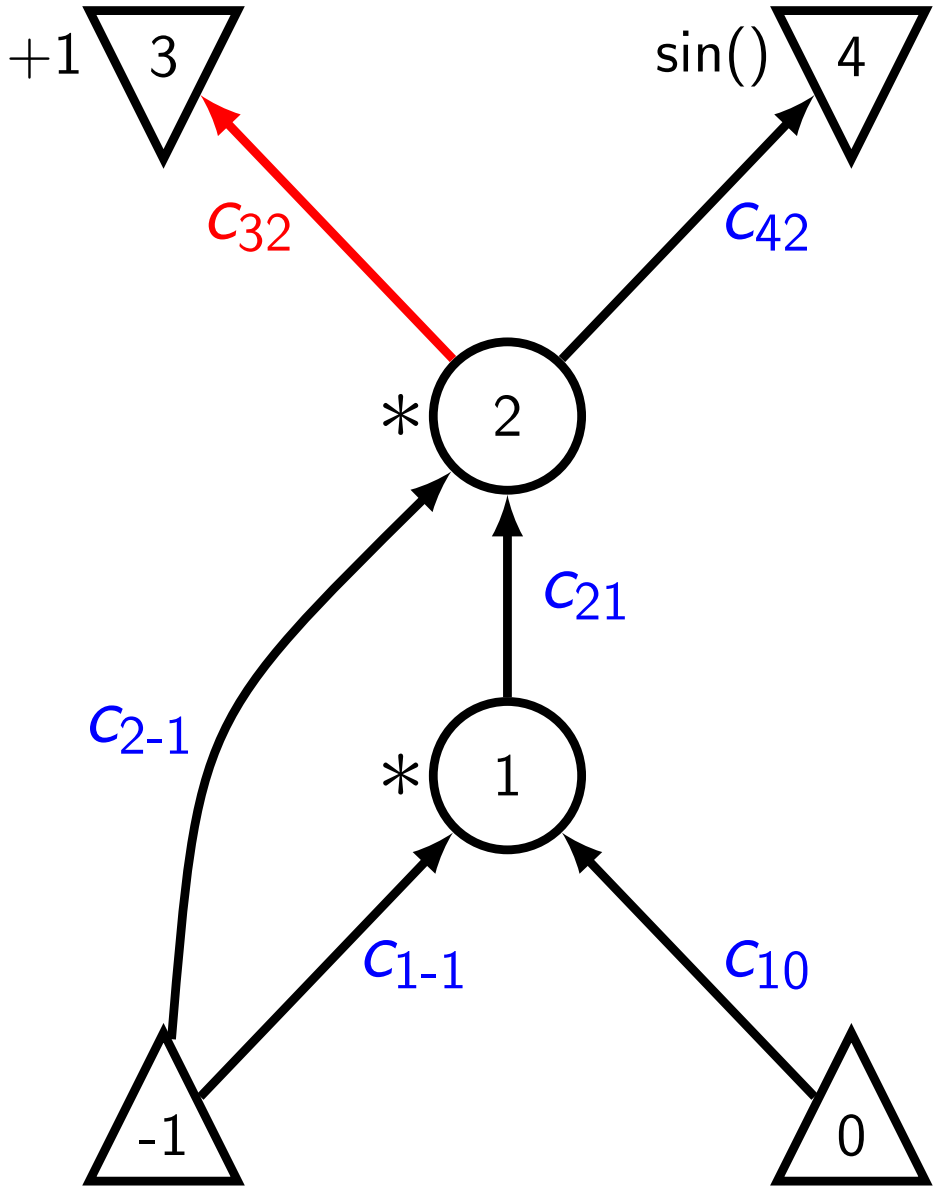
$v_4 = \sin(v_2)$

$y_1 = v_3$

$y_2 = v_4$

Computational Graphs and **Linearization**

$$y_1 = x_1 * x_1 * x_2, \quad y_2 = \sin(x_1 * x_1 * x_2)$$



$v_{-1} = x_1$

$v_0 = x_2$

$v_1 = v_{-1} * v_0$

$c_{1-1} = v_0$

$c_{10} = v_1$

$v_2 = v_{-1} * v_1$

$c_{2-1} = v_1$

$c_{21} = v_{-1}$

$v_3 = v_2 + 1$

$c_{32} = 1$

$v_4 = \sin(v_2)$

$c_{42} = \cos(v_2)$

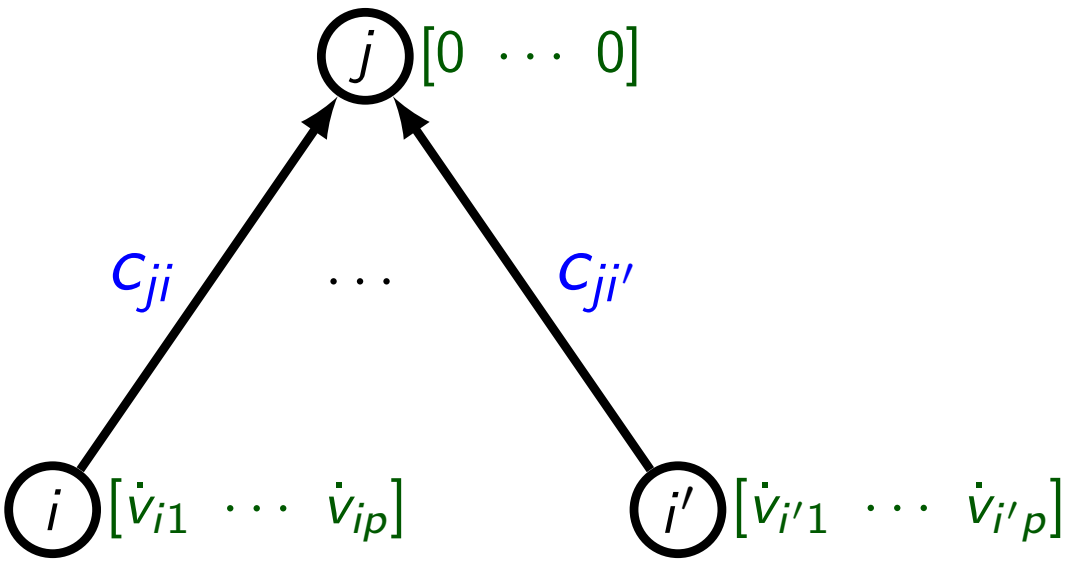
$y_1 = v_3$

$y_2 = v_4$

Forward **Propagation** of Vectors

Associate a derivative vector $\dot{\mathbf{v}}_\mathbf{j} \in \mathbb{R}^p$ with each variable v_j, propagate through G by

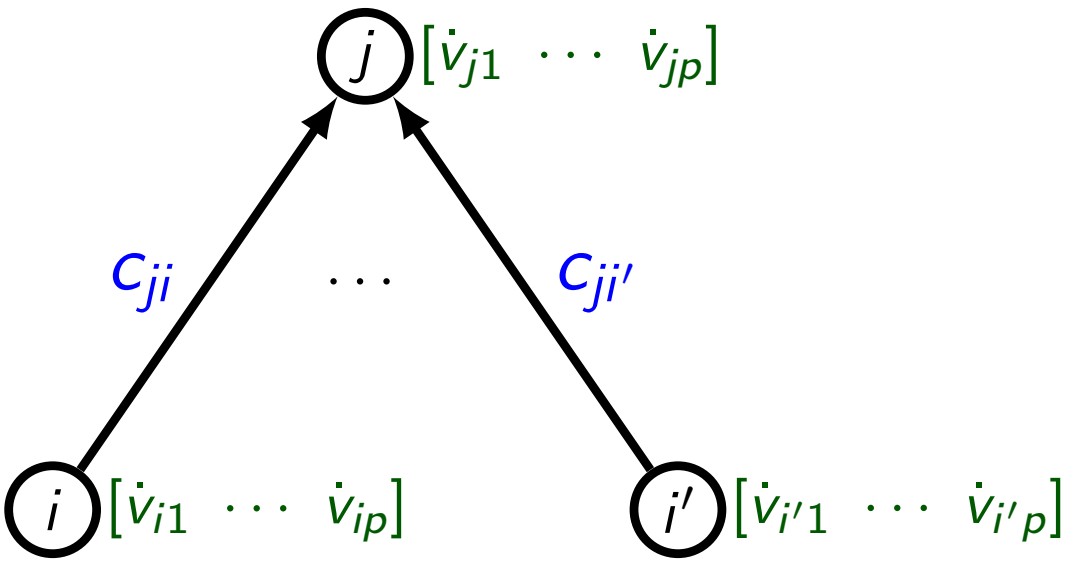
$$\dot{\mathbf{v}}_\mathbf{j} = \sum_{i \prec j} c_{ji} \dot{\mathbf{v}}_\mathbf{i} \qquad \text{(BLAS level 1 axpy operation)}$$



Forward **Propagation** of Vectors

Associate a derivative vector $\dot{\mathbf{v}}_\mathbf{j} \in \mathbb{R}^p$ with each variable v_j, propagate through G by

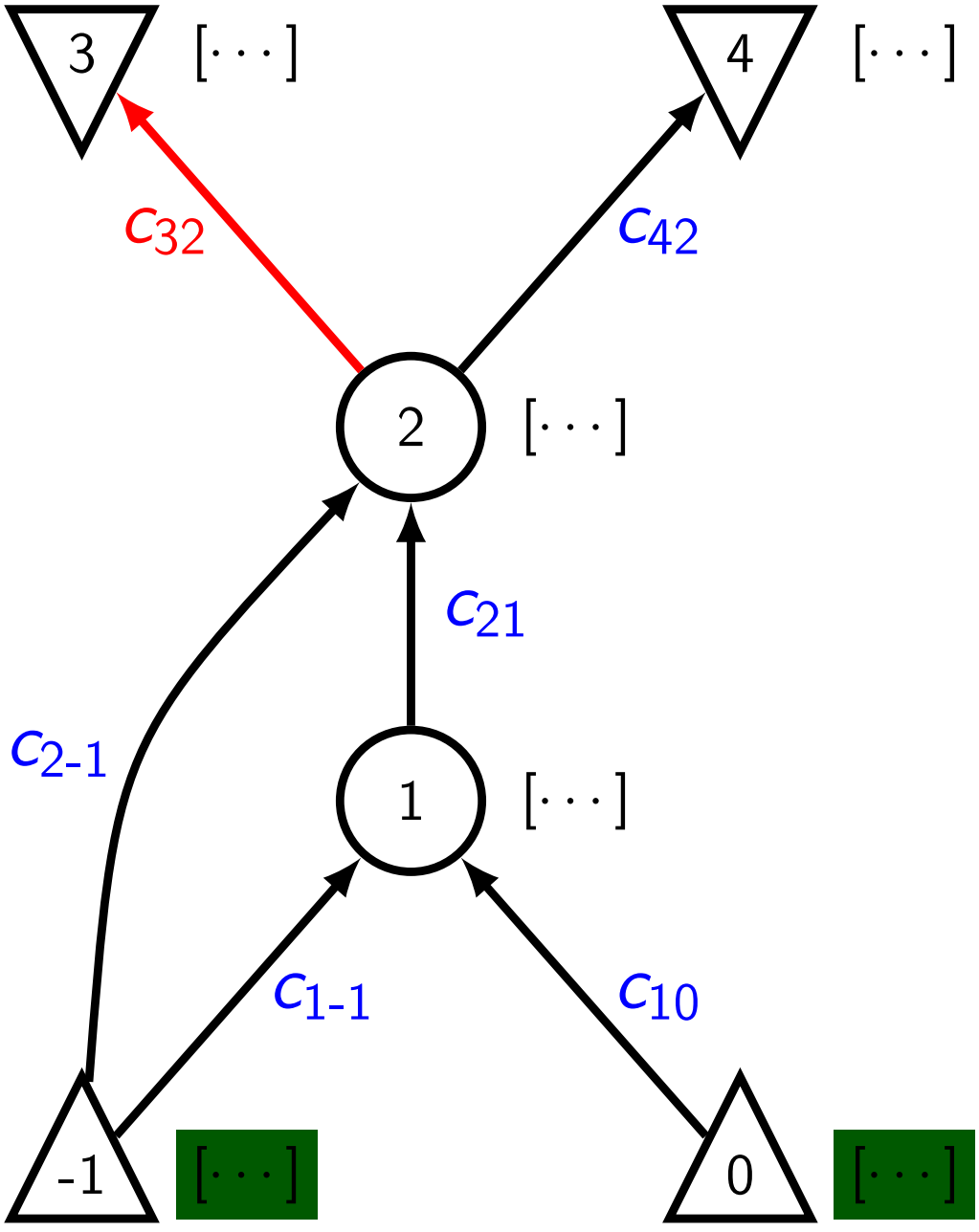
$$\dot{\mathbf{v}}_\mathbf{j} = \sum_{i \prec j} c_{ji} \dot{\mathbf{v}}_\mathbf{i} \qquad \text{(BLAS level 1 axpy operation)}$$



Generated propagation code:

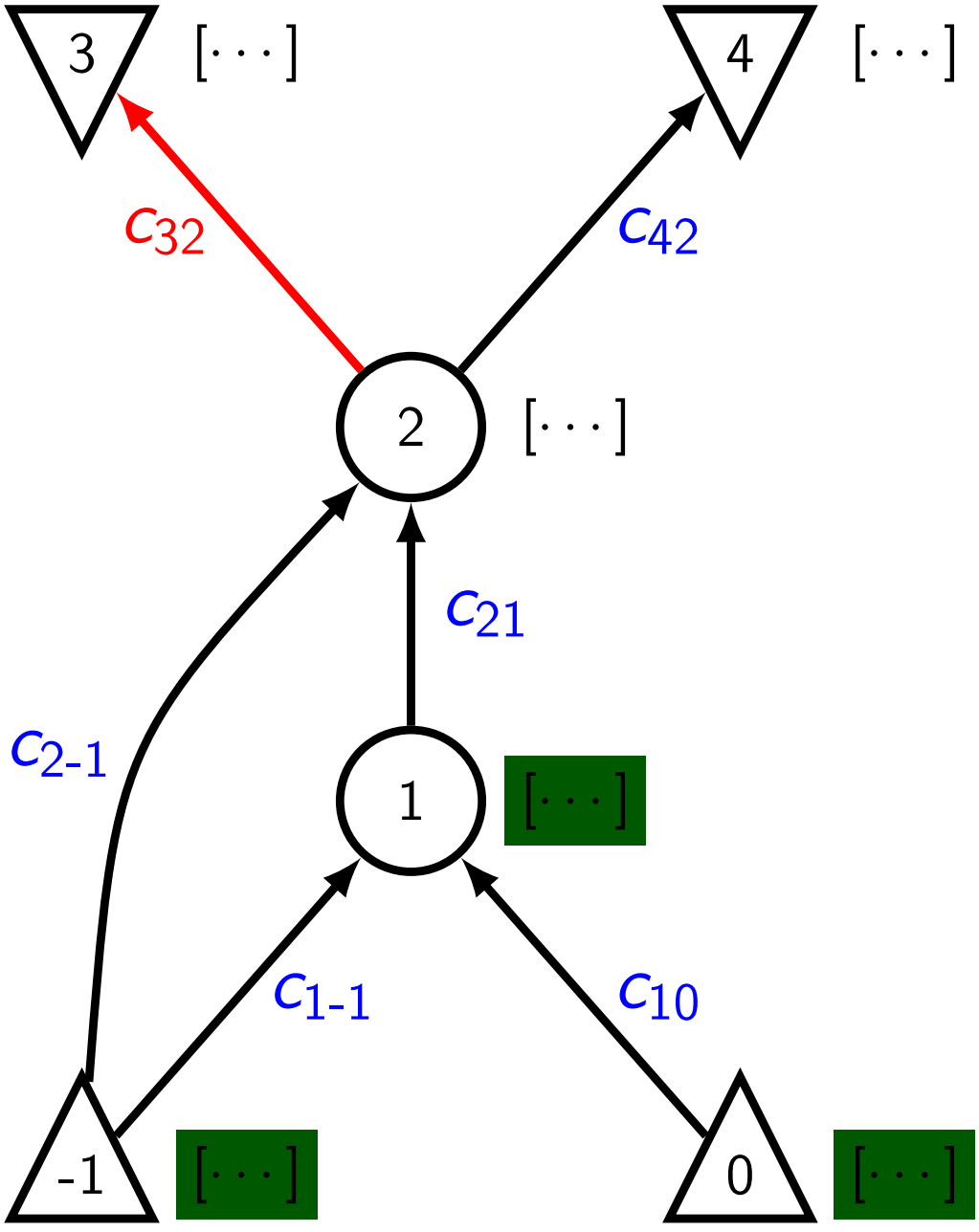
$$\begin{array}{lcl} \dot{\mathrm{v}}_j & = & c_{ji} * \dot{\mathrm{v}}_i \\ & \vdots & \\ \dot{\mathrm{v}}_j & \mathtt{+=} & c_{ji'} * \dot{\mathrm{v}}_{i'} \end{array}$$

Forward **Propagation** of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$
$$\dot{v}_0 = \dot{x}_2$$

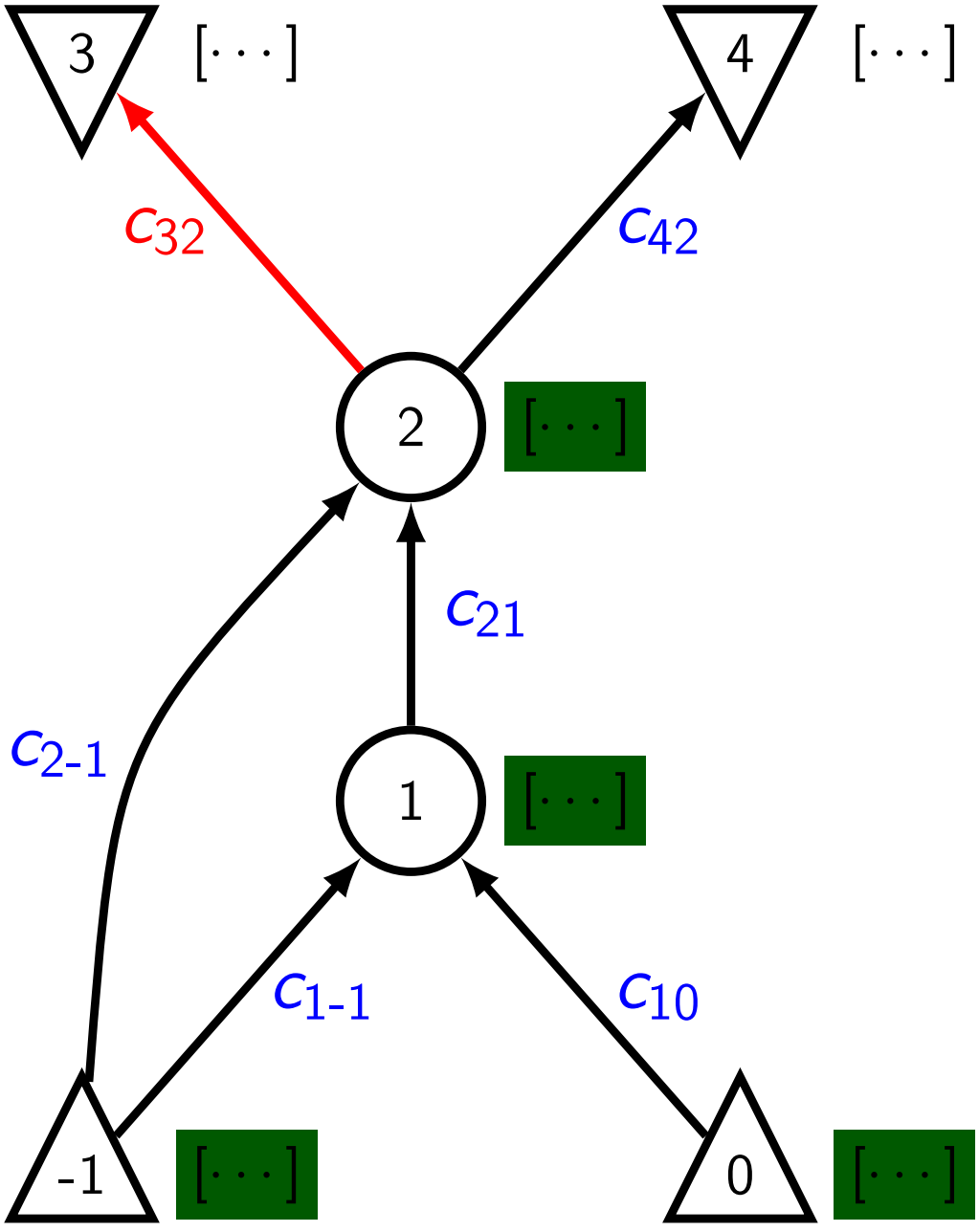
Forward **Propagation** of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$
$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$$
$$\dot{v}_1 \mathrel{+}= c_{10} * \dot{v}_0$$

Forward **Propagation** of Vectors



$\dot{v}_{-1} = \dot{x}_1$

$\dot{v}_0 = \dot{x}_2$

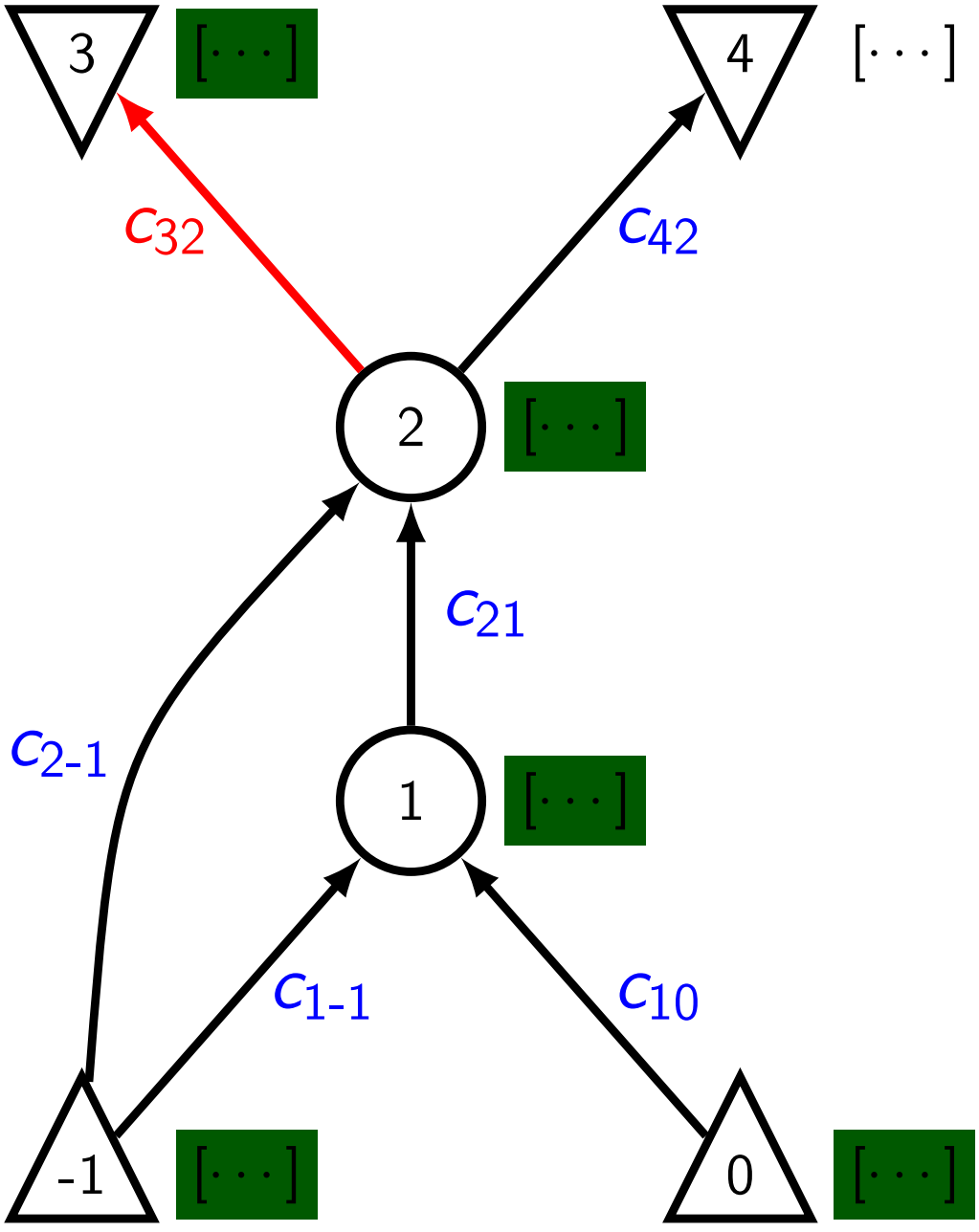
$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$

$\dot{v}_1 \mathrel{+}= c_{10} * \dot{v}_0$

$\dot{v}_2 = c_{2-1} * \dot{v}_{-1}$

$\dot{v}_2 \mathrel{+}= c_{21} * \dot{v}_1$

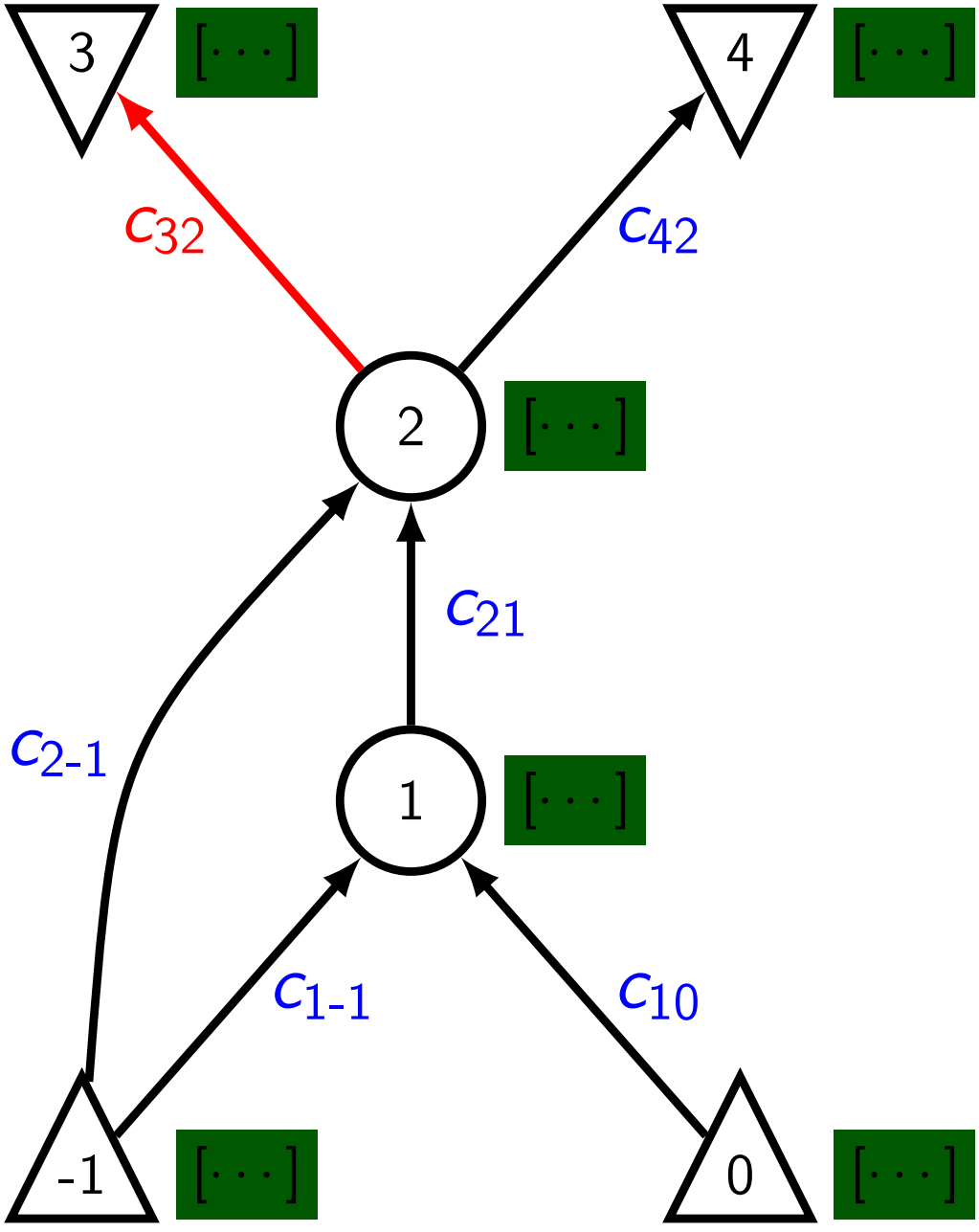
Forward **Propagation** of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$
$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$$
$$\dot{v}_1 \mathrel{+}= c_{10} * \dot{v}_0$$
$$\dot{v}_2 = c_{2-1} * \dot{v}_{-1}$$
$$\dot{v}_2 \mathrel{+}= c_{21} * \dot{v}_1$$
$$\dot{v}_3 = c_{32} * \dot{v}_2 = \dot{v}_2$$

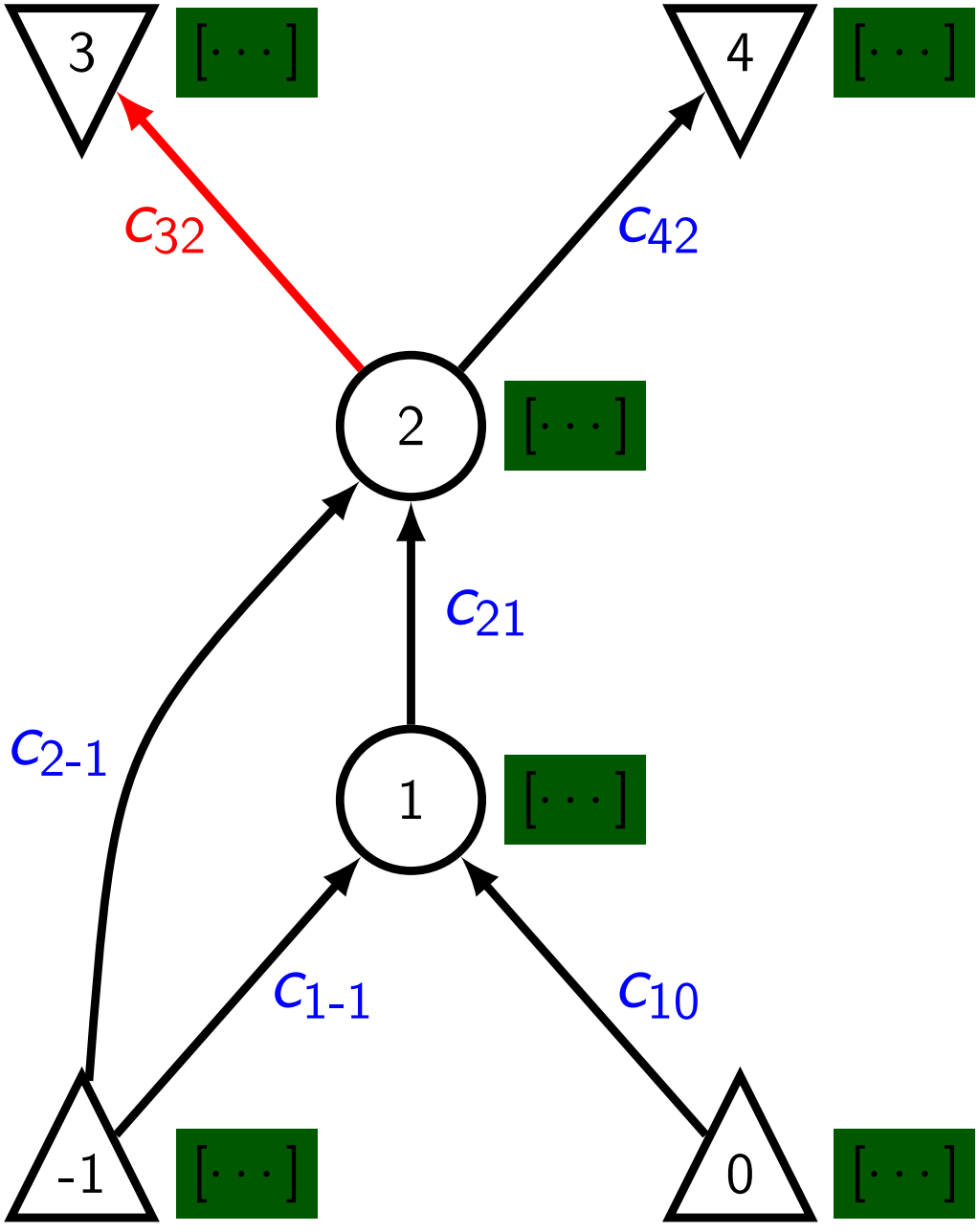
Forward **Propagation** of Vectors



$$\dot{v}_{-1} = \dot{x}_1$$
$$\dot{v}_0 = \dot{x}_2$$

$$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$$
$$\dot{v}_1 \mathrel{+}= c_{10} * \dot{v}_0$$
$$\dot{v}_2 = c_{2-1} * \dot{v}_{-1}$$
$$\dot{v}_2 \mathrel{+}= c_{21} * \dot{v}_1$$
$$\dot{v}_3 = c_{32} * \dot{v}_2 = \dot{v}_2$$
$$\dot{v}_4 = c_{42} * \dot{v}_2$$

Forward **Propagation** of Vectors



$\dot{v}_{-1} = \dot{x}_1$

$\dot{v}_0 = \dot{x}_2$

$\dot{v}_1 = c_{1-1} * \dot{v}_{-1}$

$\dot{v}_1 \mathrel{+}= c_{10} * \dot{v}_0$

$\dot{v}_2 = c_{2-1} * \dot{v}_{-1}$

$\dot{v}_2 \mathrel{+}= c_{21} * \dot{v}_1$

$5p$ mults

$\dot{v}_3 = c_{32} * \dot{v}_2 = \dot{v}_2$

$\dot{v}_4 = c_{42} * \dot{v}_2$

$\dot{y}_1 = \dot{v}_3$

$\dot{y}_2 = \dot{v}_4$

Reverse Vector Propagation

Propagates vectors $\bar{\mathbf{y}}^1, \ldots, \bar{\mathbf{y}}^p$ backwards

Works symmetrically
(same mult. cost, possibly different number of adds)

Yields Jacobian-transpose-vector products

$$F'(\mathbf{x})^T \bar{\mathbf{y}}^1, F'(\mathbf{x})^T \bar{\mathbf{y}}^2, \ldots, F'(\mathbf{x})^T \bar{\mathbf{y}}^p$$

Preaccumulation

Cost is proportional to the number of nonunit edges $\Rightarrow$ transform the graph!

Baur's formula (from chain rule) yields the entries of $F'(\mathbf{x})$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

Preaccumulation

Cost is proportional to the number of nonunit edges $\Rightarrow$ transform the graph!

Baur's formula (from chain rule) yields the entries of $F'(\mathbf{x})$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

Preaccumulation applies transformations $G \rightarrow G'$
Afterwards, Baur's formula still expresses the entries of J (we can still propagate vectors through it)

Preaccumulation

Cost is proportional to the number of nonunit edges $\Rightarrow$ transform the graph!

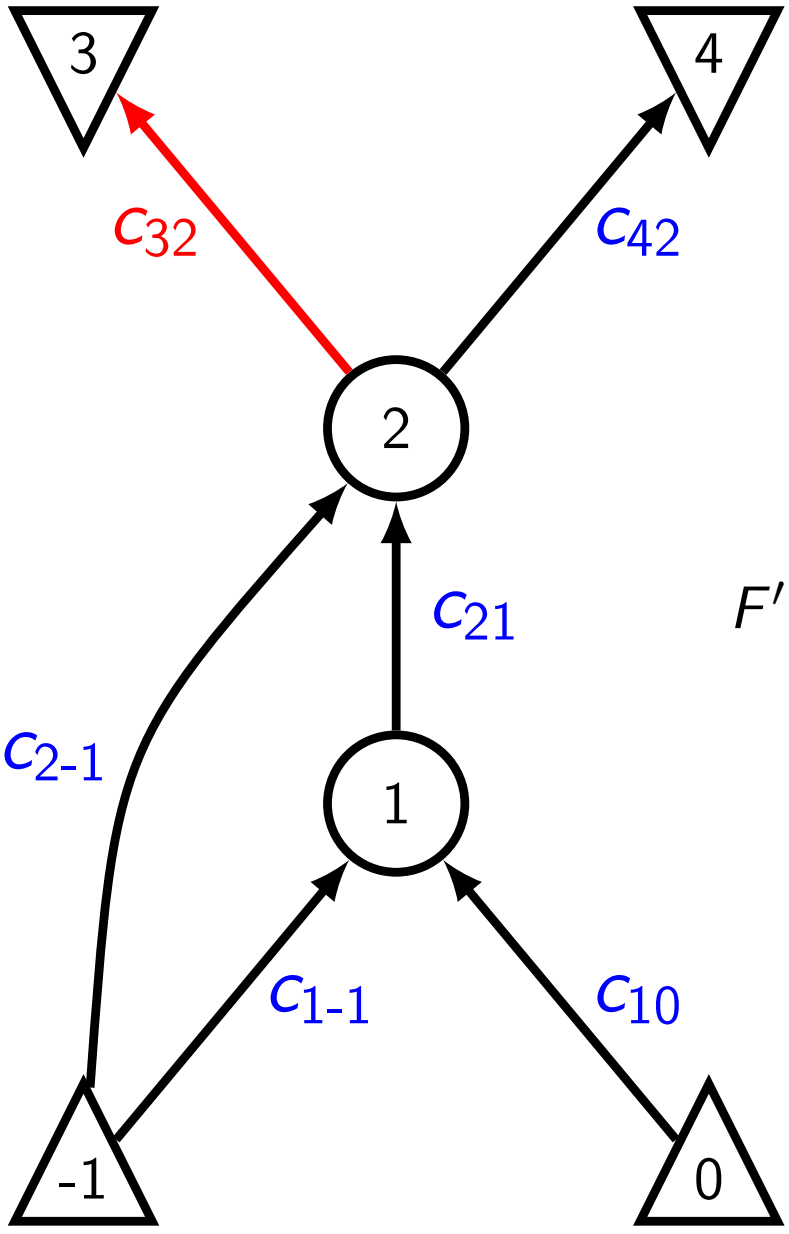
Baur's formula (from chain rule) yields the entries of $F'(\mathbf{x})$

$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

Preaccumulation applies transformations $G \rightarrow G'$
Afterwards, Baur's formula still expresses the entries of J (we can still propagate vectors through it)

Complete preaccumulation results in a bipartite graph, whose edges correspond to the nonzero entries of $F'(\mathbf{x})$
(In general, complete preaccumulation with minimal ops (OJA) is **NP**-hard)

Baur's Formula

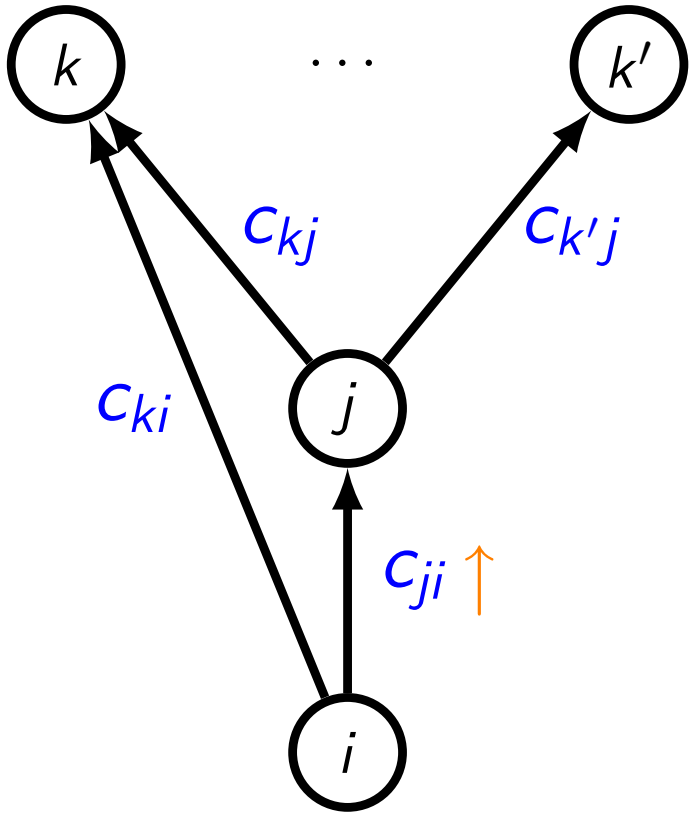


$$\frac{\partial y_j}{\partial x_i} = \sum_{P \in \mathcal{P}_{x_i}^{y_j}} \prod_{(k,\ell) \in P} c_{\ell k}$$

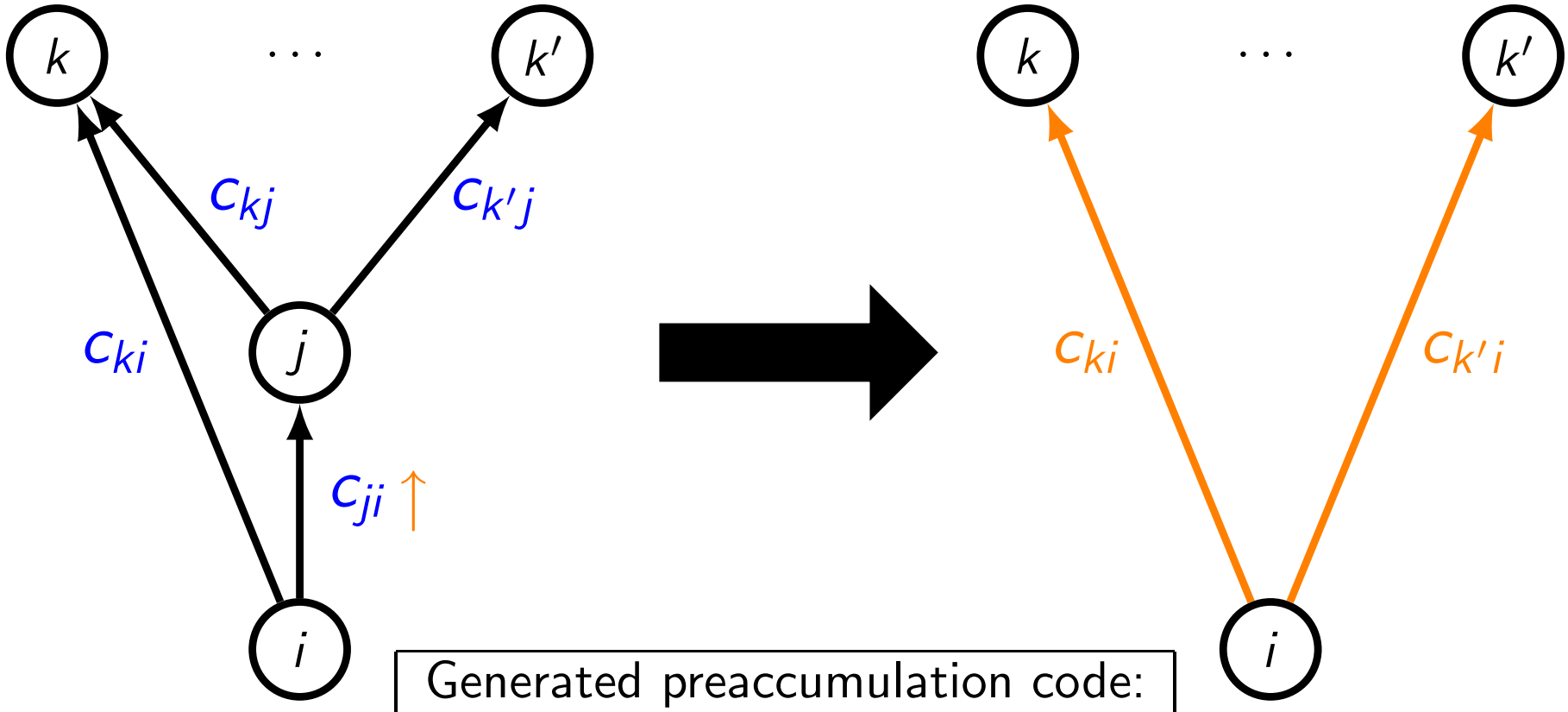
$$F'(\mathbf{x}) = \begin{bmatrix} c_{2\text{-}1}c_{32} + c_{1\text{-}1}c_{21}c_{32} & c_{10}c_{21}c_{32} \\ c_{2\text{-}1}c_{42} + c_{1\text{-}1}c_{21}c_{42} & c_{10}c_{21}c_{42} \end{bmatrix}$$

$$= \begin{bmatrix} c_{2\text{-}1} + c_{1\text{-}1}c_{21} & c_{10}c_{21} \\ c_{2\text{-}1}c_{42} + c_{1\text{-}1}c_{21}c_{42} & c_{10}c_{21}c_{42} \end{bmatrix}$$

Front Edge Elimination



Front Edge Elimination



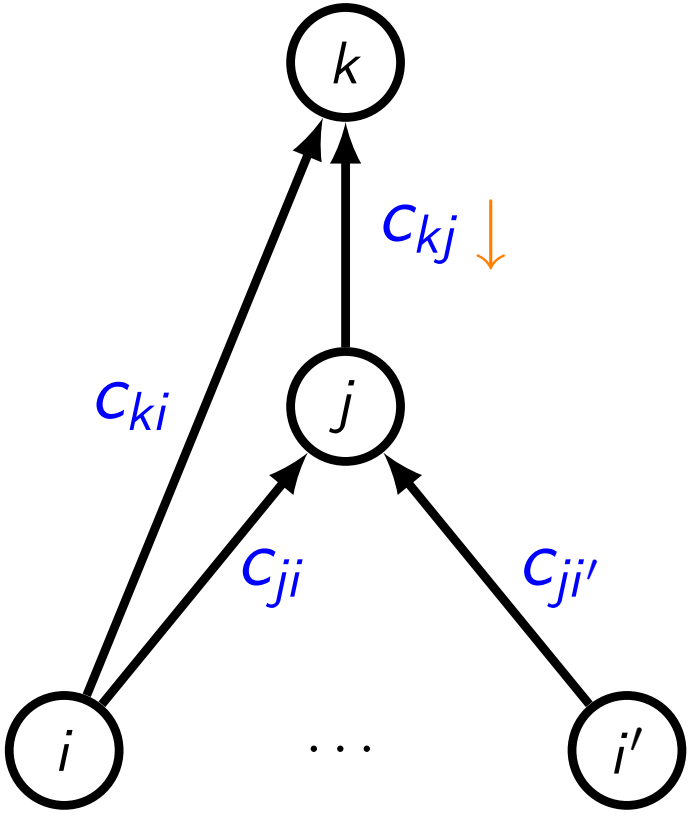
Generated preaccumulation code:

$$c_{ki} \mathrel{+}= c_{ji} * c_{kj}$$

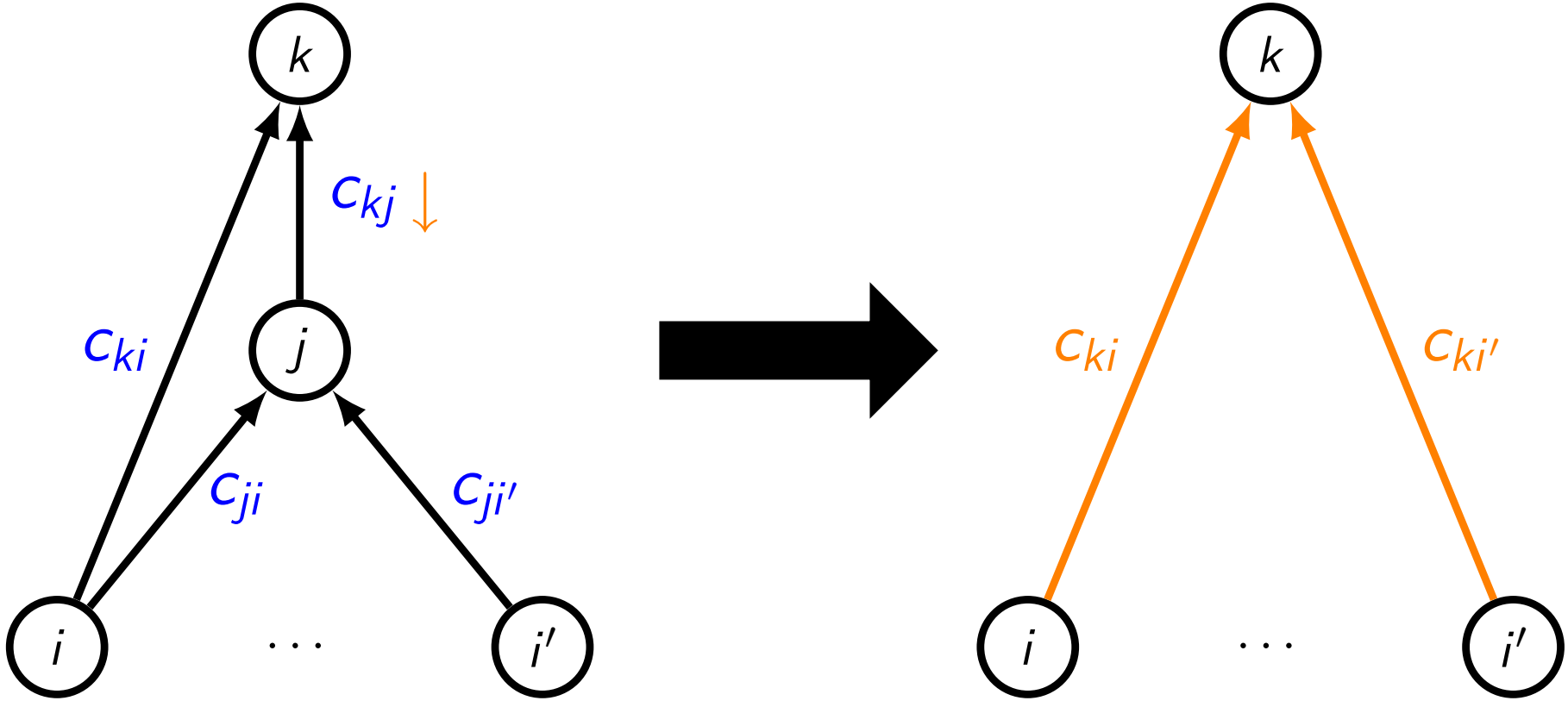
$$\vdots$$

$$c_{k'i} = c_{ji} * c_{k'j}$$

Back Edge Elimination



Back Edge Elimination

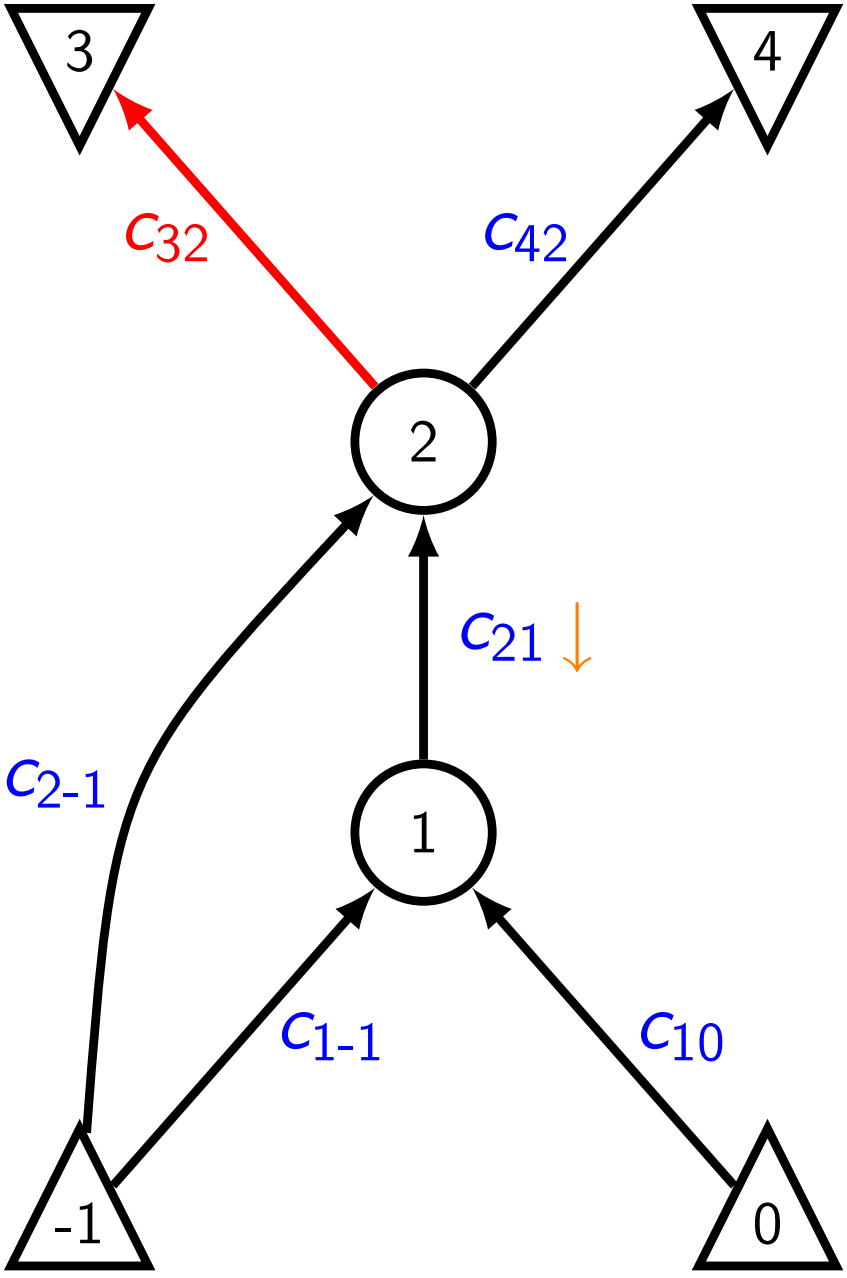


$$c_{ki} \mathrel{+}= c_{ji} * c_{kj}$$

$$\vdots$$

$$c_{ki'} = c_{ji'} * c_{kj}$$

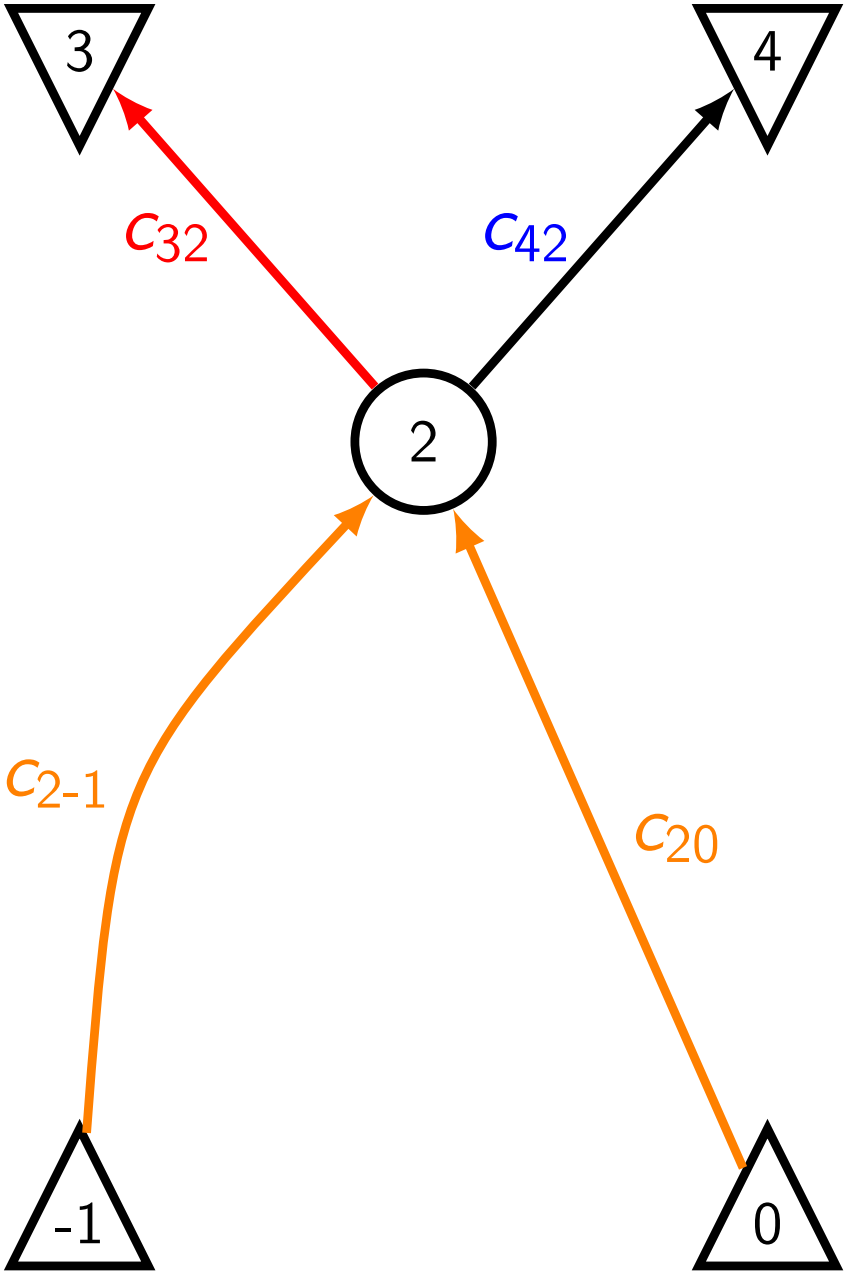
Preaccumulation



(Code for F, linearization)

$\vdots$

Preaccumulation



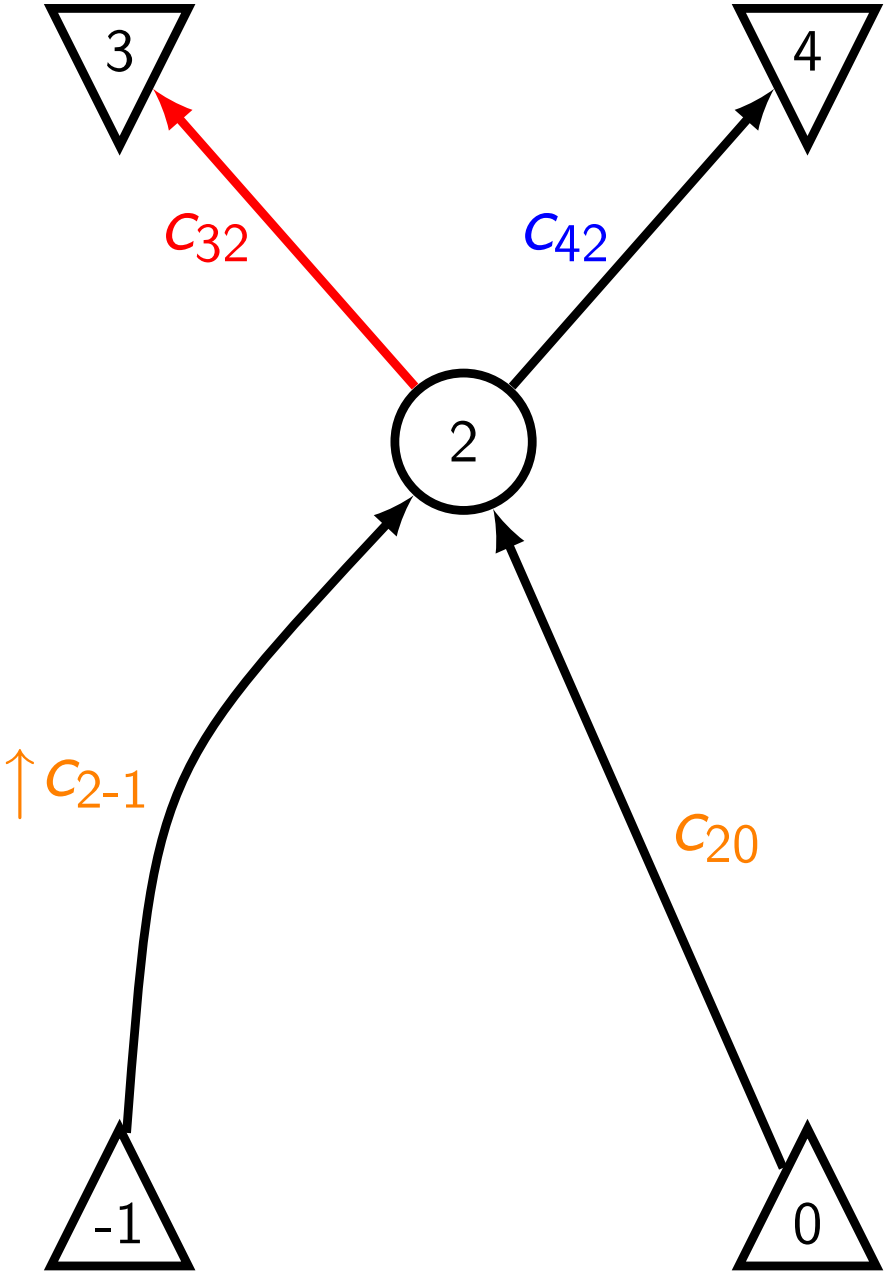
(Code for F, linearization)

$\vdots$

$c_{2-1} = c_{1-1} * c_{21}$

$c_{20} = c_{10} * c_{21}$

Preaccumulation



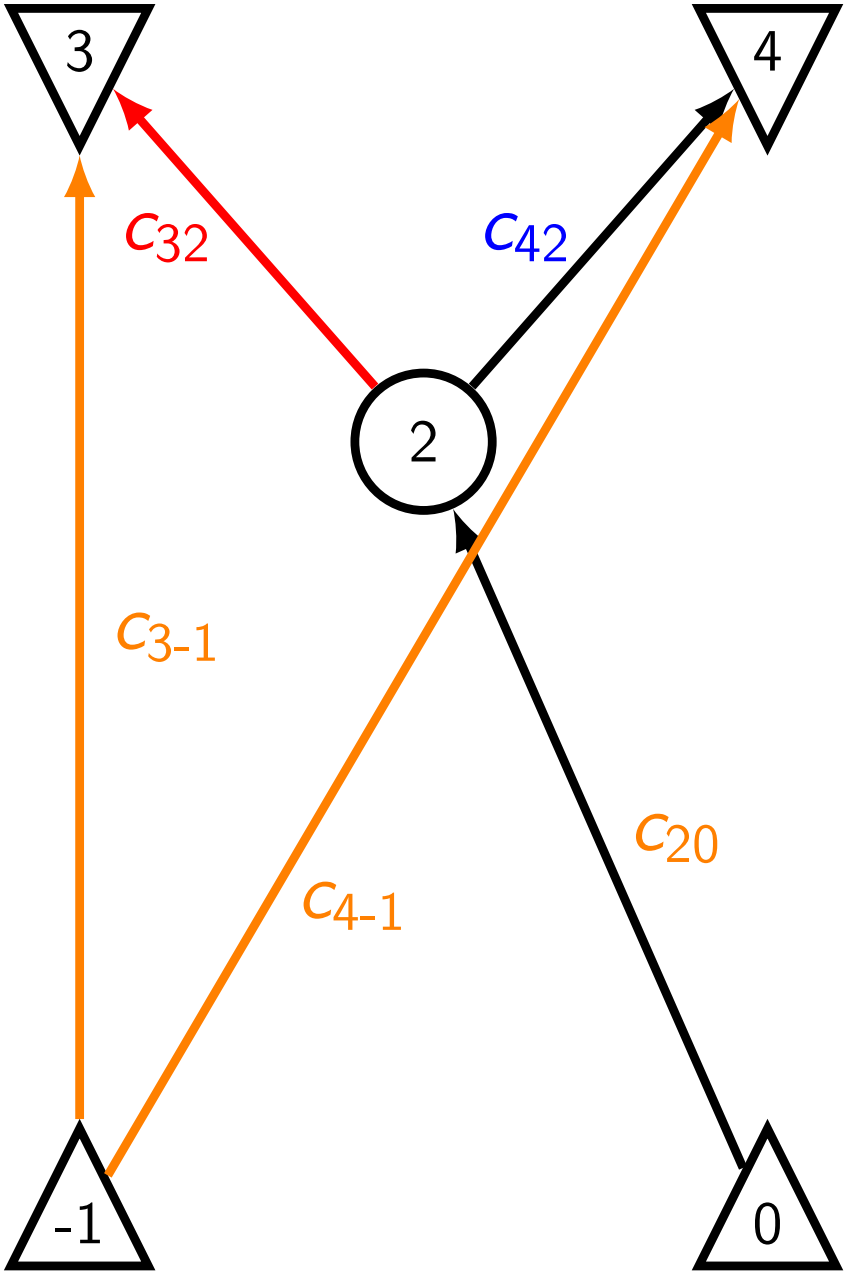
(Code for F, linearization)

$\vdots$

$c_{2-1} = c_{1-1} * c_{21}$

$c_{20} = c_{10} * c_{21}$

Preaccumulation

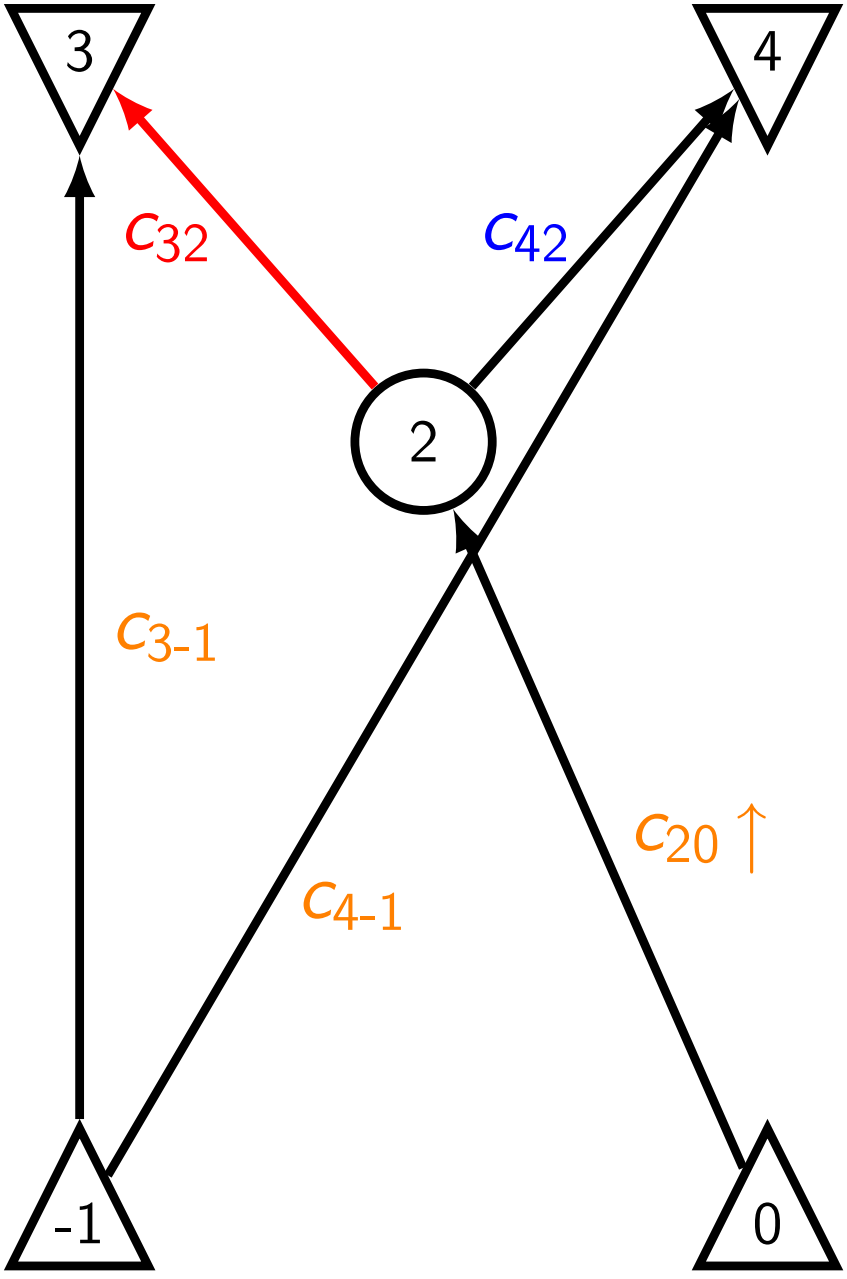


(Code for F, linearization)

$\vdots$

$$c_{2-1} = c_{1-1} * c_{21}$$
$$c_{20} = c_{10} * c_{21}$$
$$c_{3-1} = c_{2-1} * c_{32}$$
$$c_{4-1} = c_{2-1} * c_{42}$$

Preaccumulation



(Code for F, linearization)

⋮

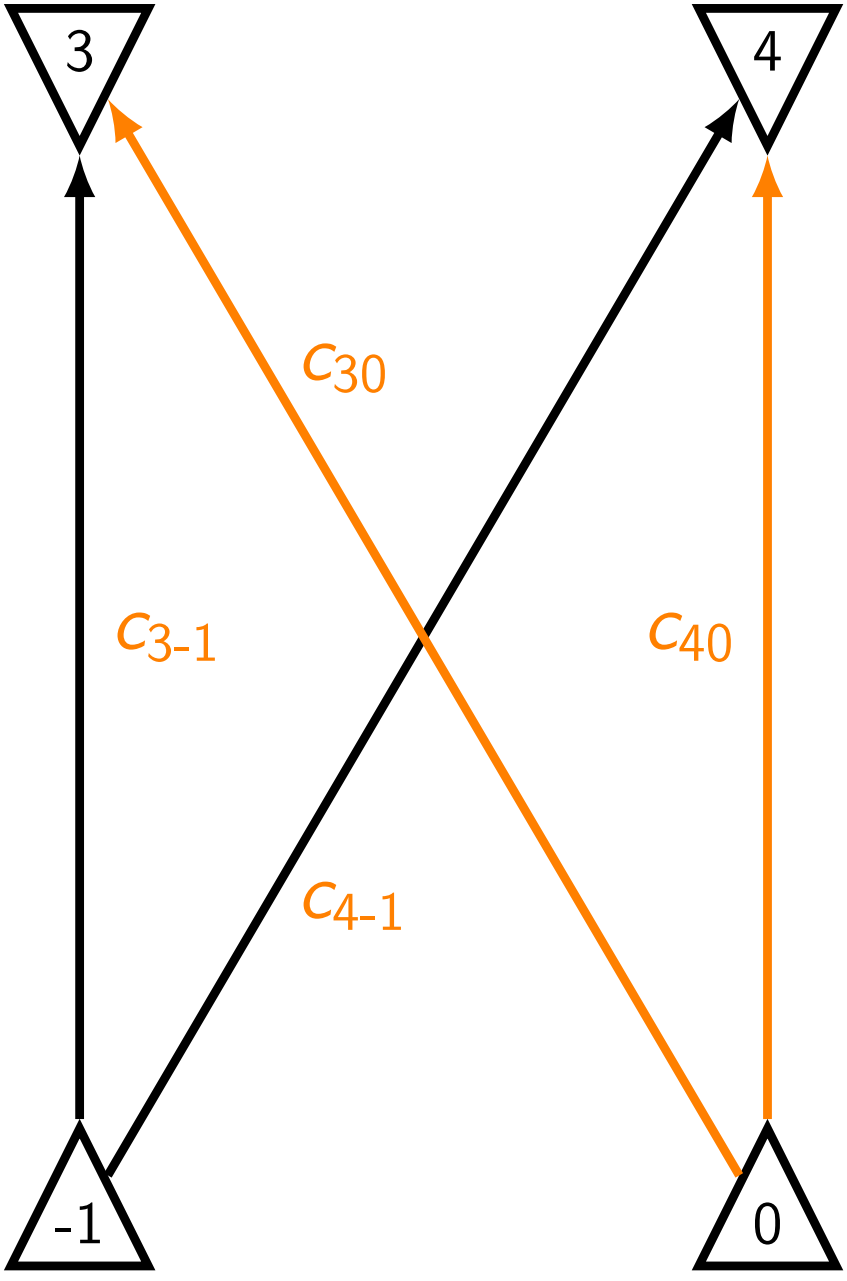
$c_{2-1} = c_{1-1} * c_{21}$

$c_{20} = c_{10} * c_{21}$

$c_{3-1} = c_{2-1} * c_{32}$

$c_{4-1} = c_{2-1} * c_{42}$

Preaccumulation



(Code for F, linearization)

$\vdots$

$c_{2-1} = c_{1-1} * c_{21}$

$c_{20} = c_{10} * c_{21}$

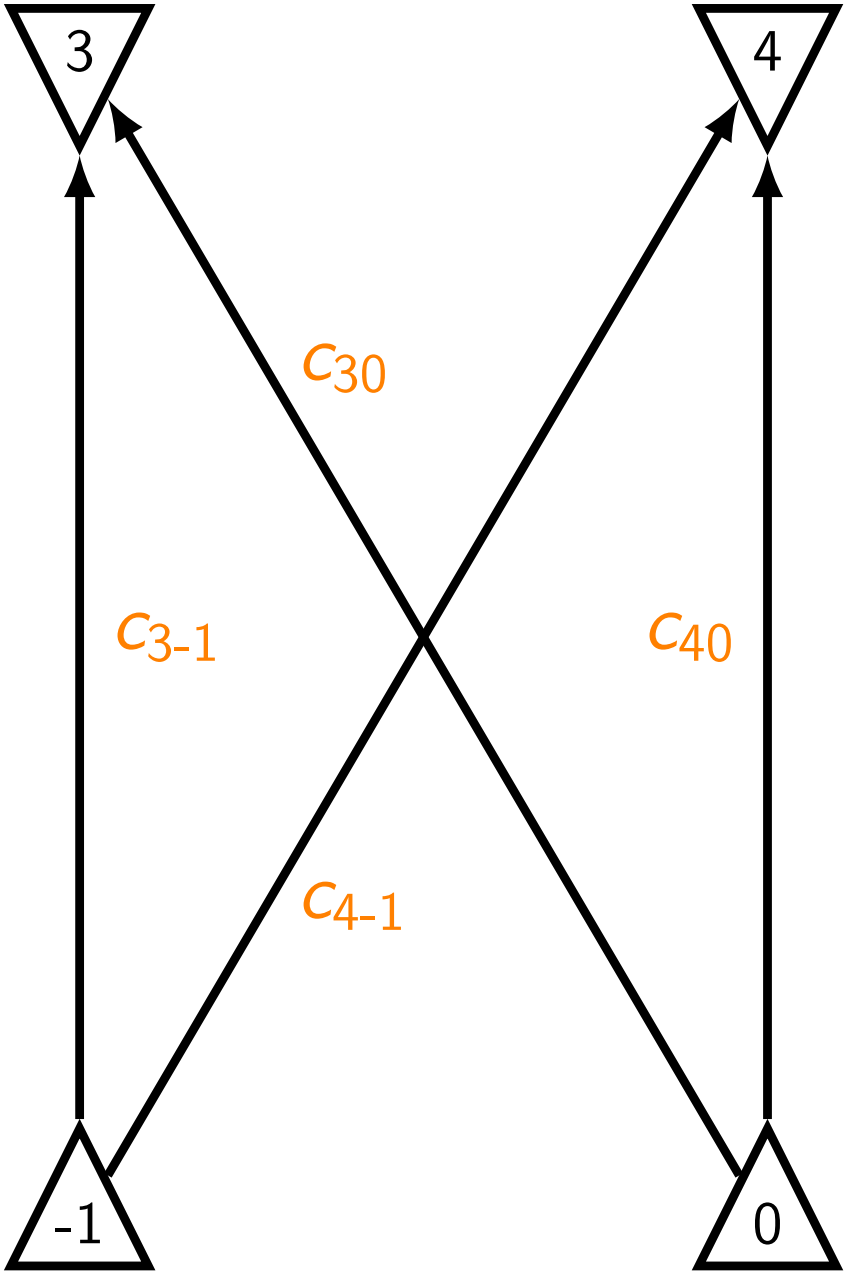
$c_{3-1} = c_{2-1} * c_{32}$

$c_{4-1} = c_{2-1} * c_{42}$

$c_{30} = c_{20} * c_{32}$

$c_{40} = c_{20} * c_{42}$

Preaccumulation



(Code for F, linearization)

$\vdots$

$c_{2-1} = c_{1-1} * c_{21}$

$c_{20} = c_{10} * c_{21}$

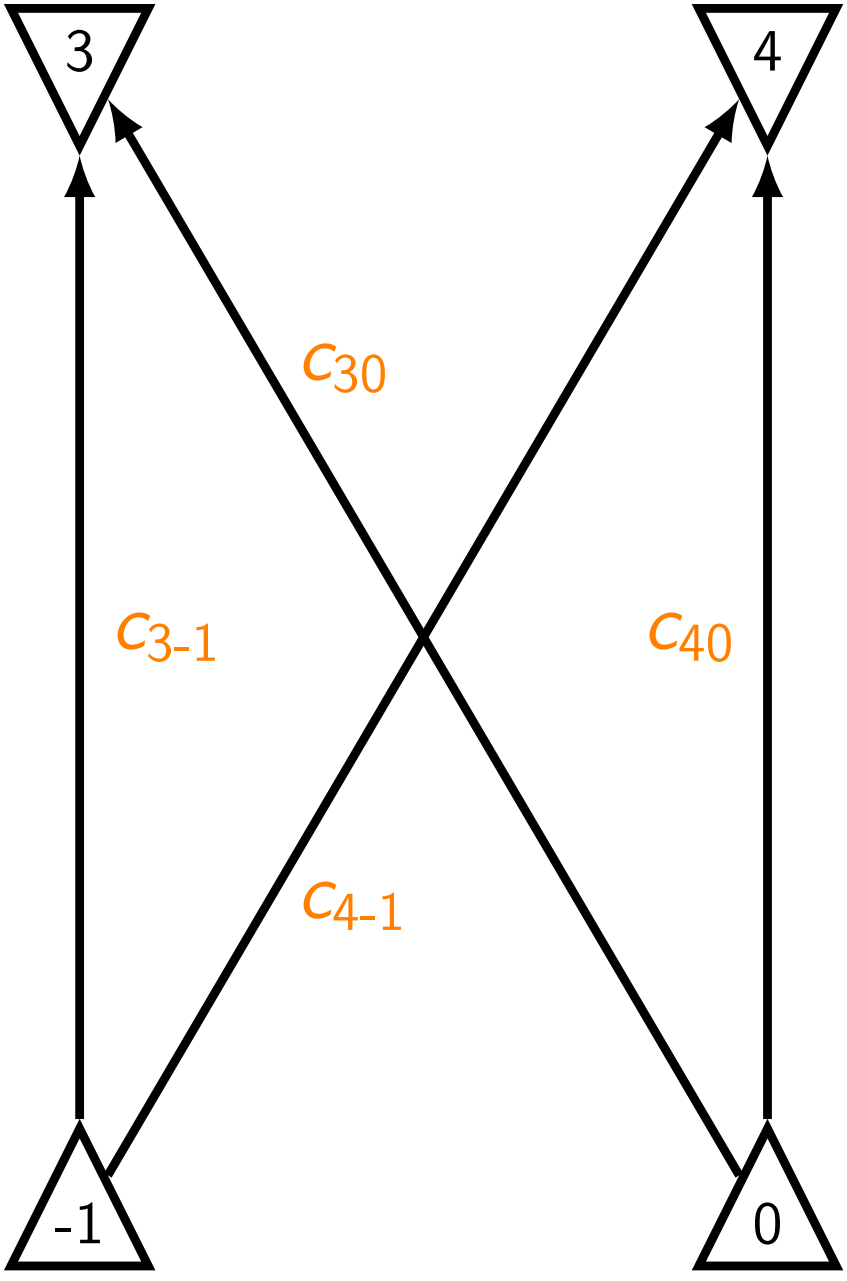
$c_{3-1} = c_{2-1} * c_{32}$

$c_{4-1} = c_{2-1} * c_{42}$

$c_{30} = c_{20} * c_{32}$

$c_{40} = c_{20} * c_{42}$

Preaccumulation



(Code for F, linearization)

$\vdots$

$c_{2-1} = c_{1-1} * c_{21}$

$c_{20} = c_{10} * c_{21}$

$c_{3-1} = c_{2-1} * c_{32}$

$c_{4-1} = c_{2-1} * c_{42}$

4 mults

$c_{30} = c_{20} * c_{32}$

$c_{40} = c_{20} * c_{42}$

$\dot{v}_3 = c_{3-1} * \dot{v}_{-1}$

$\dot{v}_3 \mathrel{+}= c_{30} * \dot{v}_0$

$4p$ mults

$\dot{v}_4 = c_{4-1} * \dot{v}_{-1}$

$\dot{v}_4 \mathrel{+}= c_{40} * \dot{v}_0$

Costs

Fixed costs: Evaluation of F and linearization

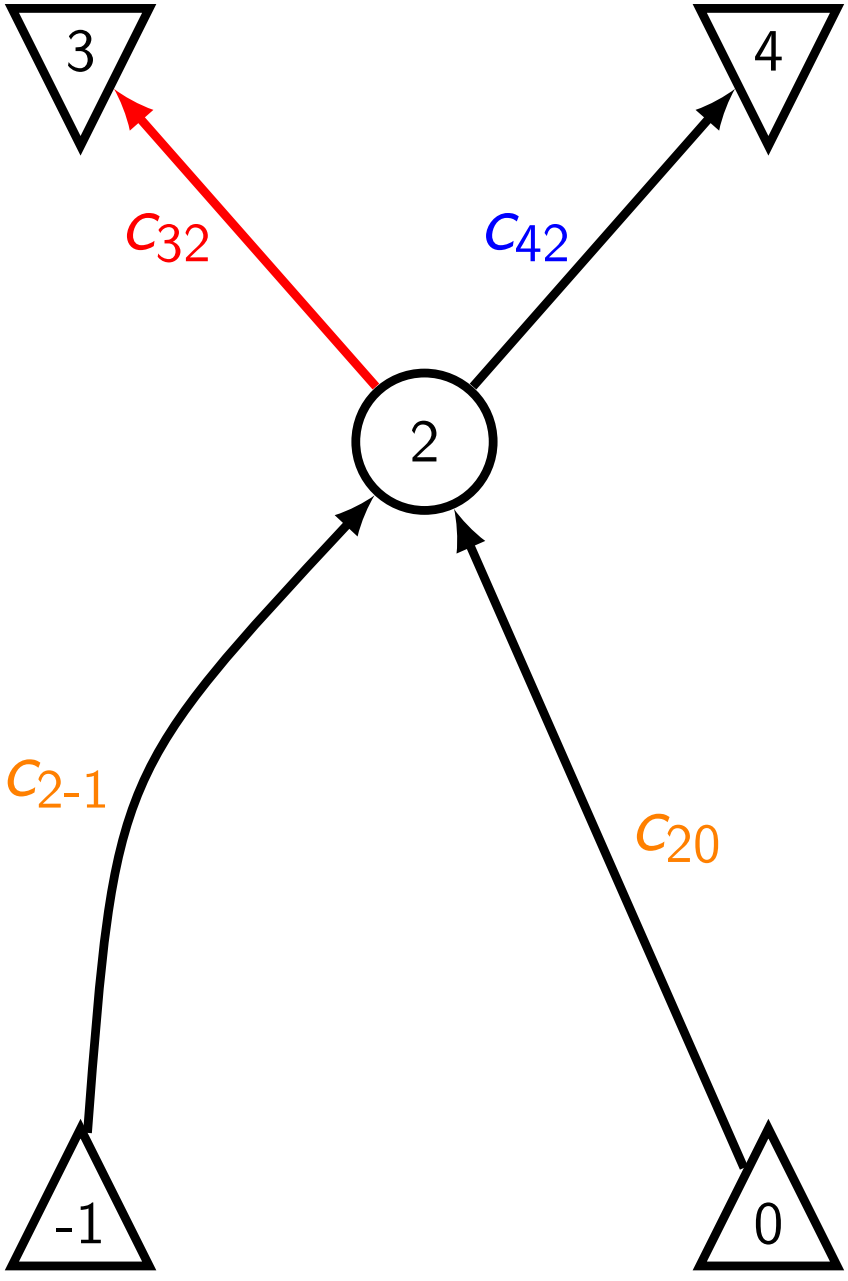
Costs

Fixed costs: Evaluation of F and linearization

Propagation: $5p$ multiplications

Complete Preaccumulation + **Propagation**: $4 + 4p$ multiplications

Partial Preaccumulation



(Code for F, linearization)

$$\vdots$$

$$c_{2-1} = c_{1-1} * c_{21}$$
$$c_{20} = c_{10} * c_{21}$$

2 mults

$$\dot{v}_2 = c_{2-1} * \dot{v}_{-1}$$
$$\dot{v}_2 \mathrel{+}= c_{20} * \dot{v}_0$$
$$\dot{v}_3 = c_{32} * \dot{v}_2 = \dot{v}_2$$
$$\dot{v}_4 = c_{42} * \dot{v}_2$$

$3p$ mults

Costs

Fixed costs: Evaluation of F and linearization

Propagation: $5p$ multiplications

Complete Preaccumulation + **Propagation**: $4 + 4p$ multiplications

Partial Preaccumulation + **Propagation**: $2 + 3p$ multiplications

$\Rightarrow$ Assume p is large, so ignore preaccumulation cost and focus on propagation cost.

Outline

Jacobian Scarsity

Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.
For example, the Jacobian is low rank (# of vertex-disjoint paths).

$\Rightarrow$ Jacobian **scarsity** (Griewank)

Jacobian Scarsity

Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.
For example, the Jacobian is low rank (# of vertex-disjoint paths).

$\Rightarrow$ Jacobian **scarsity** (Griewank)

Scarsity is a kind of deficiency, approximated by the number of nonunit edges in G
Graph transformations that *don't increase* the number of nonunit edges are said to be **scarsity preserving**.

Jacobian Scarsity

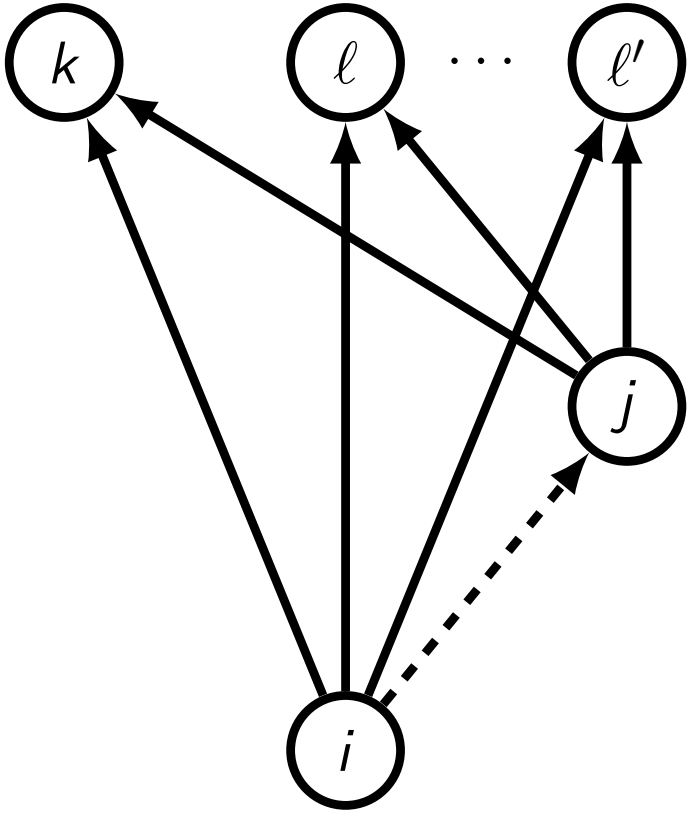
Example Jacobian happens to be dense, but some structure is lost when $F'(\mathbf{x})$ is accumulated to a matrix.
For example, the Jacobian is low rank (# of vertex-disjoint paths).

$\Rightarrow$ Jacobian **scarsity** (Griewank)

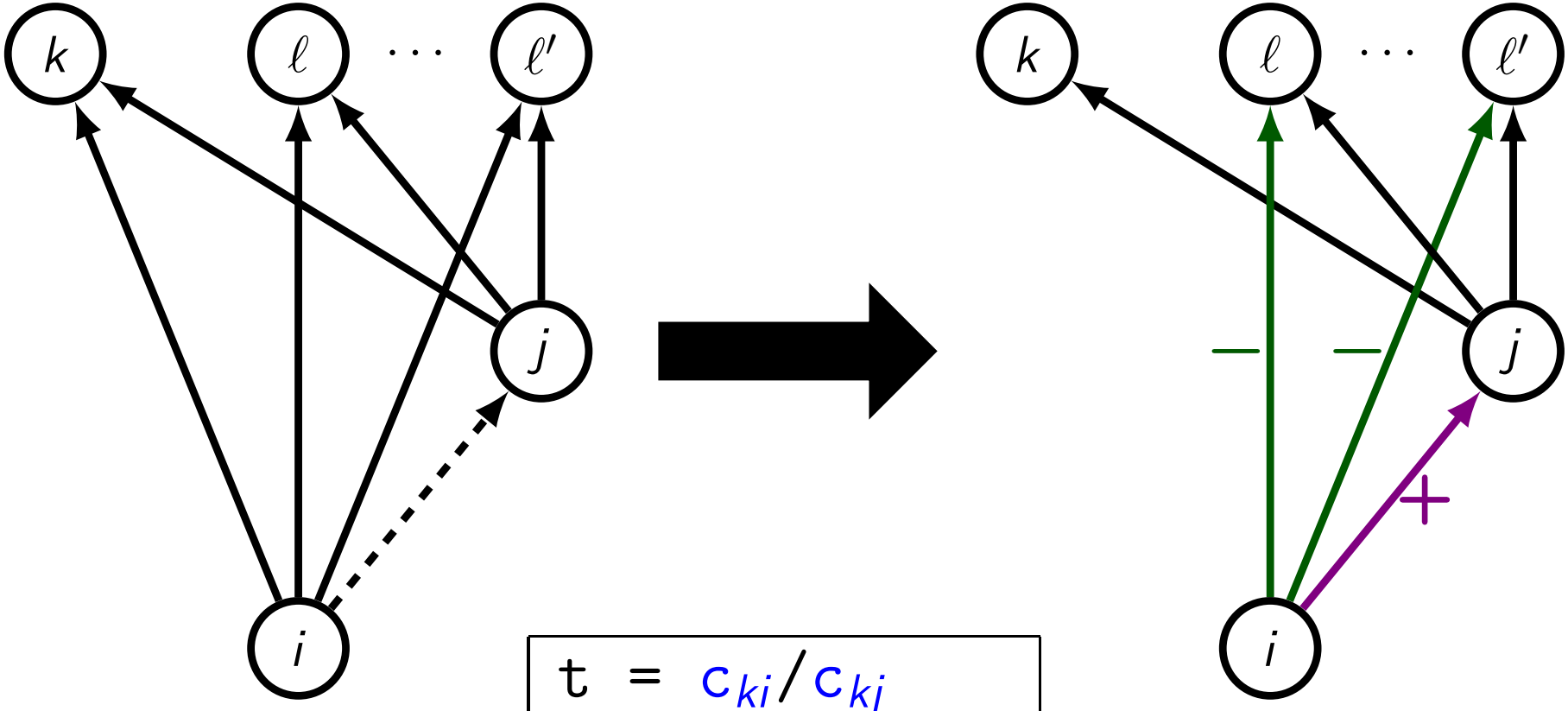
Scarsity is a kind of deficiency, approximated by the number of nonunit edges in G
Graph transformations that *don't increase* the number of nonunit edges are said to be **scarsity preserving**.

$\Rightarrow$ Exploiting Scarsity: finding minimal representation of $G(F'(\mathbf{x}))$

Edge **Prerouting**

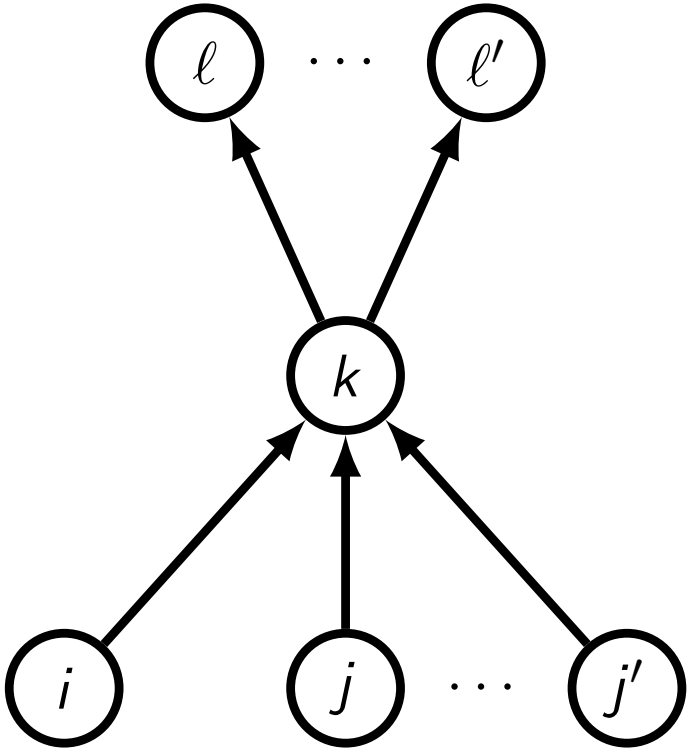


Edge **Prerouting**

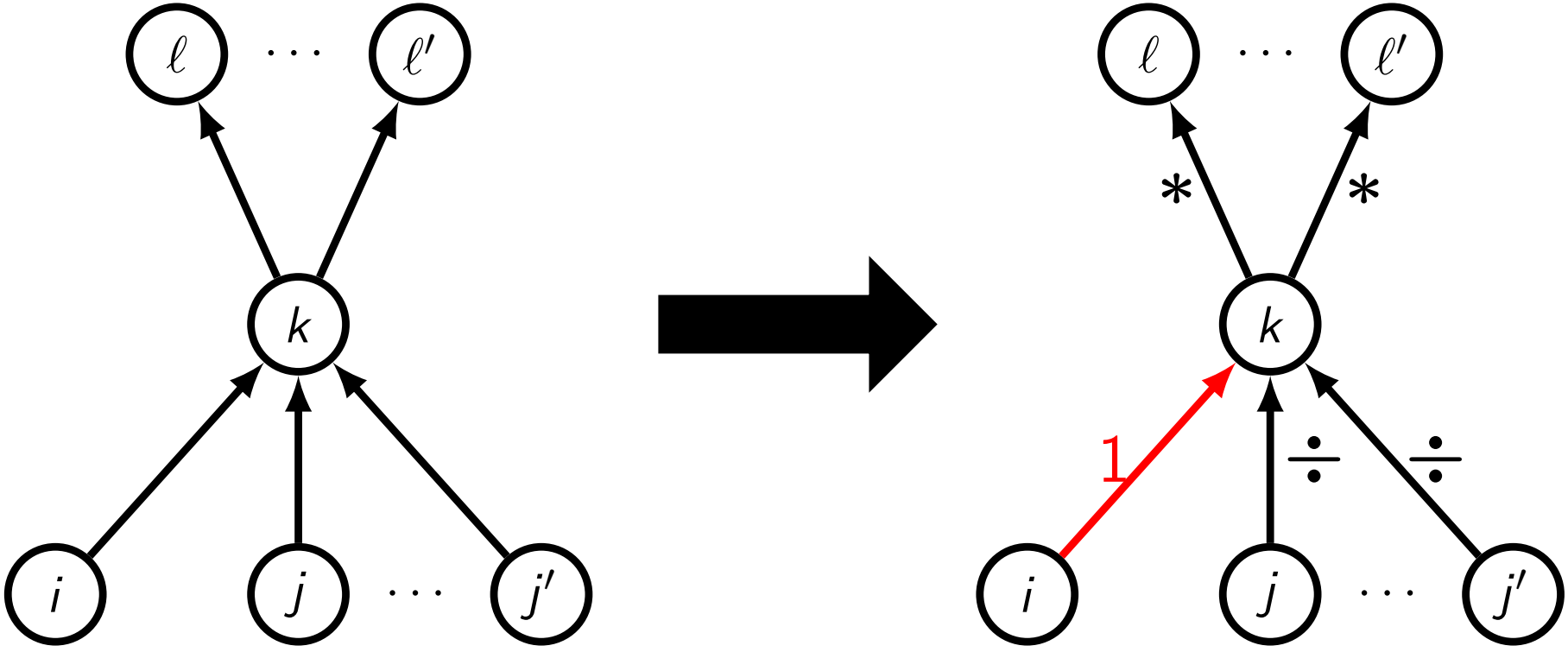



```
t = c_ki / c_kj
c_ji  += t
c_ℓi  -= c_ℓj * t
   ⋮
c_ℓ′i -= c_ℓ′i * t
```

Edge **Normalization** Forward

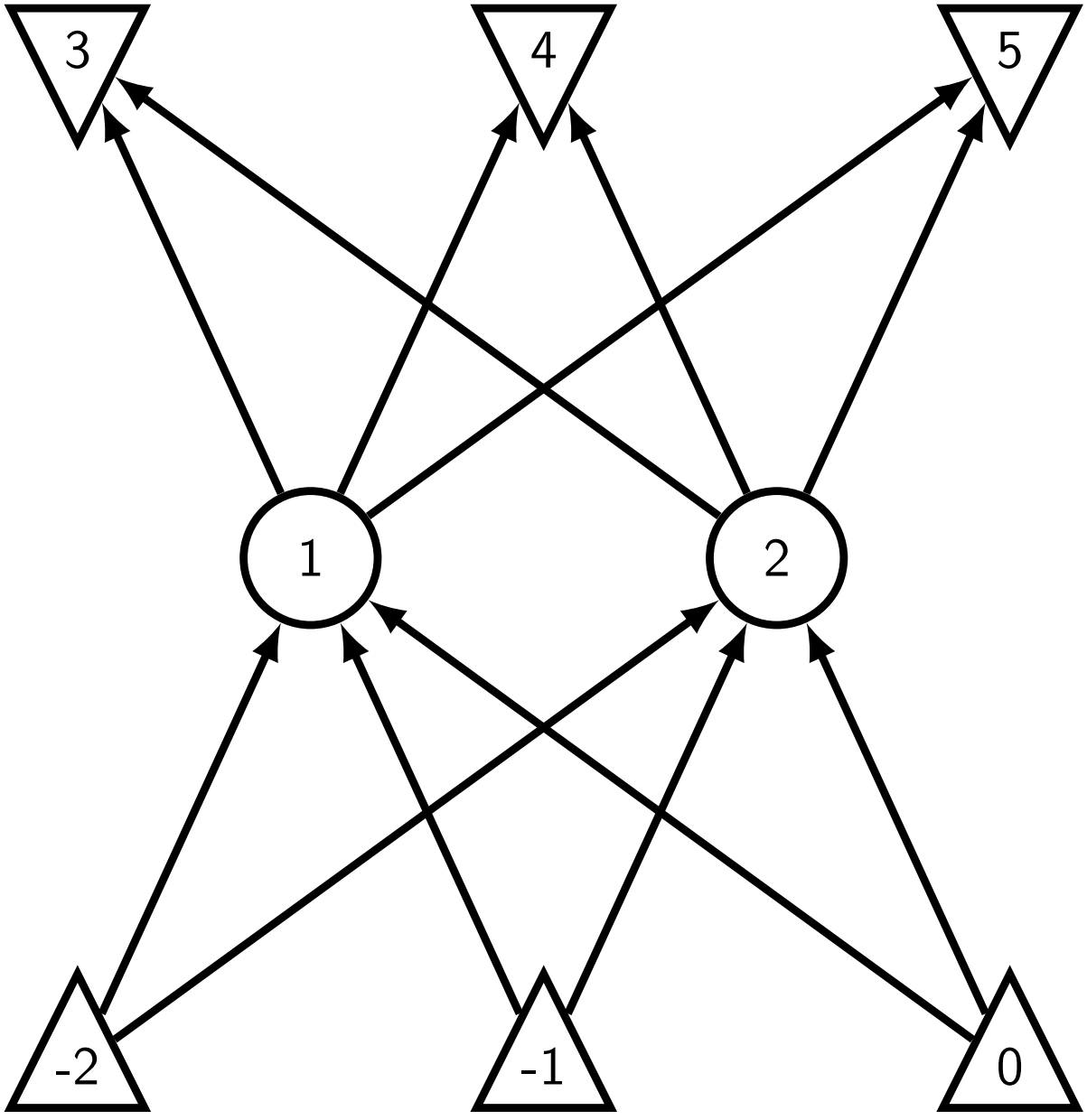


Edge **Normalization** Forward

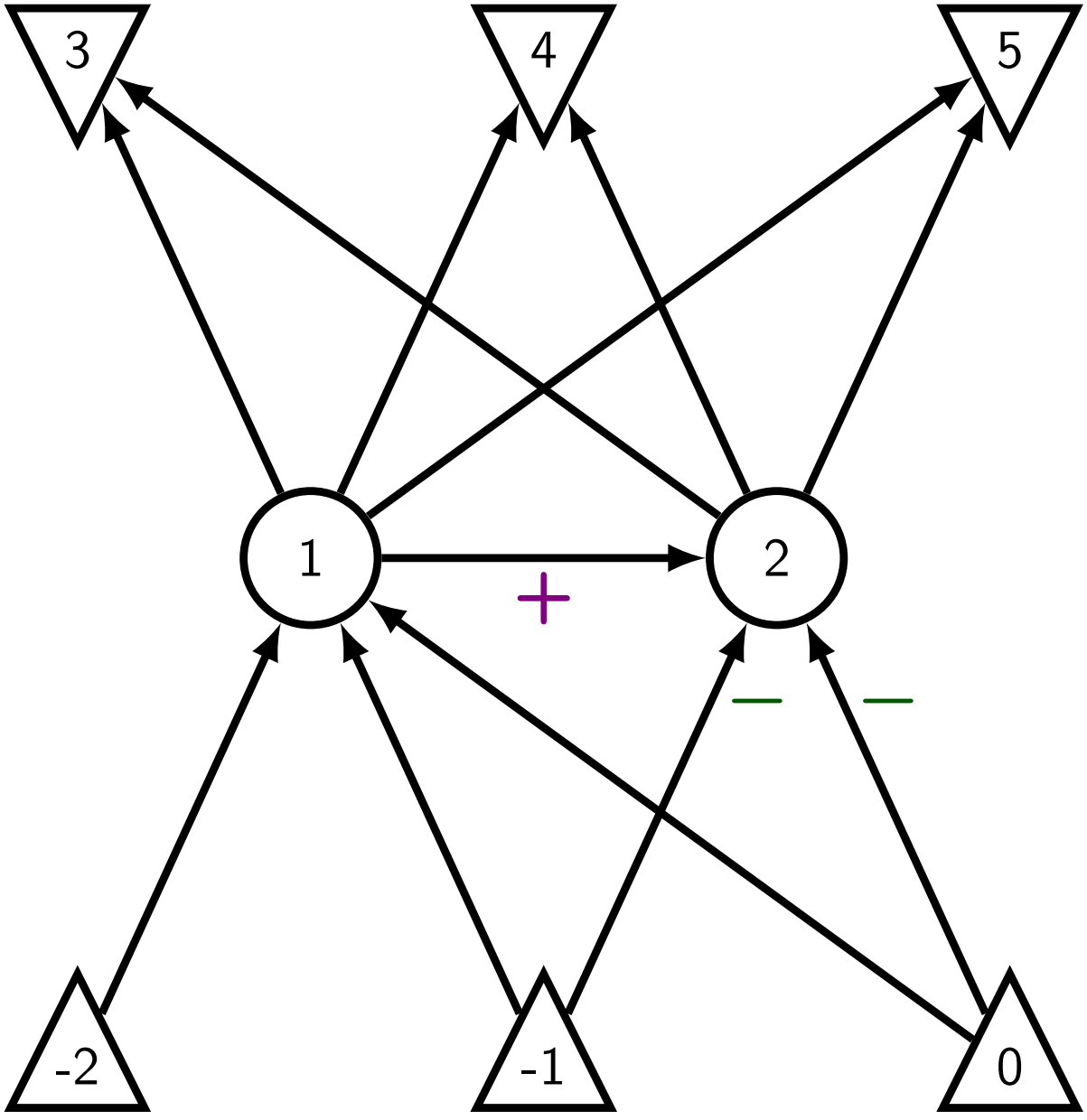


$$
\begin{array}{ll}
c_{kj} \;/\!= c_{ki} & c_{\ell k} \;*\!= c_{ki} \\
\vdots & \vdots \\
c_{kj'} \;/\!= c_{ki} & c_{\ell' k} \;*\!= c_{ki} \\
c_{ki} = 1 &
\end{array}
$$

Example

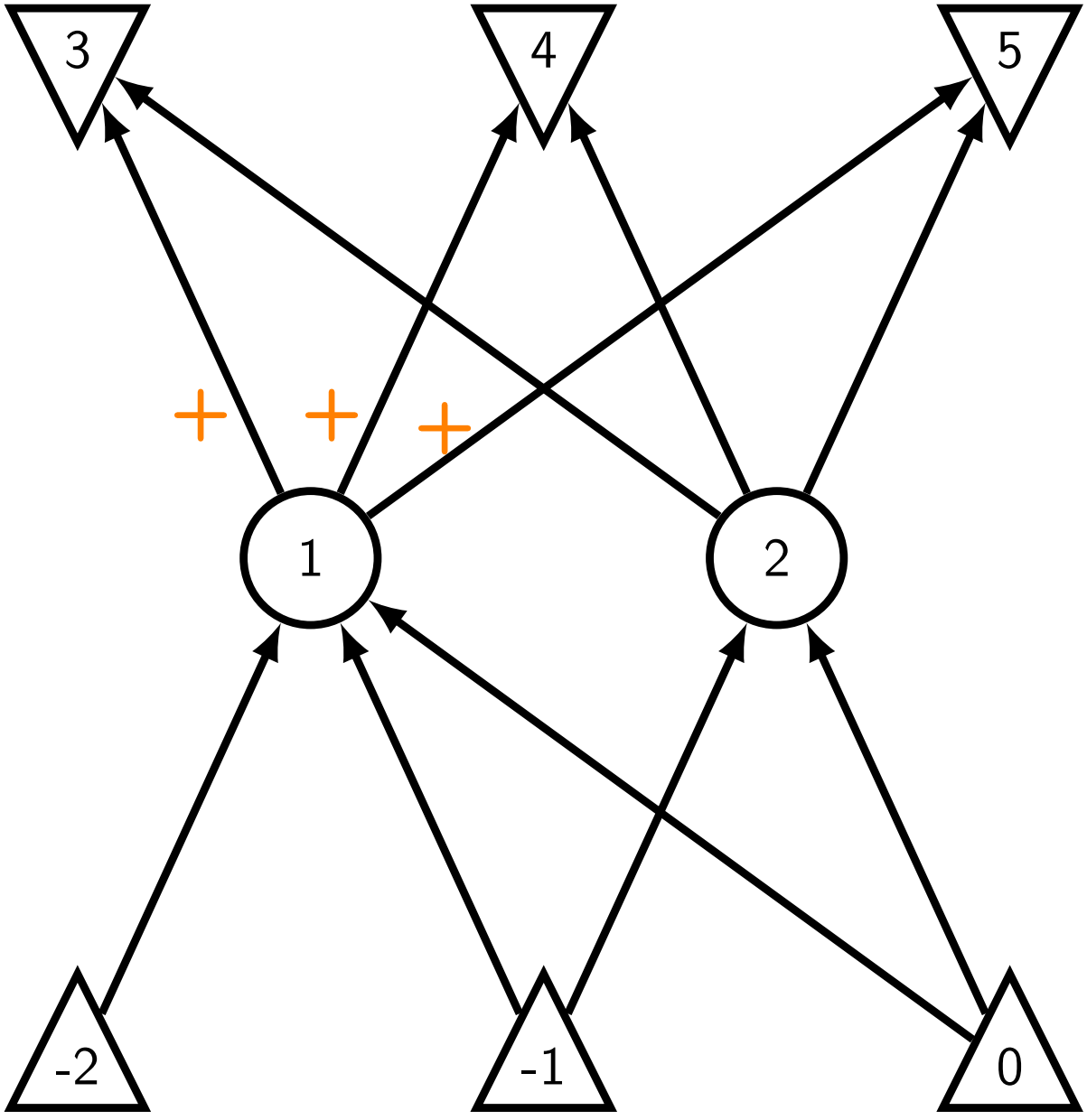


Example



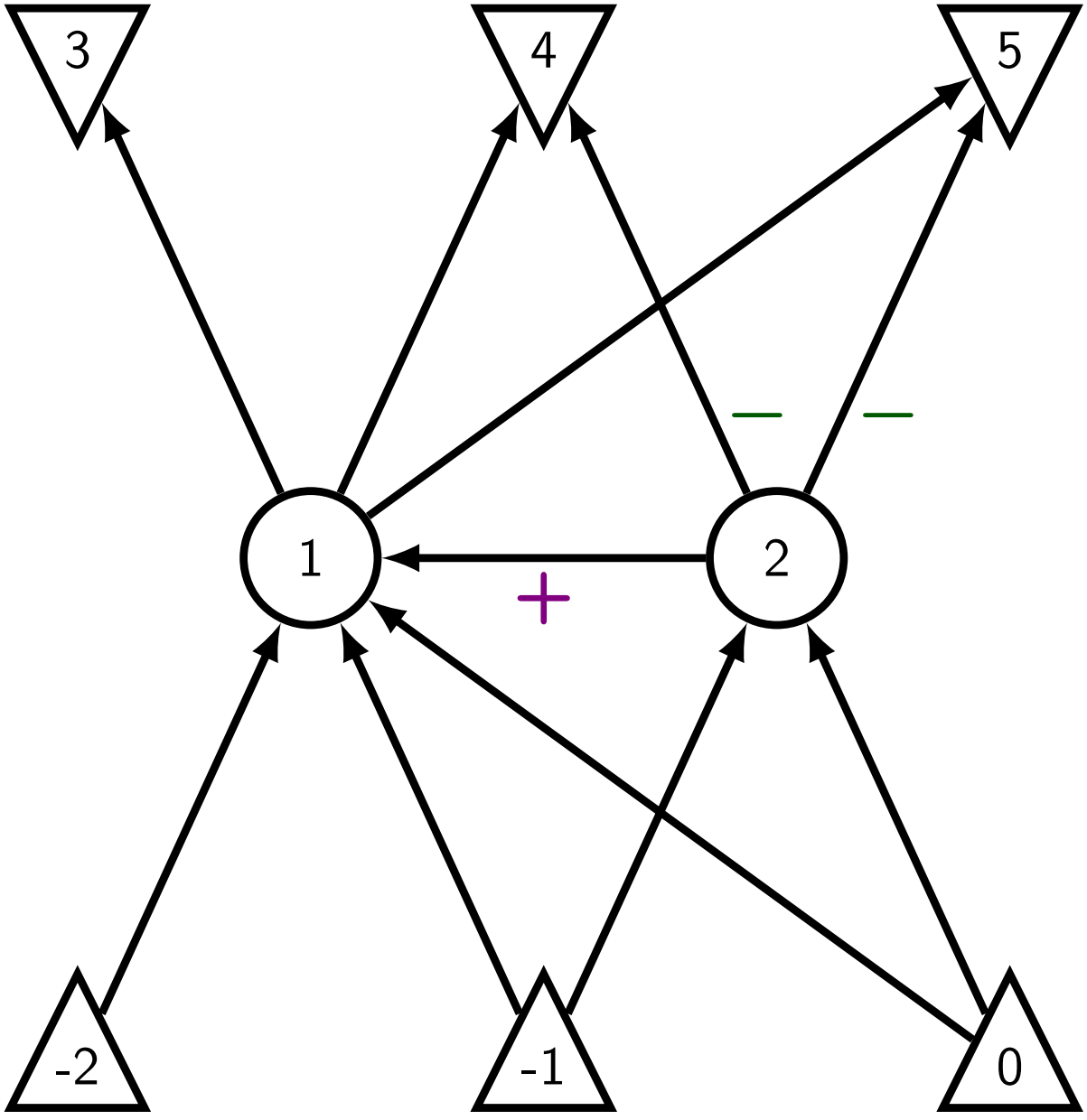
- Postroute (-2, 2)

Example



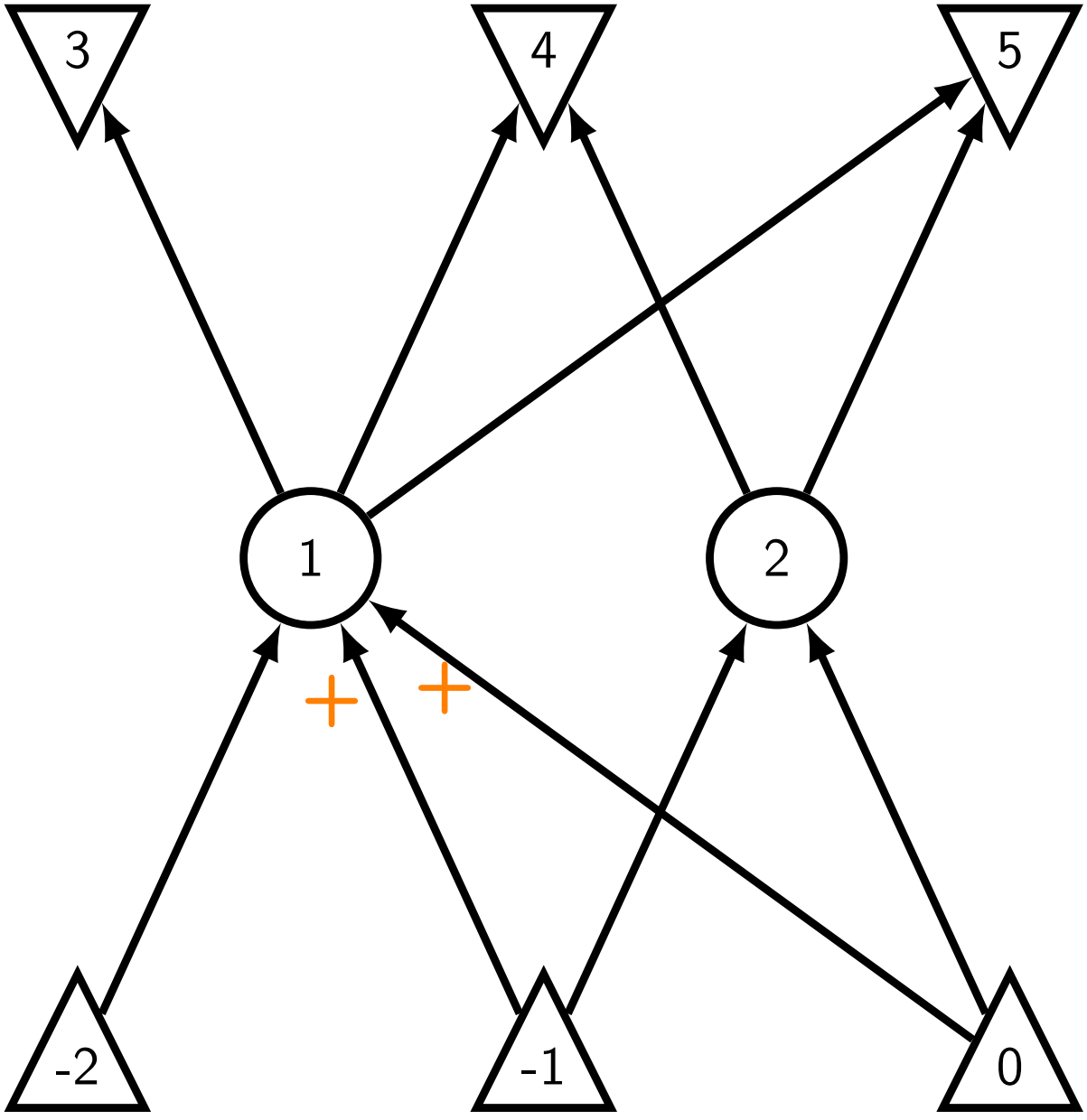
- Postroute $(-2, 2)$
- Front Eliminate $(1, 2)$

Example



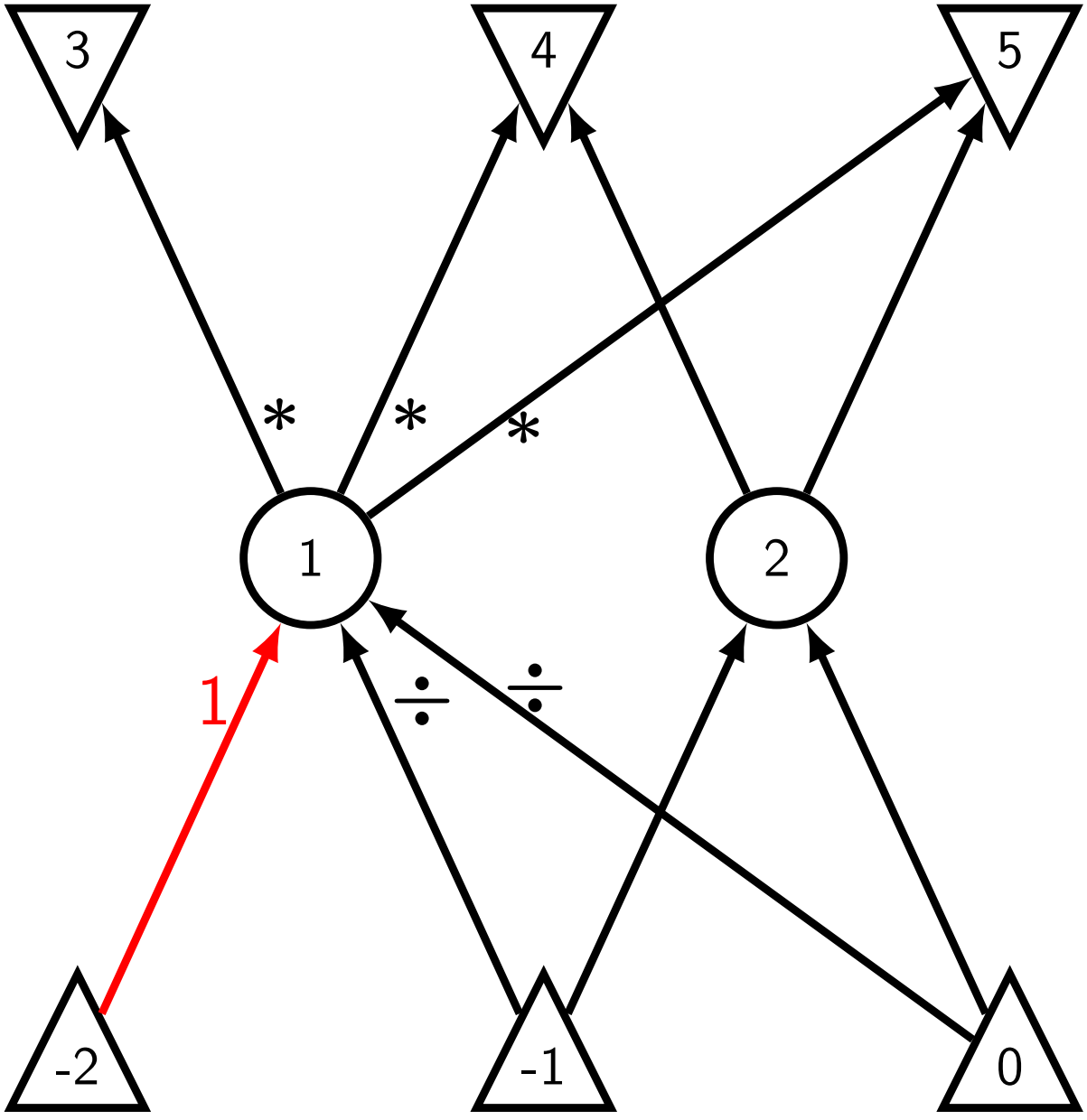
- Postroute $(-2, 2)$
- Front Eliminate $(1, 2)$
- Preroute $(2, 3)$

Example



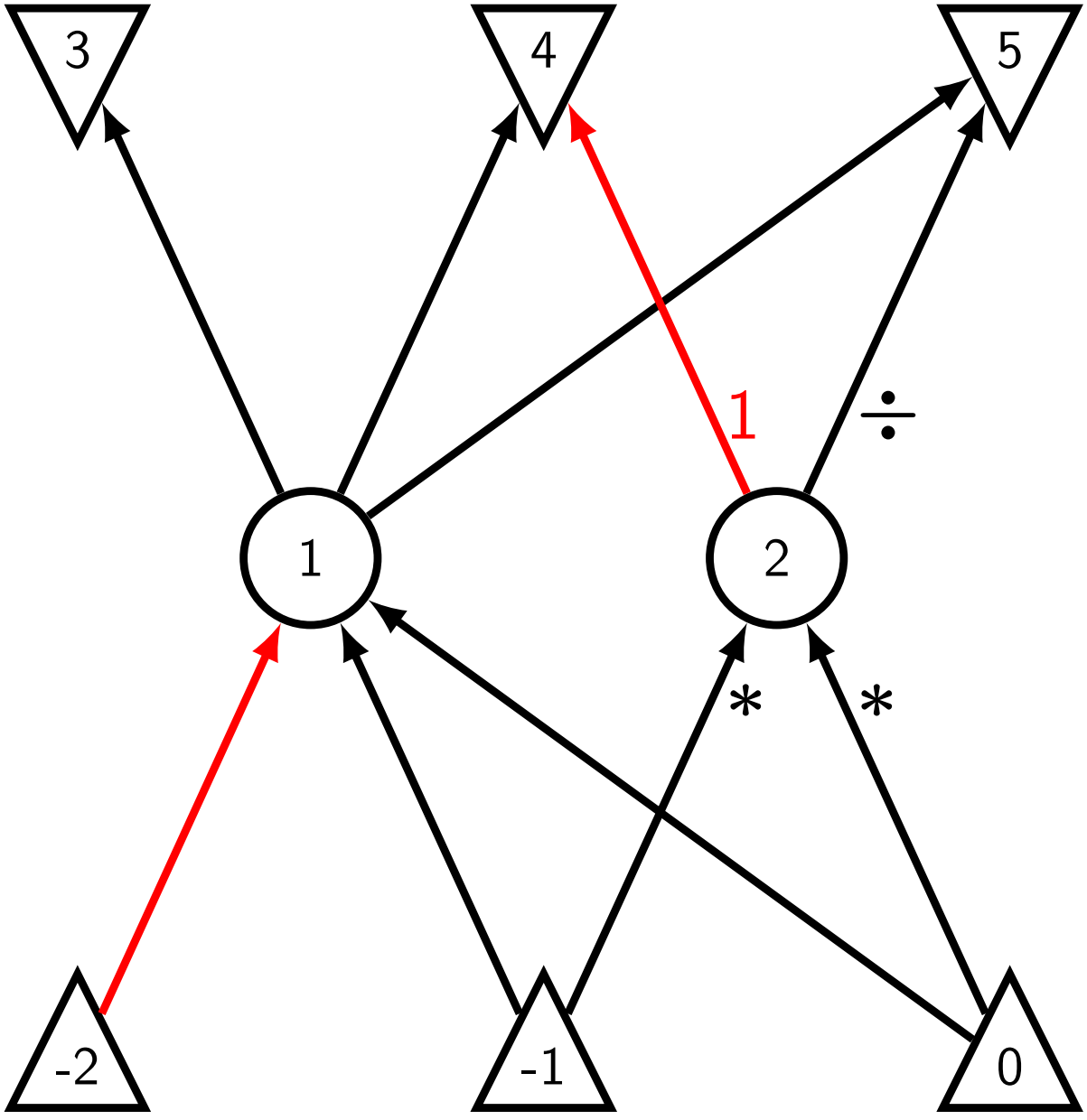
- Postroute $(-2, 2)$
- Front Eliminate $(1, 2)$
- Preroute $(2, 3)$
- Back Eliminate $(2, 1)$

Example



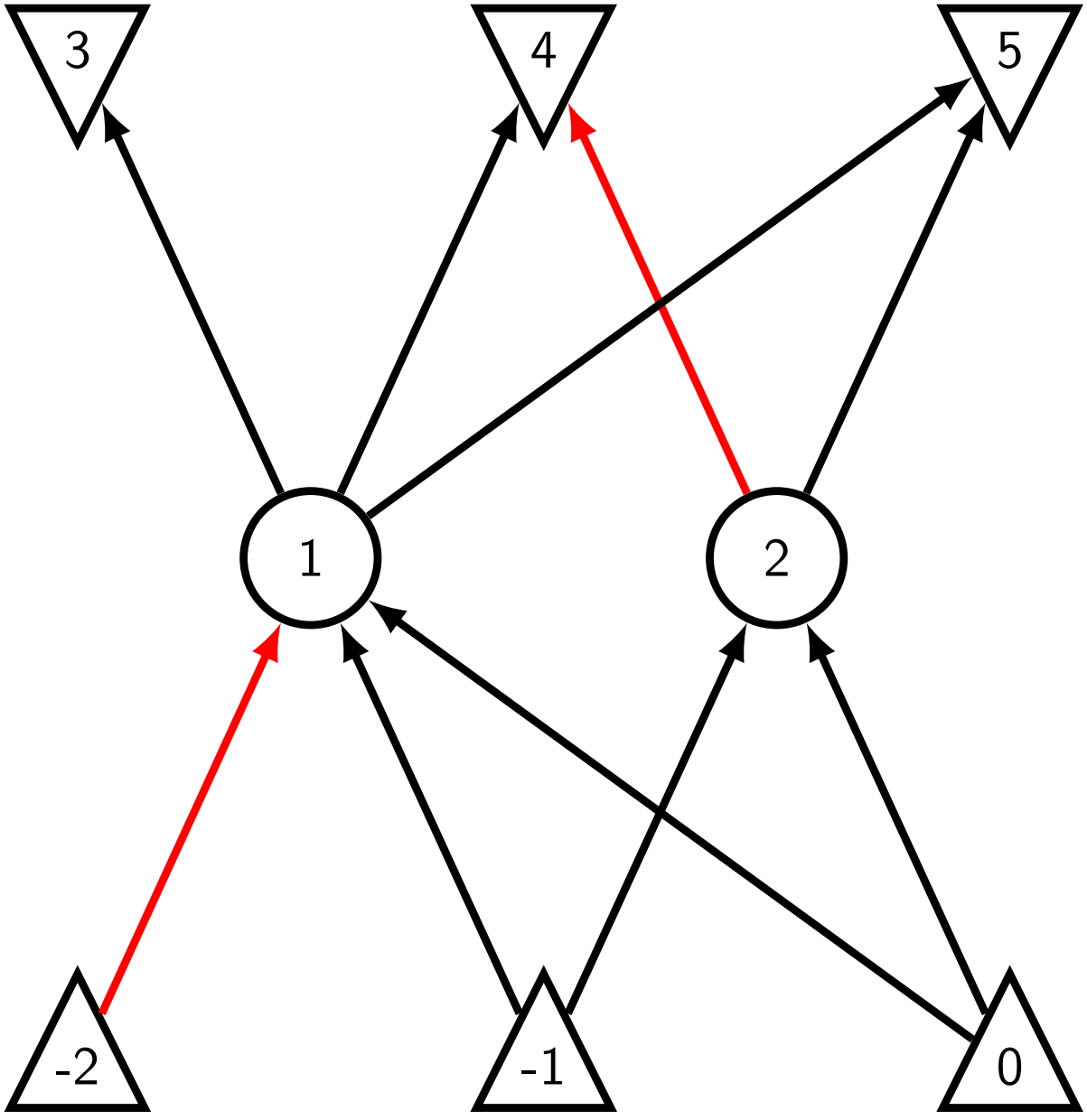
- Postroute $(-2, 2)$
- Front Eliminate $(1, 2)$
- Preroute $(2, 3)$
- Back Eliminate $(2, 1)$
- Normalize $(-2, 1)$

Example



- Postroute (-2, 2)
- Front Eliminate (1, 2)
- Preroute (2, 3)
- Back Eliminate (2, 1)
- Normalize (-2, 1)
- Normalize (2, 4)

Example

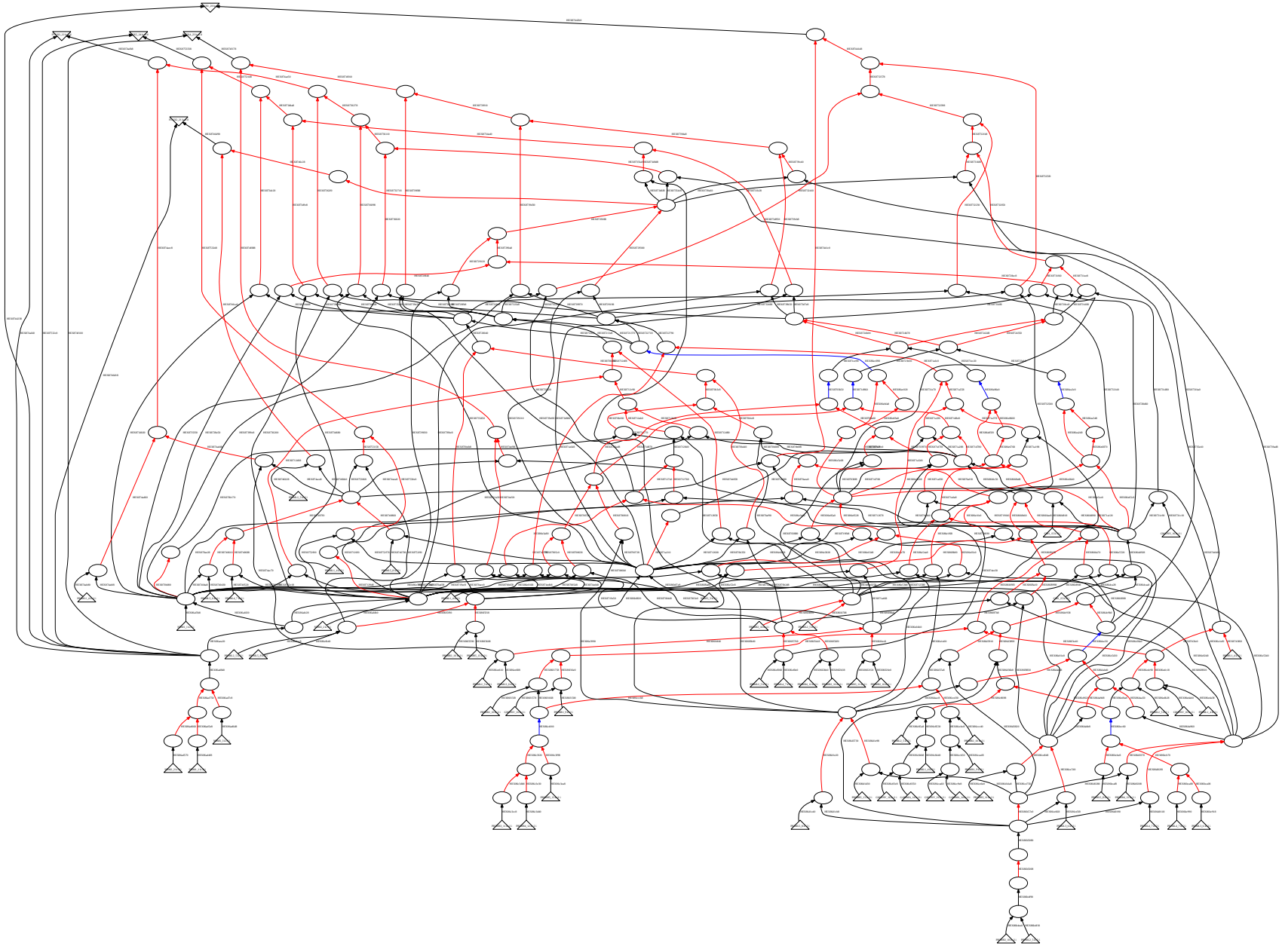


- Postroute (-2, 2)
- Front Eliminate (1, 2)
- Preroute (2, 3)
- Back Eliminate (2, 1)
- Normalize (-2, 1)
- Normalize (2, 4)

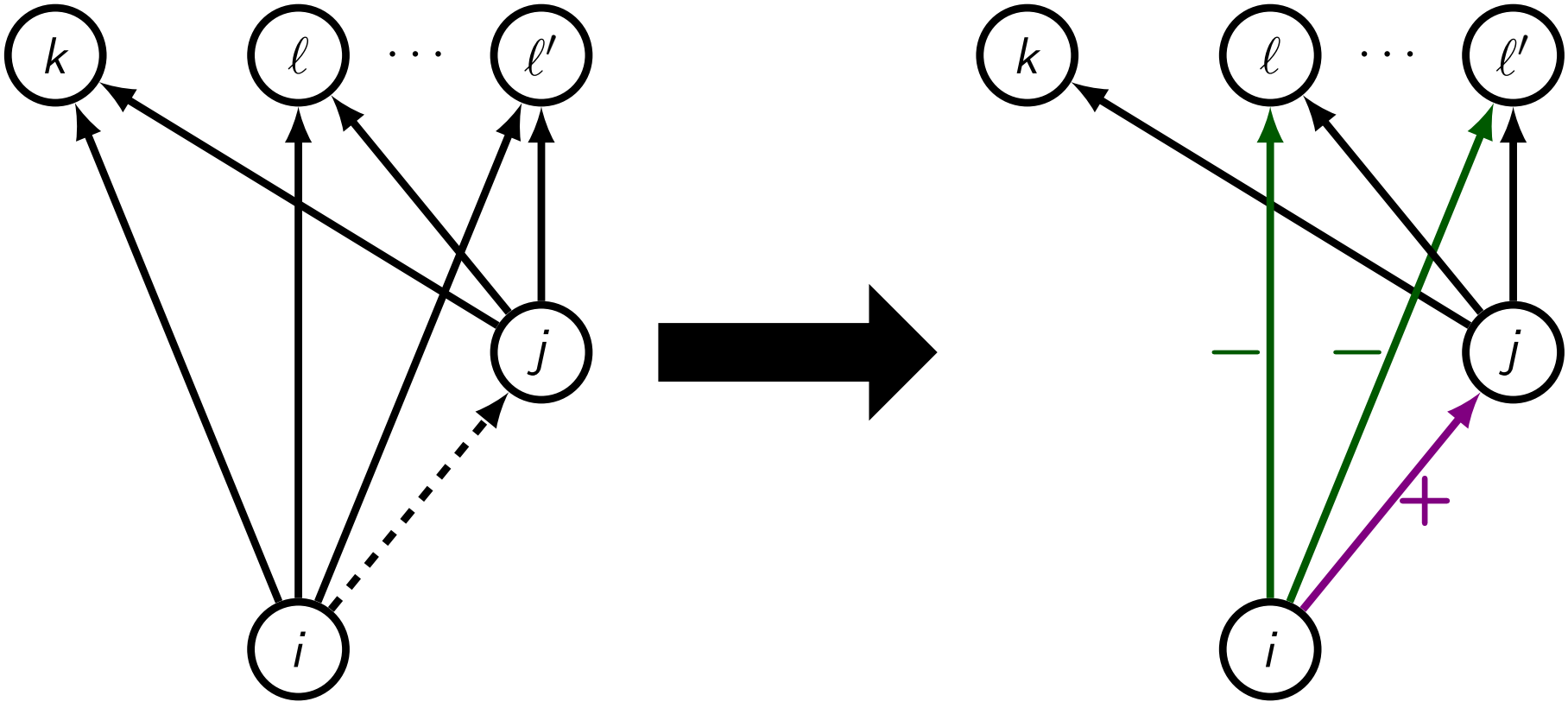
$\Rightarrow$ 8 nonunit edges

Outline

Scarsity in Practice

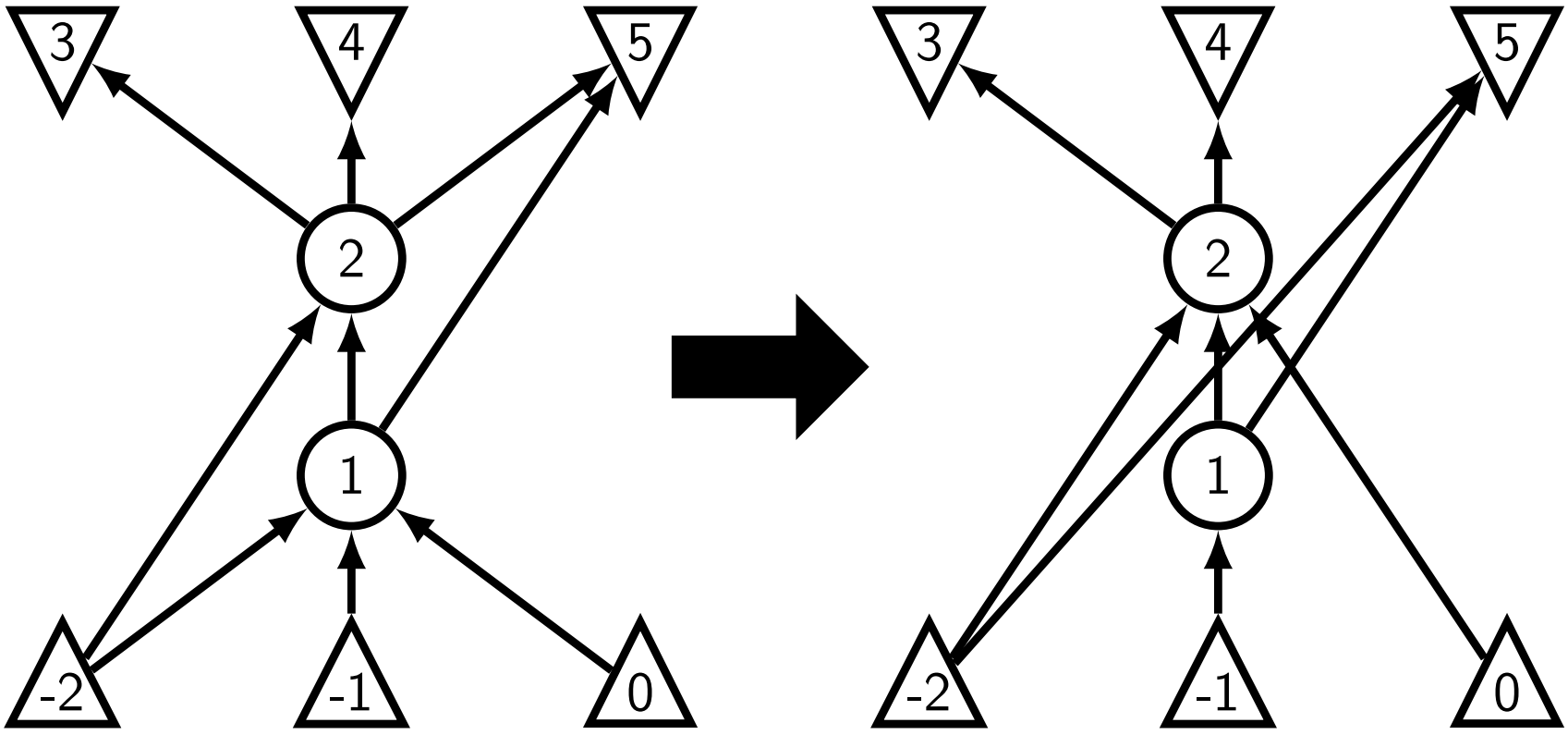


Edge **Prerouting**



Observations: Vertex vs. Edge Elimination

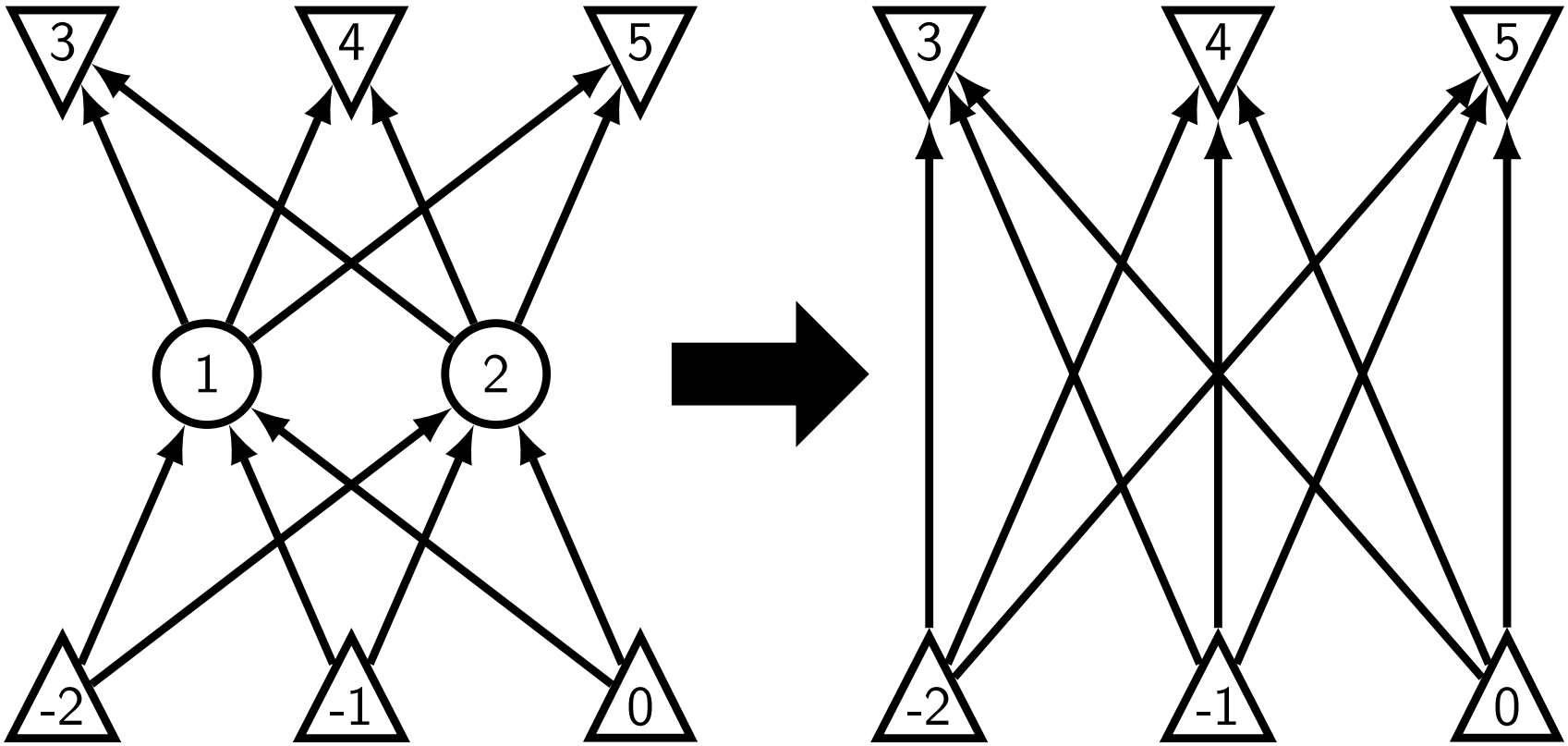
Vertex elimination is not fine-grained enough to get the minimal representation



$\Rightarrow$ Use edge elimination
(or perhaps face elimination is necessary. . . ?)

Observations: Monotonicity

Pure edge elimination: we don't have **monotonicity**
(the count may have to go up before going down)



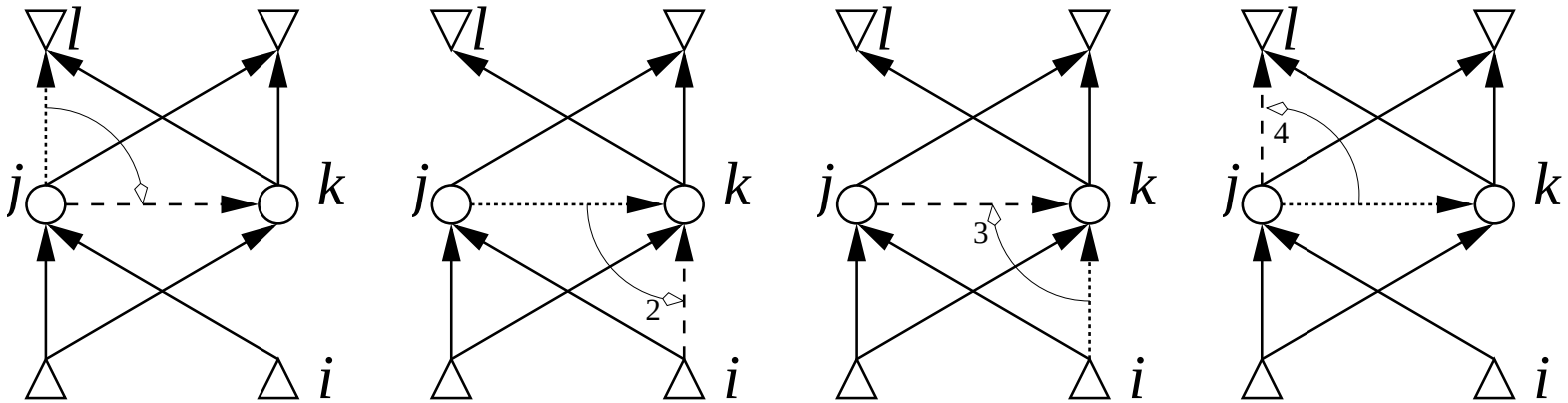
⇒ Perform **complete** edge elimination sequence,
only generate code up to minimal count reached

Observations: Termination

Guaranteed for pure edge elimination. This is **not** the case when we include reroutings and normalizations.

Observations: Termination

Guaranteed for pure edge elimination. This is **not** the case when we include reroutings and normalizations.



The Heuristic: Concerns

Prefer edge eliminations (no divisions). Otherwise...

The Heuristic: Concerns

Prefer edge eliminations (no divisions). Otherwise...

Reroutings:

Can reduce iff **increment** edge is present and **nonunit**
$\Rightarrow$ In this case, there must be an edge elim. that reduces count

The Heuristic: Concerns

Prefer edge eliminations (no divisions). Otherwise...

Reroutings:

Can reduce iff **increment** edge is present and **nonunit**
$\Rightarrow$ In this case, there must be an edge elim. that reduces count

Pair with subsequent edge elim. for **overall** edge count reduction
Combine with restriction that no particular rerouting can be re-performed
$\Rightarrow$ ensures **termination**

The Heuristic: Concerns

Prefer edge eliminations (no divisions). Otherwise...

Reroutings:

Can reduce iff **increment** edge is present and **nonunit**
⇒ In this case, there must be an edge elim. that reduces count

Pair with subsequent edge elim. for **overall** edge count reduction
Combine with restriction that no particular rerouting can be re-performed
⇒ ensures **termination**

Normalizations:

Can't lead to subsequent transformations that reduce count
⇒ perform as post-processing step, once per intermediate vertex

The Heuristic: Refill Awareness

Refill: Eliminating an edge, then subsequently re-creating it as fill

"No free refill" conjecture (Naumann): There is an optimal edge elimination sequence (ops) that produces no refill

The Heuristic: Refill Awareness

Refill: Eliminating an edge, then subsequently re-creating it as fill

"No free refill" conjecture (Naumann): There is an optimal edge elimination sequence (ops) that produces no refill

Check whether e can be refilled: Is there an alternative path from the source to the target?
Check in context of reroutings: Not entirely clear.

The Heuristic: Refill Awareness

Refill: Eliminating an edge, then subsequently re-creating it as fill

"No free refill" conjecture (Naumann): There is an optimal edge elimination sequence (ops) that produces no refill

Check whether e can be refilled: Is there an alternative path from the source to the target?
Check in context of reroutings: Not entirely clear.

Idea: If our heuristic creates refill, rerun the heuristic with this extra information

The Heuristic

input : Linearized computational graph G
output: A sequence of transformations for turning G into the remainder graph G'

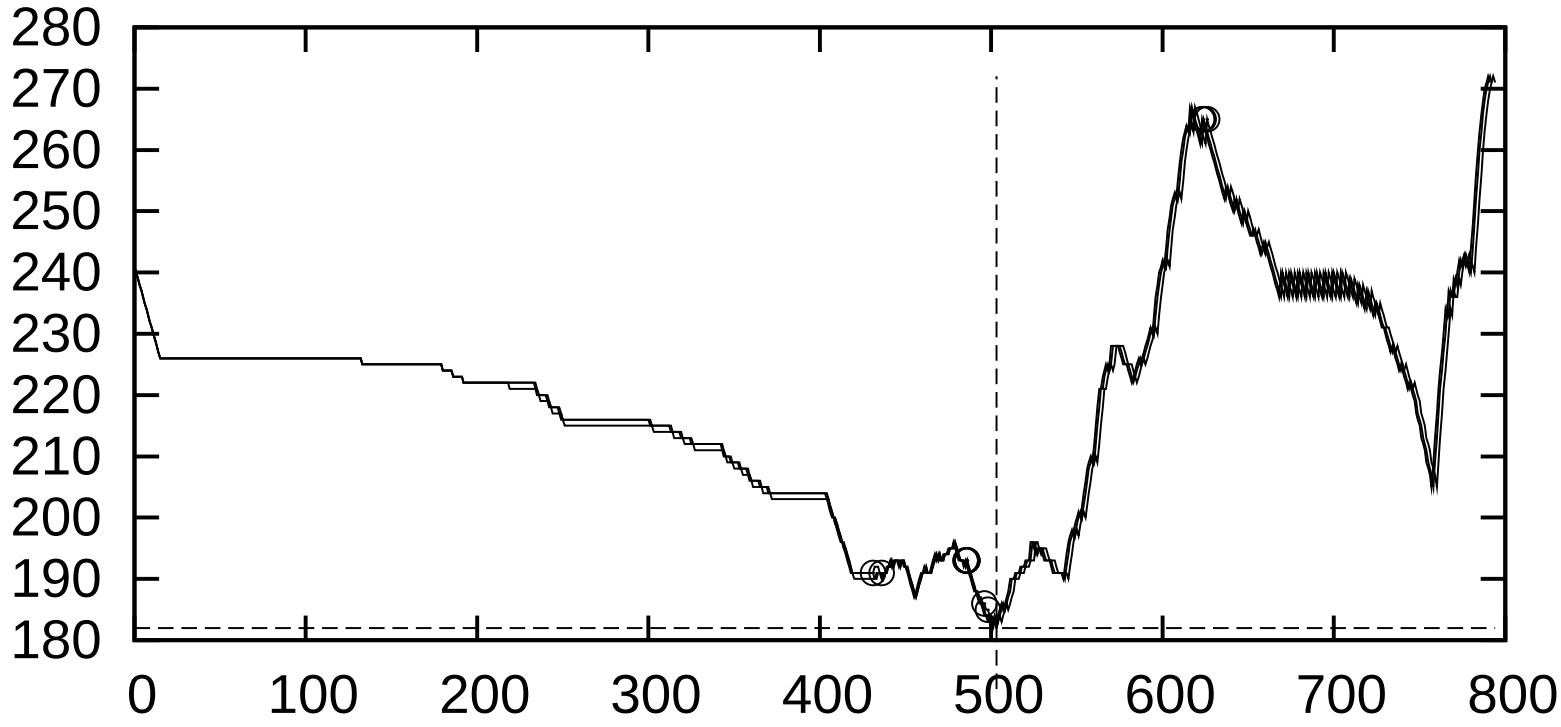
```
while new refill information do
    while ∃ possible transformations do
        if ∃ edge elimination that reduces count then
            eliminate an edge
        else if ∃ elim.-reroute that reduces count then
            Prefer bigger reduction;
            tiebreak (reverse);
        else
            eliminate an edge
        end
    end
end
Find minimal point
Perform normalizations
```

Choosing an Edge Elimination

1. Biggest reduction in nonunit edge count
2. Avoid refill
3. Minimal ops (Markowitz degree)
4. Tiebreak (reverse)

Results

Generally: Reroutings don't help much, few performed

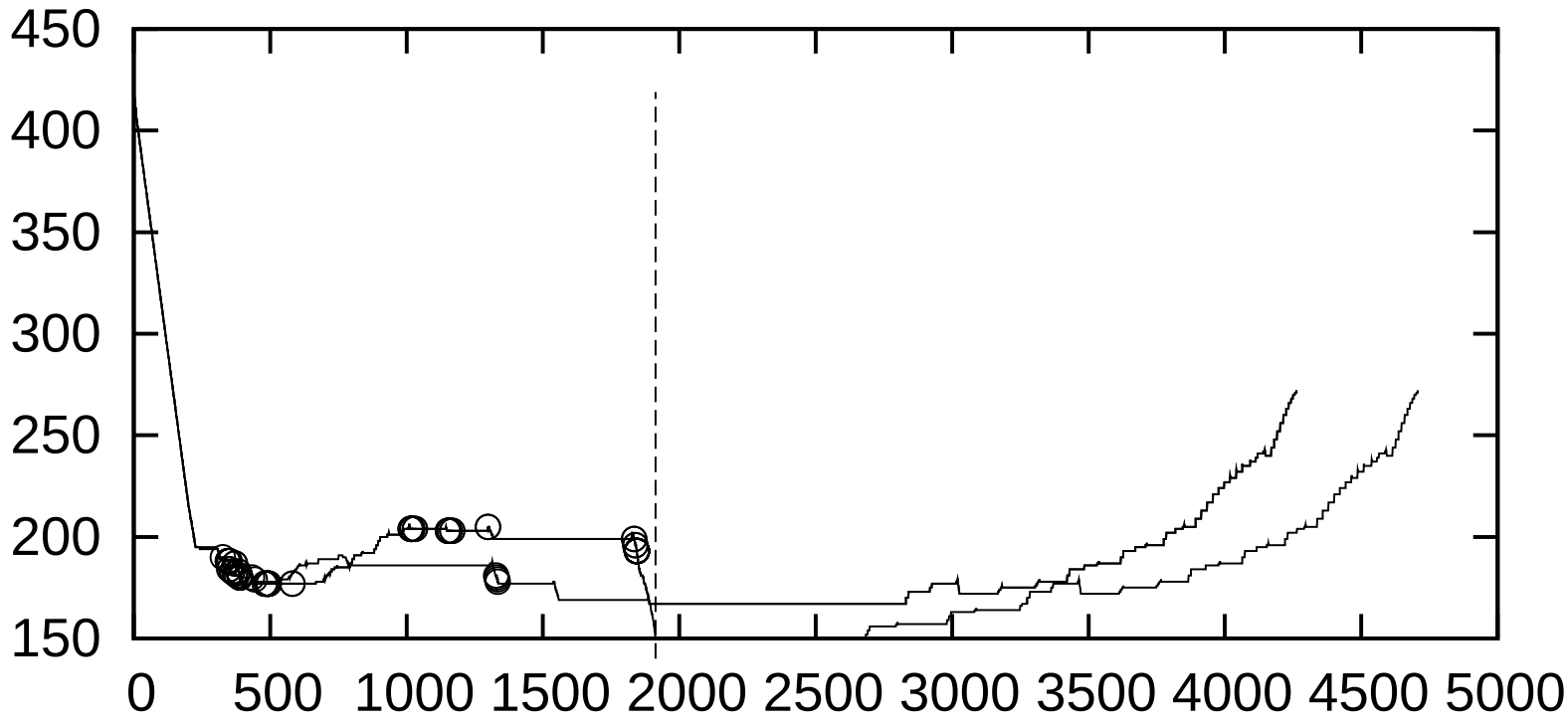


26 reroutings performed

Normalization: 8 intermediates with no incident unit edge

Results

Just for kicks: Allowing reroutings that can't be followed by an edge elim. that reduces nonunit edge count



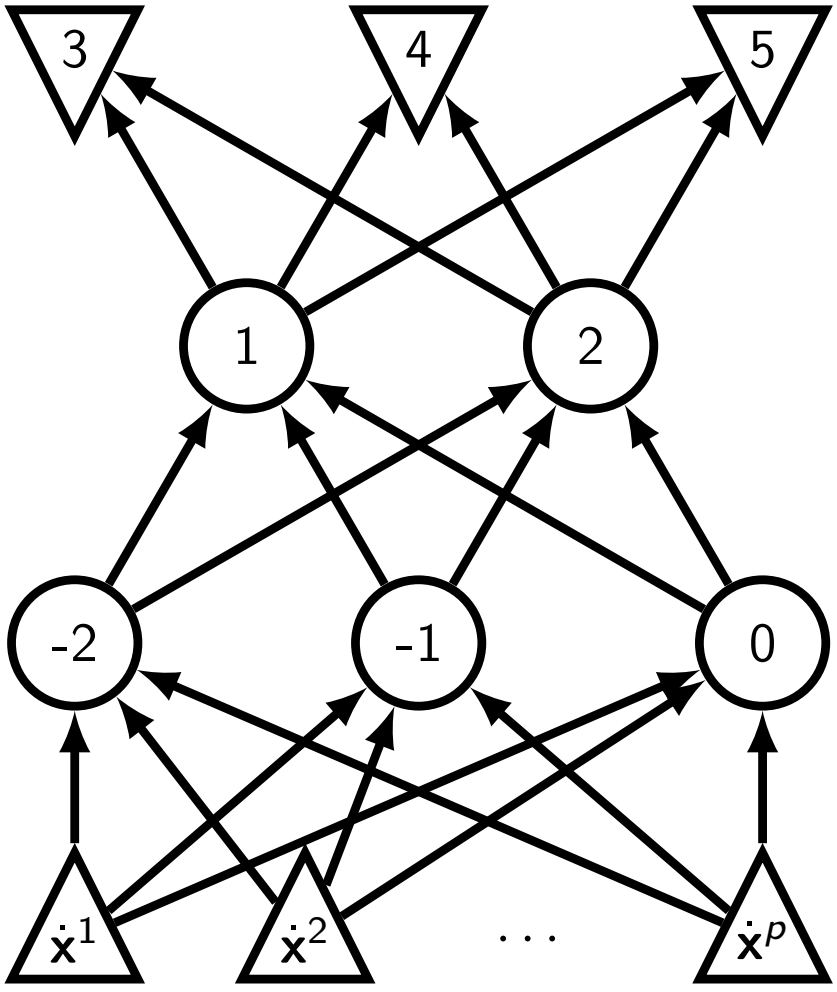
Performs 768 reroutings

Outline

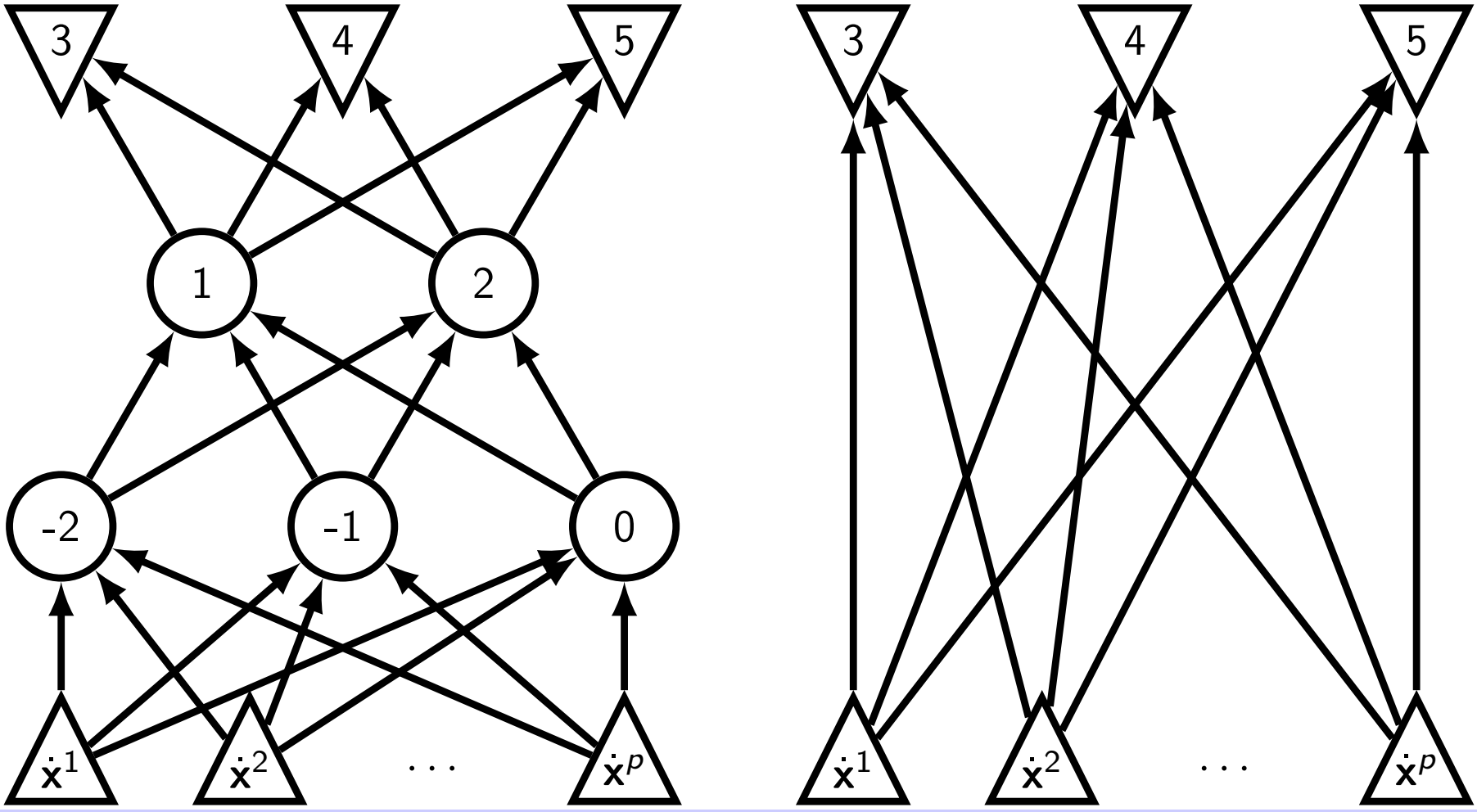
Future Work

- Run-time check for divisions, with “safe” sequence as alternative
- Complexity issues
  - Minimum Jacobian representation by edge elimination?
  - What about when we include reroutings and normalizations?
- Better heuristics
  - Randomized heuristic (like simulated annealing)
  - ...

Scarsity $\leftrightarrow$ Optimal Jacobian Accumulation

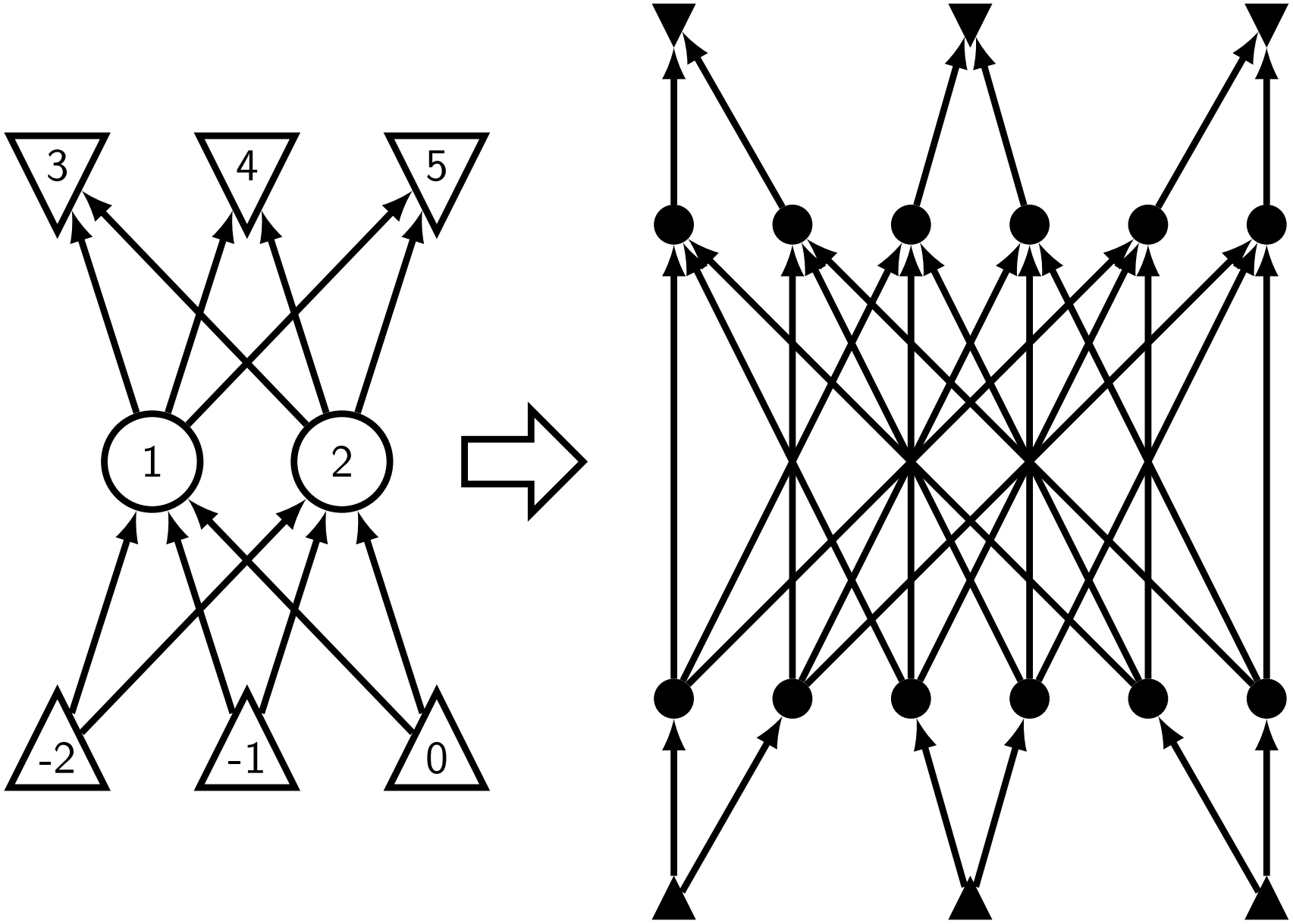


Scarsity $\leftrightarrow$ Optimal Jacobian Accumulation

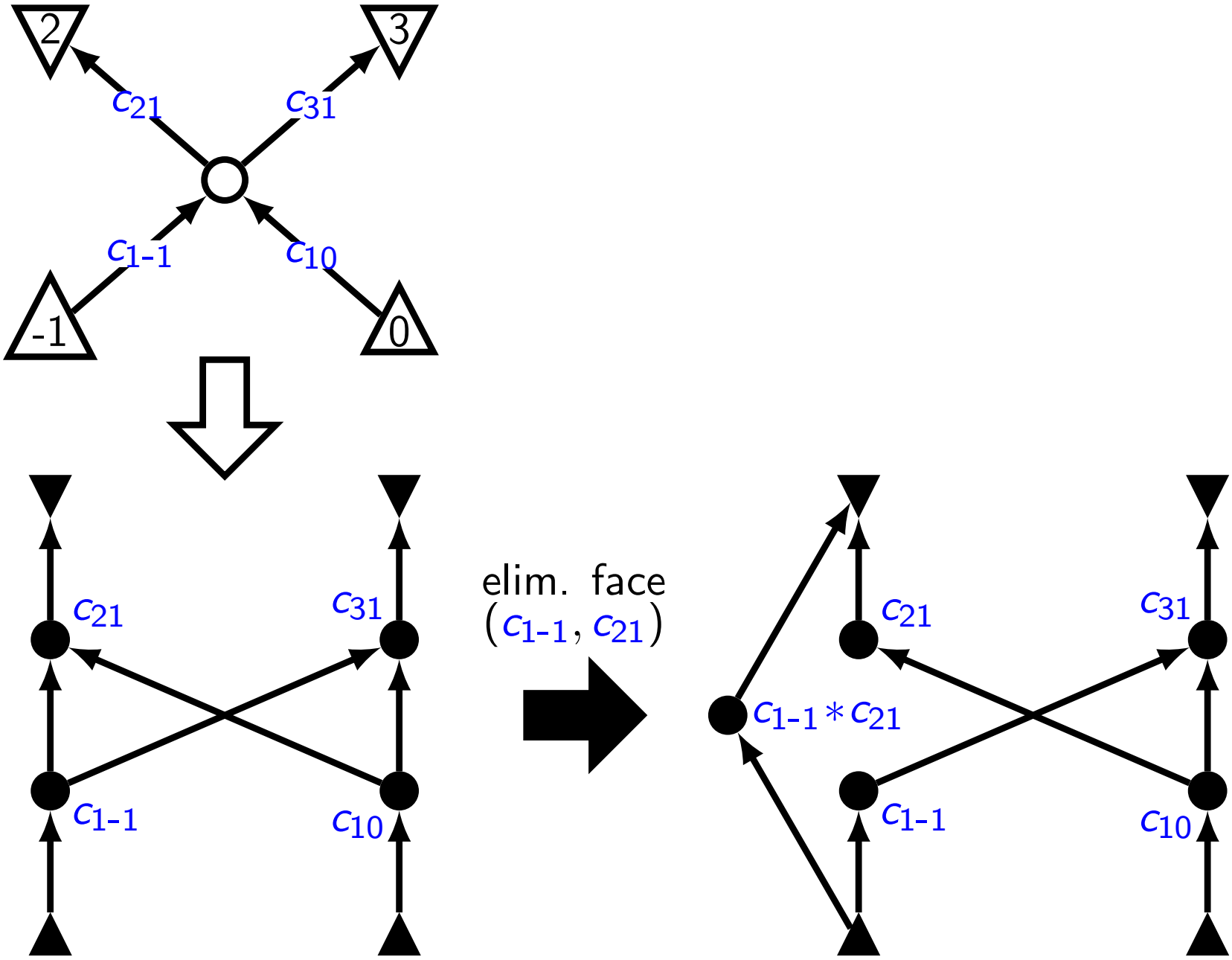


– Combined cost metric: preaccumulation followed by forward vertex elim.
– Results for OJA apply (modulo the fact that p tends to infinity)
– Implication: divisions are useful for OJA (or not? face elim.?)

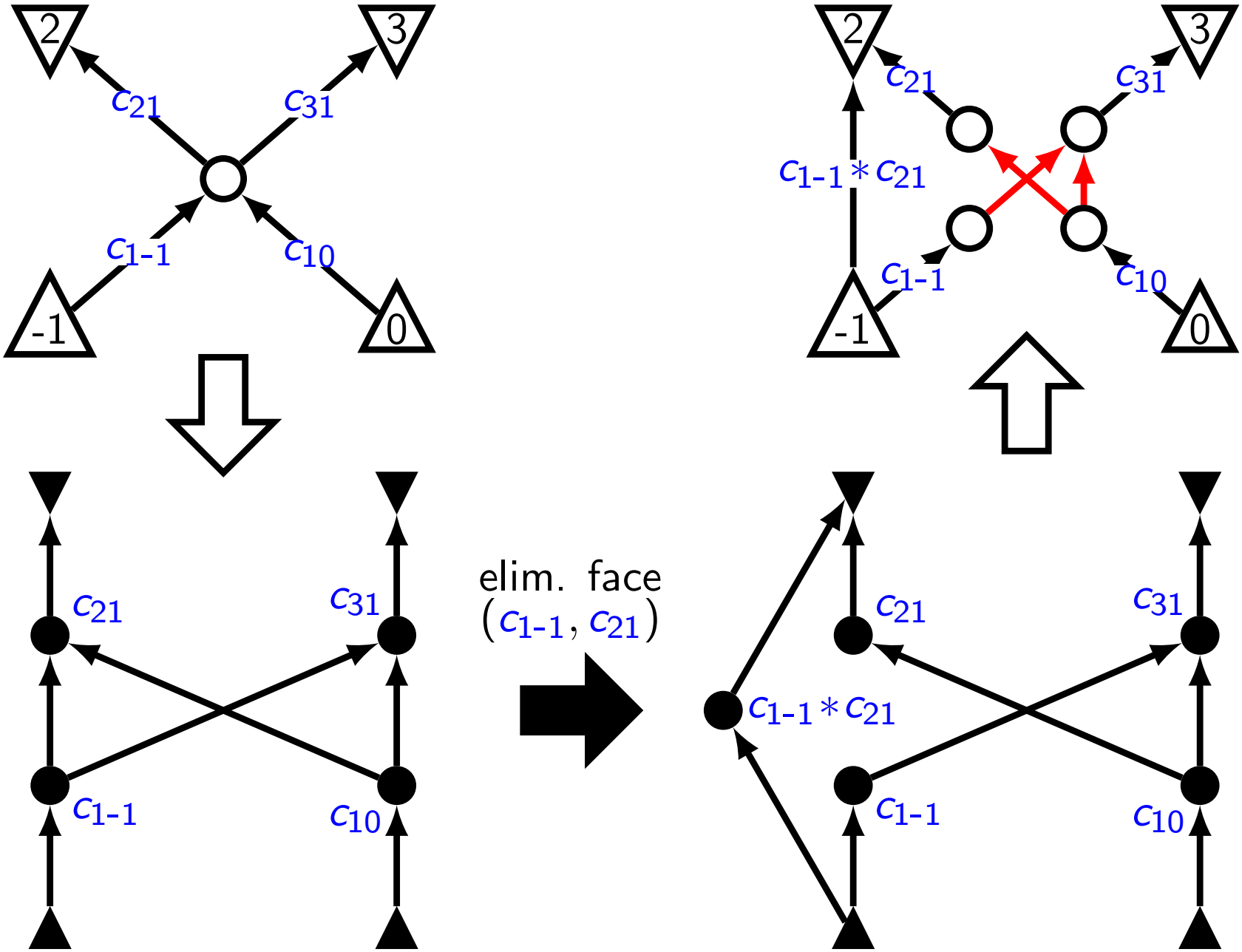
Exploiting Scarsity with Face Elimination



Exploiting Scarsity with Face Elimination



Exploiting Scarsity with Face Elimination



Vote Time!

"Scarsity" or "Scarcity" ?

Thanks!

Questions?