

# Language and Statistics II

---

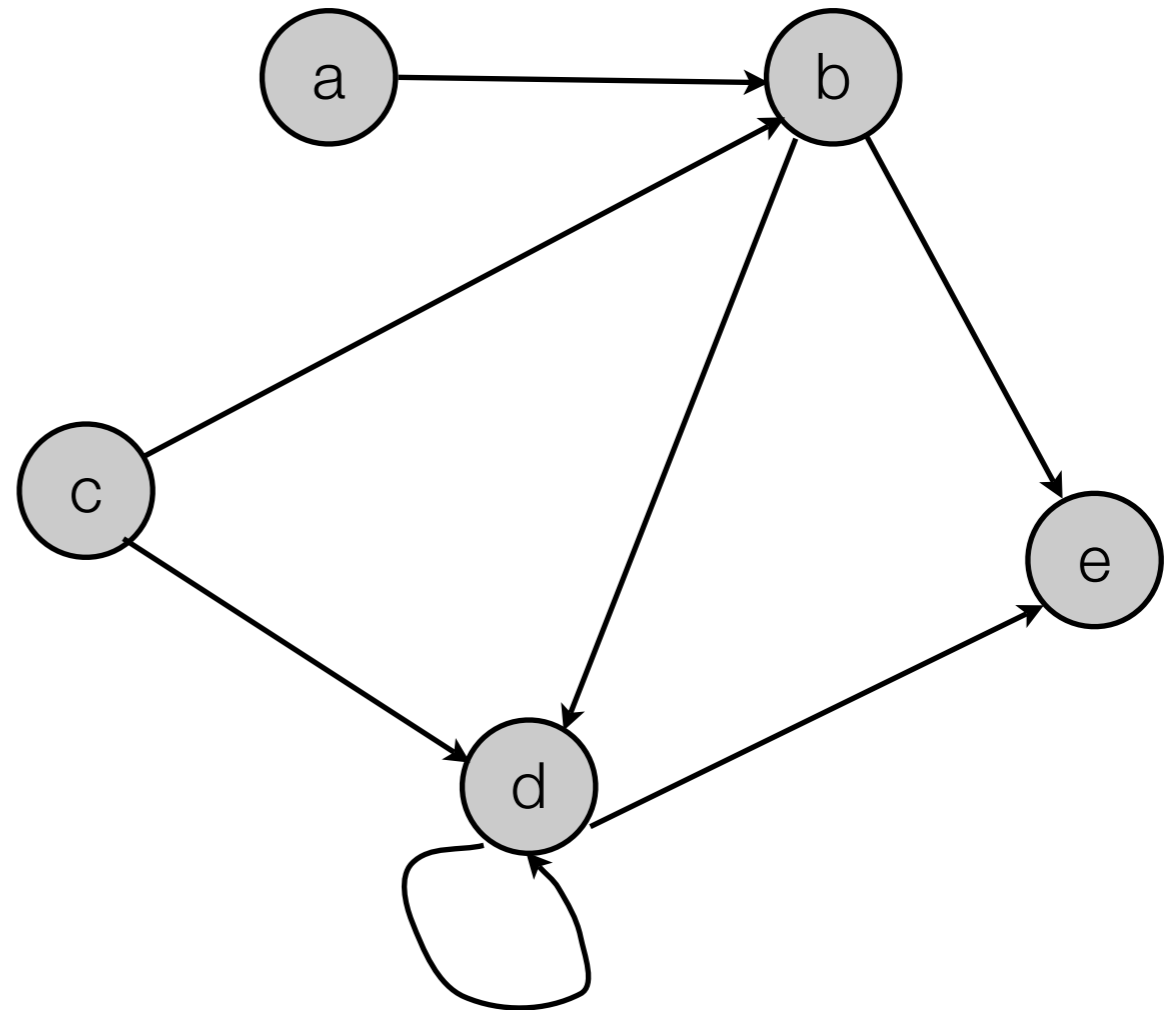
Lecture 4: Dynamic programming algorithms as grounded weighted logic programs

# Graph Reachability, in Prolog

---

`reachable(X) :- initial(X).`

`reachable(Y) :- reachable(X), edge(X, Y).`



# Graph Reachability, in Prolog

---

reachable(X) :- initial(X).

reachable(Y) :- reachable(X), edge(X, Y).

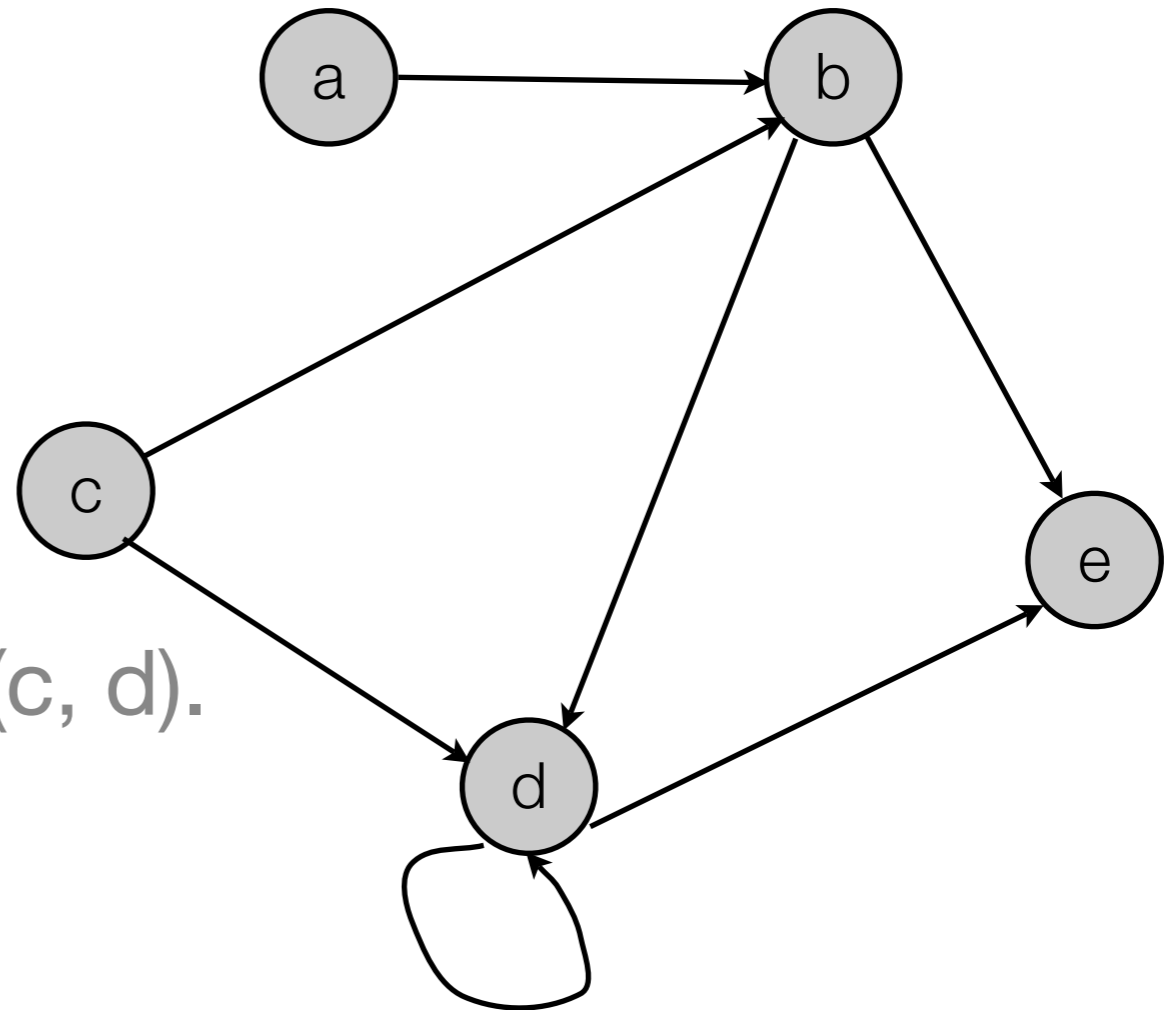
*Problem encoding:*

initial(a).

edge(a, b). edge(b, d).

edge(b, e). edge(c, b). edge(c, d).

edge(d, d). edge(d, e).



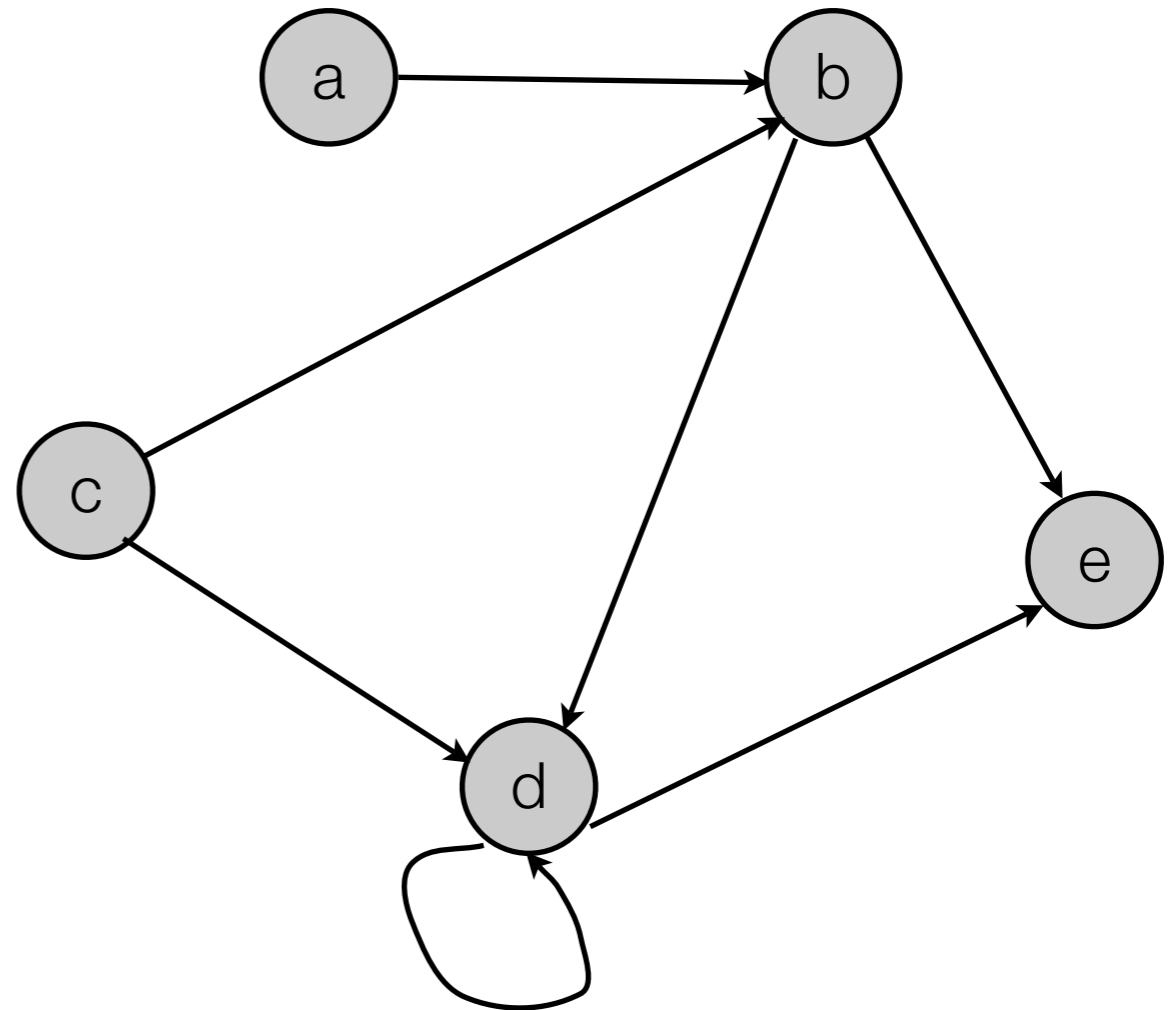
# Graph Reachability, in Prolog

---

`reachable(X) :- initial(X).`

`reachable(Y) :- reachable(X), edge(X, Y).`

**goal** :- `reachable(e).`



# Graph Reachability, in Prolog

---

reachable(X) :- initial(X).

reachable(Y) :- reachable(X), edge(X, Y).

**goal** :- reachable(e).

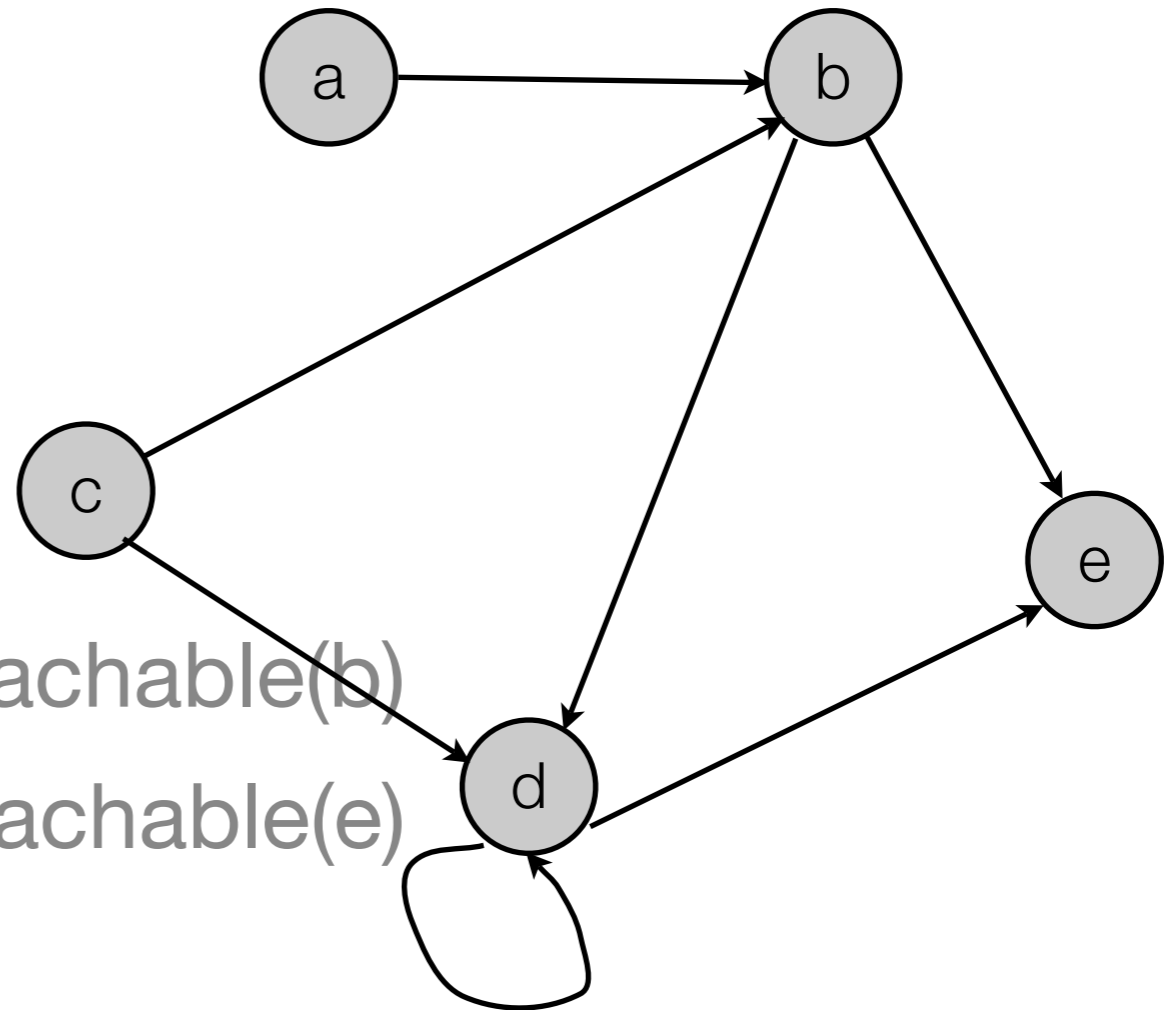
*Proof:*

initial(a) → reachable(a)

reachable(a), edge(a, b) → reachable(b)

reachable(b), edge(b, e) → reachable(e)

reachable(e) → **goal**



# Another View

---

initial(a)

reachable(a)    edge(a, b)

reachable(b)                      edge(b, e)

reachable(e)

**goal**

# Another Proof

---

initial(a)

reachable(a)    edge(a, b)

reachable(b)                      edge(b, d)

reachable(d)                      edge(d, e)

reachable(e)

**goal**

# Another View (Database)

---

| <u>initial</u> |
|----------------|
| a              |

| <u>edge</u>   |             |
|---------------|-------------|
| <i>"from"</i> | <i>"to"</i> |
| a             | b           |
| b             | d           |
| b             | e           |
| c             | b           |
| c             | d           |
| d             | d           |
| d             | e           |

| reachable |
|-----------|
|           |
|           |
|           |
|           |
|           |
|           |

# A Few Comments

---

- There are multiple paths to the goal; each corresponds to a proof.
- We have said nothing about an **algorithm** for finding proofs.
  - Compare last lecture: *first* we defined convex programs, *then* we talked about algorithms for solving them.
  - Intuitively, efficiency means getting a proof fast, minimizing extra work.
- This kind of proof is sometimes called **bottom-up deduction** - start with grounded axioms and work up to the theorem you want to prove (goal).

# Generalizations

---

- Boolean values  $\rightarrow$  arbitrary semirings
- More tables, more rules
- General techniques for solving grounded weighted logic programs

# Semiring $(K, \oplus, \otimes, \mathbf{0}, \mathbf{1})$

---

- $K$ : a set of values
- $\oplus$ : an associative, commutative operator with an identity value  $\mathbf{0}$ 
  - $(x \oplus y) \oplus z = x \oplus (y \oplus z)$
  - $x \oplus y = y \oplus x$
  - $x \oplus \mathbf{0} = x$
- $\otimes$ : an associative operator that distributes over  $\oplus$ , with an identity value  $\mathbf{1}$ 
  - $(x \otimes y) \otimes z = x \otimes (y \otimes z)$
  - $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$
  - $x \otimes \mathbf{1} = x$

# Boolean Semiring $(\{F, T\}, \vee, \wedge, F, T)$

---

- $\{T, F\}$ : a set of values
- $\vee$ : an associative, commutative operator with an identity value  $F$ 
$$(x \vee y) \vee z = x \vee (y \vee z)$$
$$x \vee y = y \vee x$$
$$x \vee F = x$$
- $\wedge$ : an associative operator that distributes over  $\vee$ , with an identity value  $T$ 
$$(x \wedge y) \wedge z = x \wedge (y \wedge z)$$
$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$
$$x \wedge T = x$$

# “Cost” Semiring $(\mathbb{R}_+, \min, +, \infty, 0)$

---

- $\mathbb{R}_+$ : a set of values
- $\min$ : an associative, commutative operator with an identity value  $\infty$ 
  - $\min(\min(x, y), z) = \min(x, \min(y, z))$
  - $\min(x, y) = \min(y, x)$
  - $\min(x, \infty) = x$
- $+$ : an associative operator that distributes over  $\min$ , with an identity value  $0$ 
  - $(x + y) + z = x + (y + z)$
  - $x + \min(y, z) = \min(x + y, x + z)$
  - $x + 0 = x$

# Graph Reachability: Cost Version

---

$\text{reachable}(X) \text{ min} = \underline{\text{initial}}(X).$

$\text{reachable}(Y) \text{ min} = \text{reachable}(X) + \underline{\text{edge}}(X, Y).$

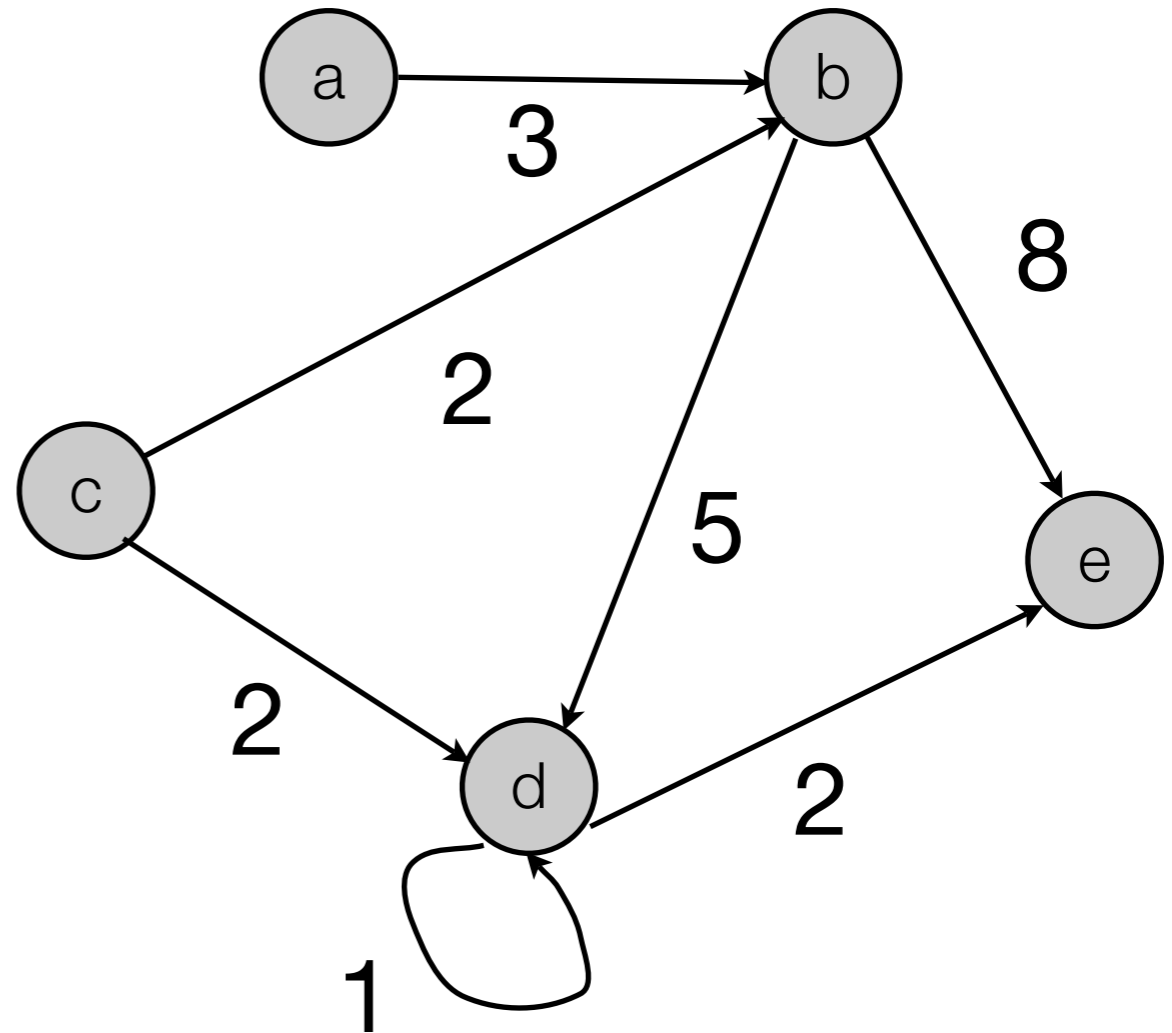
*Problem encoding:*

$\text{initial}(a) := 0.$

$\text{edge}(a, b) := 3.$

$\text{edge}(b, d) := 5.$

... etc.





# Another Cost Program

---

$$a(l, J) \text{ min} = a(l - 1, J - 1) + \underline{x}(X, l) + \underline{y}(X, J).$$

$$a(l, J) \text{ min} = a(l - 1, J - 1) + \underline{s} + \underline{x}(X, l) + \underline{y}(Y, J).$$

$$a(l, J) \text{ min} = a(l, J - 1) + \underline{i} + \underline{y}(Y, J).$$

$$a(l, J) \text{ min} = a(l - 1, J) + \underline{d} + \underline{x}(X, l).$$

$$\text{goal min} = a(M, N) + \underline{x}_l(M) + \underline{y}_l(N).$$

$$a(0, 0) := 0.$$

# More Readable

---

$\text{align}(l, j) \text{ min} = \text{align}(l - 1, j - 1) + \underline{x}(X, l) + \underline{y}(X, j).$

$\text{align}(l, j) \text{ min} = \text{align}(l - 1, j - 1) + \underline{\text{subcost}} + \underline{x}(X, l) + \underline{y}(Y, j).$

$\text{align}(l, j) \text{ min} = \text{align}(l, j - 1) + \underline{\text{inscost}} + \underline{y}(Y, j).$

$\text{align}(l, j) \text{ min} = \text{align}(l - 1, j) + \underline{\text{delcost}} + \underline{x}(X, l).$

$\text{goal min} = \text{align}(M, N) + \underline{\text{xlen}}(M) + \underline{\text{ylen}}(N).$

$\text{align}(0, 0) := 0.$

# Another Notation

---

$$\begin{aligned} \text{align}(I, J) = & \min(\min_X [ \text{align}(I-1, J-1) + \underline{x}(X, I) + \underline{y}(X, J) ], \\ & \min_{X, Y} [ \text{align}(I-1, J-1) + \underline{\text{subcost}} + \underline{x}(X, I) + \underline{y}(Y, J) ], \\ & \min_Y [ \text{align}(I, J-1) + \underline{\text{inscost}} + \underline{y}(Y, J) ], \\ & \min_X [ \text{align}(I-1, J) + \underline{\text{delcost}} + \underline{x}(X, I) ] ) \end{aligned}$$

$$\text{goal} = \min_{M, N} \text{align}(M, N) + \underline{\text{xlen}}(M) + \underline{\text{ylen}}(N).$$

$$\text{align}(0, 0) = 0.$$

# Simplified Again

---

$$a_{i,j} = \min(a_{i-1,j-1} \quad \text{if } x_i = y_j, \\ a_{i-1,j-1} + \underline{\text{subcost}}, \\ a_{i,j-1} + \underline{\text{inscost}}, \\ a_{i-1,j} + \underline{\text{delcost}})$$

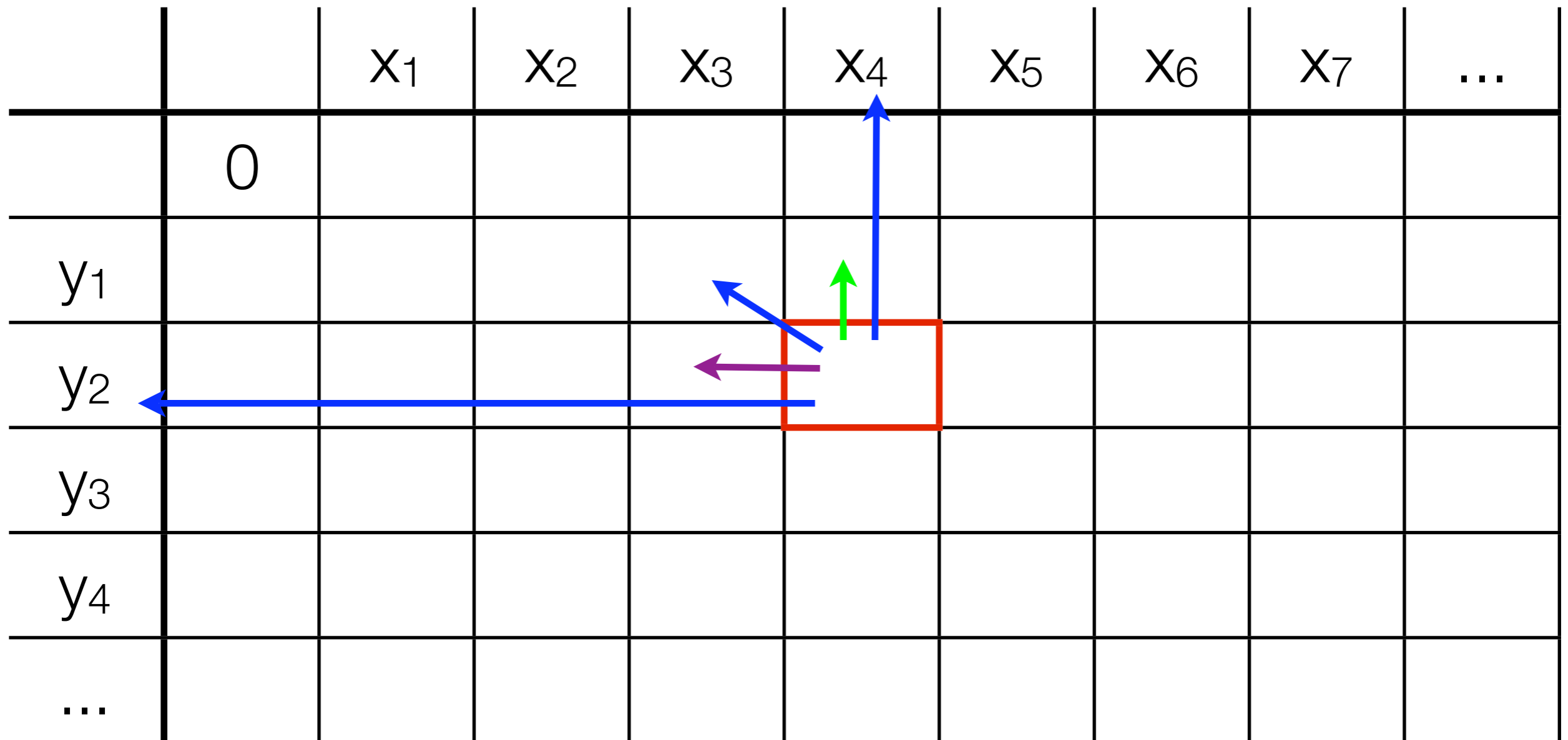
$$\text{goal} = a_{\text{length}(x), \text{length}(y)}$$

$$a_{0,0} = 0.$$



# Classic View

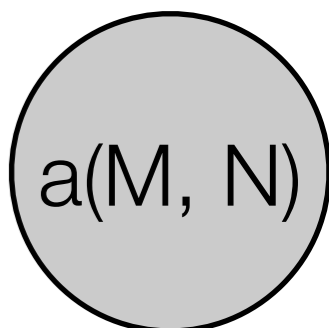
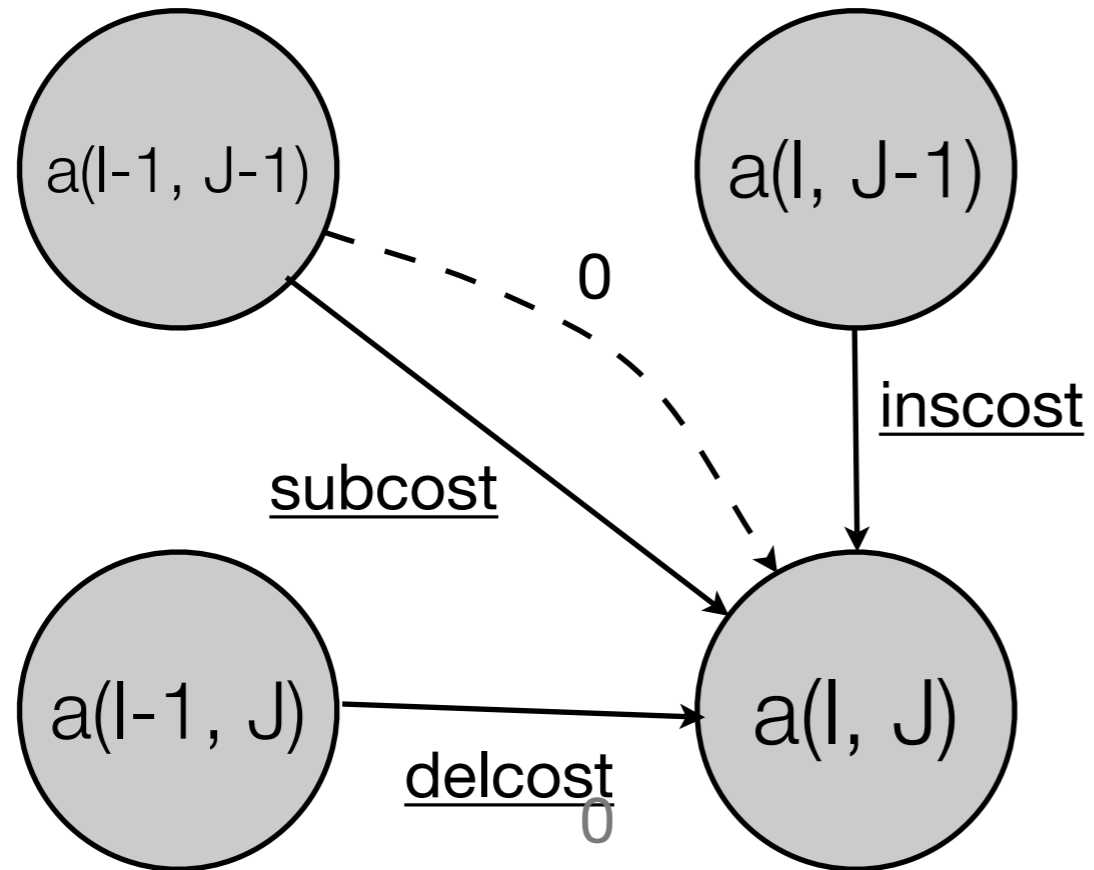
---





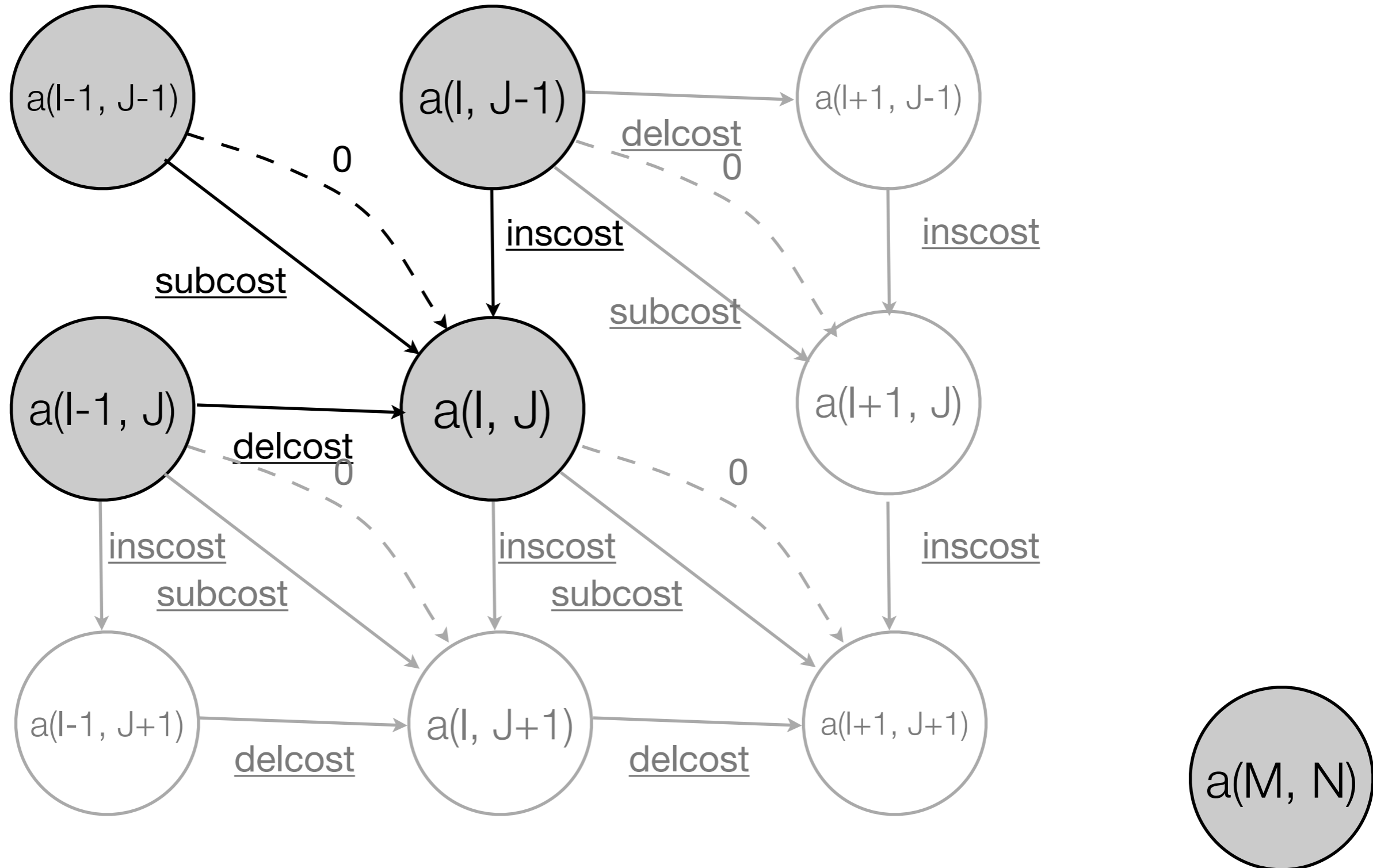
# Graph View

---



# Graph View

---



# A Few Comments

---

- Building up the database or the graph may not be the most efficient way to solve the problem ... but you should see that these are equivalent to the original problem.
- Recall that when calculating Levenshtein distance, we sometimes want to know the actual **alignment** ... how do we do this?
- Each **alignment** corresponds to a single **proof**
- Can capture backpointers in a semiring!

# “Cost”/Derivation Semiring

---

$\mathbf{K} = \{ (x, p) : x \in \mathbb{R}_+, \text{ and } p \text{ is a derivation string} \}$

$(x_1, p_1) \oplus (x_2, p_2) = (\min(x_1, x_2), x_1 < x_2 ? p_1 : p_2)$

$(x_1, p_1) \otimes (x_2, p_2) = (x_1 + x_2, p_1 \cdot p_2)$  (not commutative!)

$\mathbf{0} = (\infty, \varepsilon)$

$\mathbf{1} = (0, \varepsilon)$

- Note: Goodman (1999) generalizes this slightly to keep track of all “ties” by storing a **set** of proofs, which he calls a “derivation forest.”
- Question: what are the x, y, inscost, delcost, subcost, axiom values?

# Derivations as Strings

---

$\underline{x}(b, 1) := (0, \text{"b"}).$

b b sub a o n n a a n n ins z a a

$\underline{x}(a, 2) := (0, \text{"a"}).$

...

b b del a ins o ...

$\underline{y}(b, 1) := (0, \text{"b"}).$

b b ins o del a ...

$\underline{y}(o, 2) := (0, \text{"o"}).$

del b ins b ...

...

$\underline{\text{inscost}} := (1, \text{"ins"}).$

$\underline{\text{delcost}} := (1, \text{"del"}).$

$\underline{\text{subcost}} := (1, \text{"sub"}).$

# Generalizing Edit Distance

---

$\text{align}(I, J) \min = \text{align}(I - 1, J - 1) + \underline{x}(X, I) + \underline{y}(X, J).$

$\text{align}(I, J) \min = \text{align}(I - 1, J - 1) + \underline{\text{subcost}}(X, Y) + \underline{x}(X, I) + \underline{y}(Y, J).$

$\text{align}(I, J) \min = \text{align}(I, J - 1) + \underline{\text{inscost}}(Y) + \underline{y}(Y, J).$

$\text{align}(I, J) \min = \text{align}(I - 1, J) + \underline{\text{delcost}}(X) + \underline{x}(X, I).$

$\text{goal} \min = \text{align}(M, N) + \underline{\text{xlen}}(M) + \underline{\text{ylen}}(N).$

$\text{align}(0, 0) := 0.$

# Generalizing Edit Distance

---

$\text{align}(I, J) \min = \text{align}(I - 1, J - 1) + \underline{x}(X, I) + \underline{y}(X, J).$

$\text{align}(I, J) \min = \text{align}(I - 1, J - 1) + \underline{\text{subcost}} + \underline{x}(X, I) + \underline{y}(Y, J).$

$\text{align}(I, J) \min = \text{align}(I, J - 1) + \underline{\text{inscost}} + \underline{y}(Y, J).$

$\text{align}(I, J) \min = \text{align}(I - 1, J) + \underline{\text{delcost}} + \underline{x}(X, I).$

$\text{align}(I, J) \min = \text{align}(I - 2, J - 2) + \underline{\text{transcost}} + \underline{x}(A, I - 1) + \underline{x}(B, I) + \underline{y}(B, J - 1) + \underline{y}(A, J).$

$\text{goal} \min = \text{align}(M, N) + \underline{\text{xlen}}(M) + \underline{\text{ylen}}(N).$

$\text{align}(0, 0) := 0.$

# Another Algorithm

---

`constit(X, I, I) :- word(W, I), unary(X, W).`

`constit(X, I, K) :- constit(Y, I, J), constit(Z, J+1, K), binary(X, Y, Z).`

`goal :- constit("S", 1, N), length(N).`

How is this different from before?

# CKY

---

`constit(X, I, I) :- word(W, I), unary(X, W).`

`constit(X, I, K) :- constit(Y, I, J), constit(Z, J+1, K), binary(X, Y, Z).`

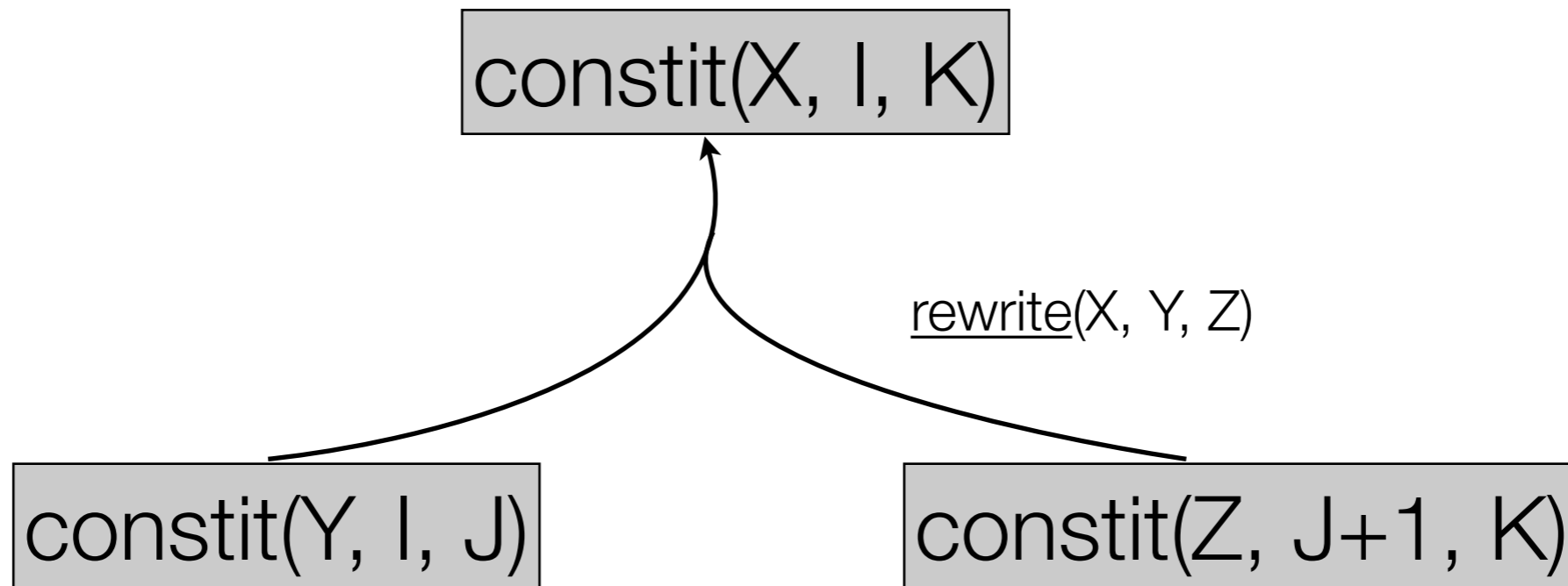
`goal :- constit("S", 1, N), length(N).`

How is this different from before?

The graph for constit items isn't a graph; a single step of the proof uses **two** constits as antecedents.

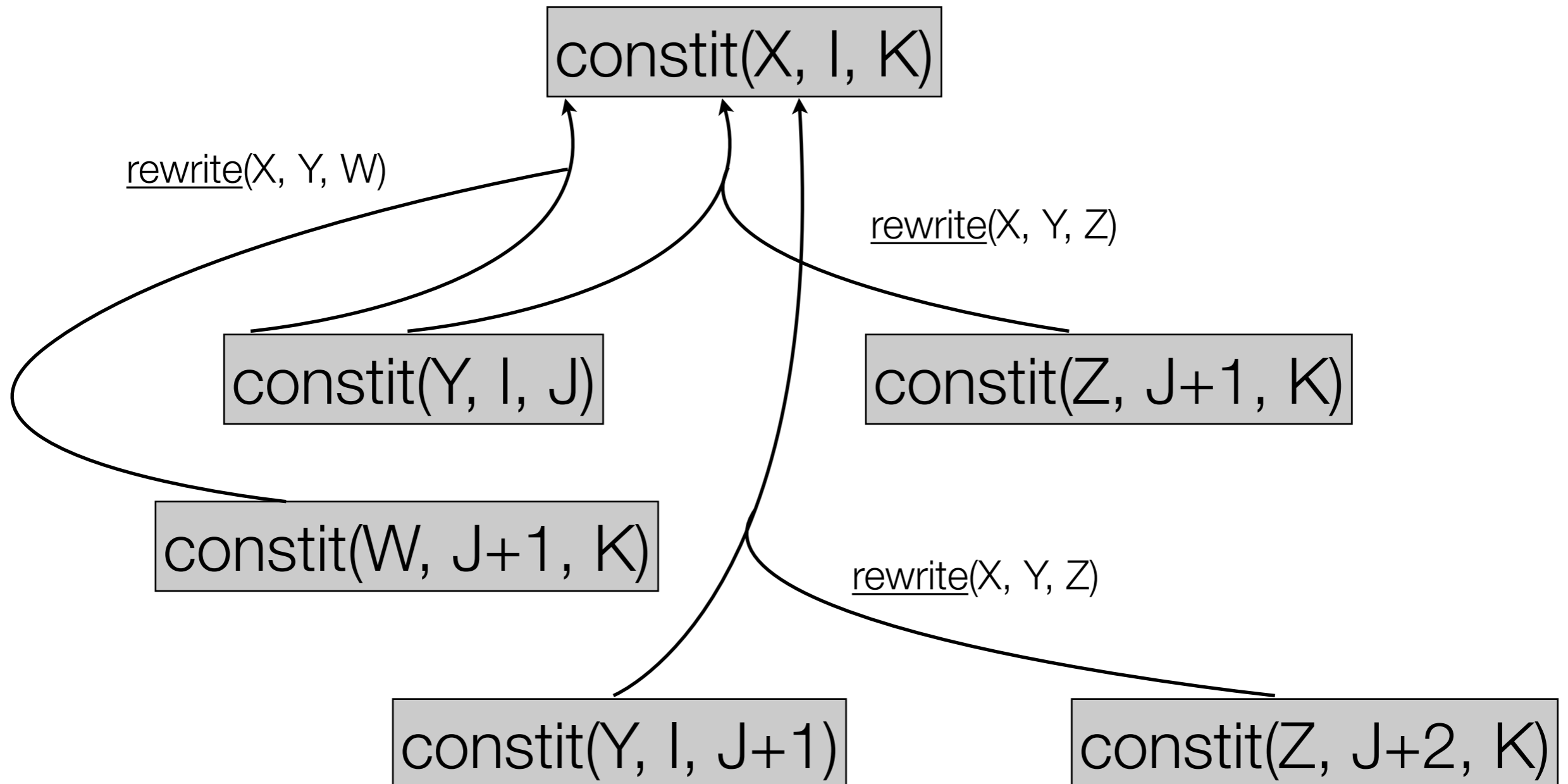
# Hypergraph View

---



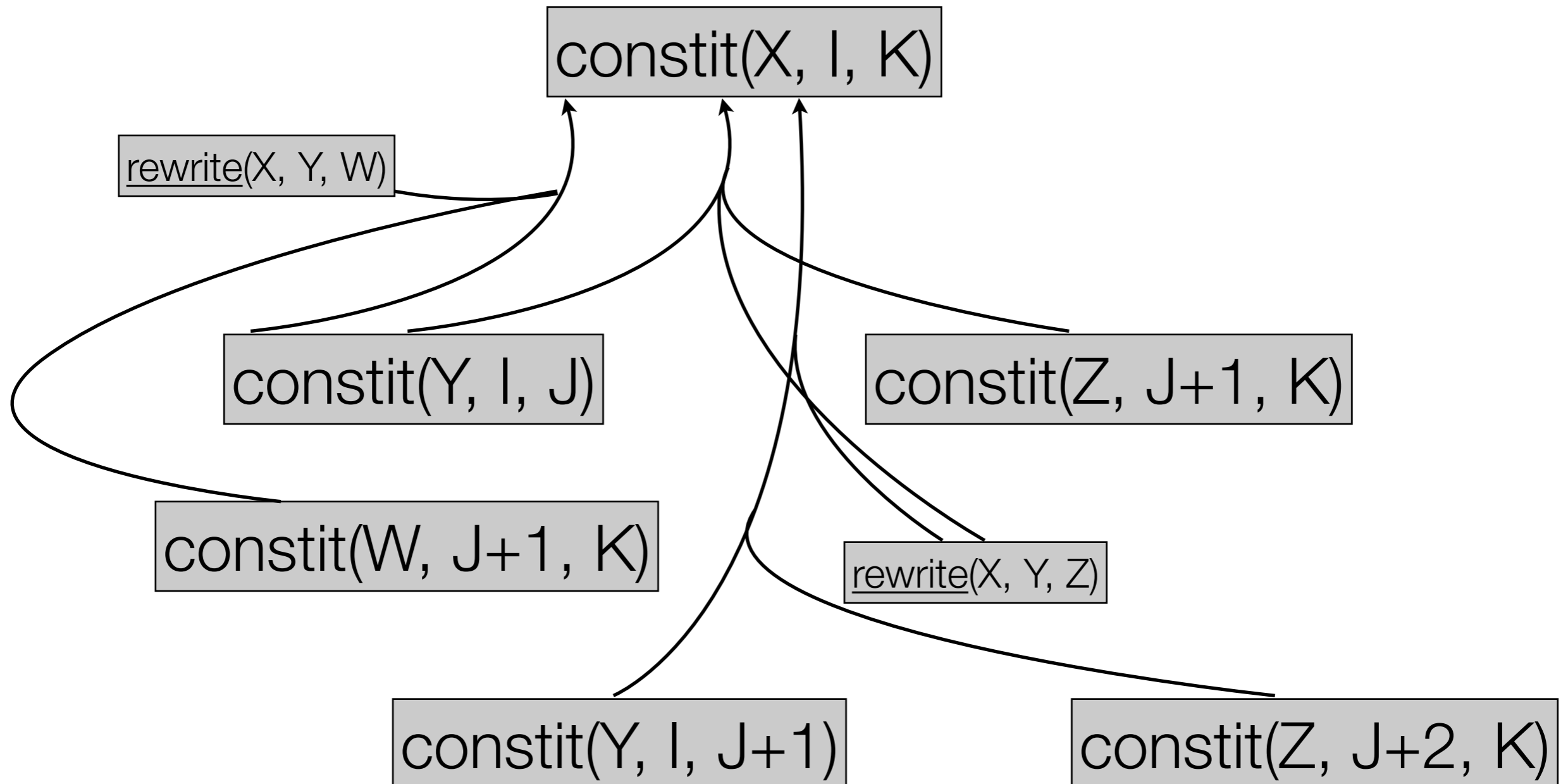
# Hypergraph View

---



# Hypergraph View

---



# Classic (Chart) View

---

*“to”*

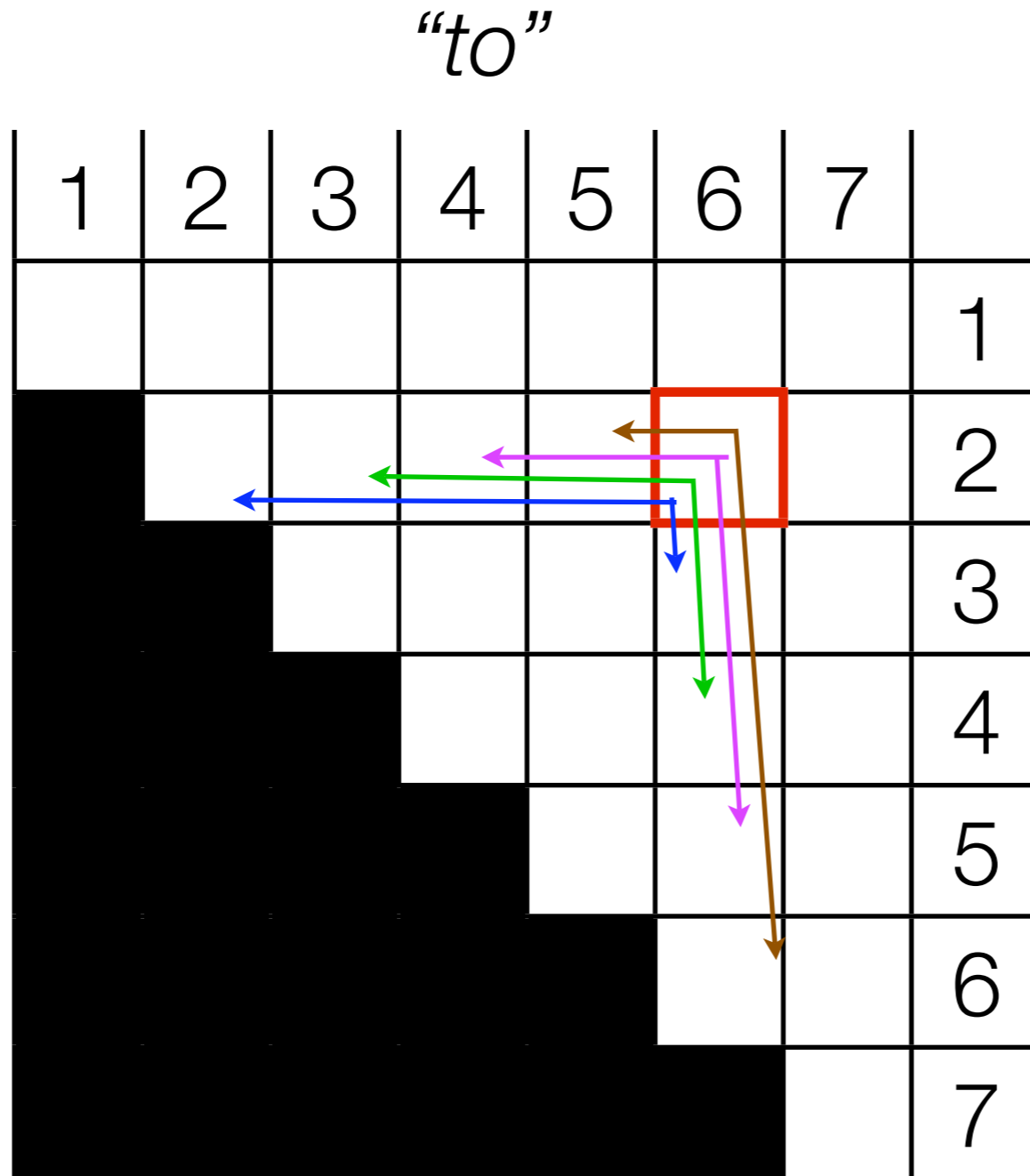
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |   |
|---|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   | 1 |
|   |   |   |   |   |   |   | 2 |
|   |   |   |   |   |   |   | 3 |
|   |   |   |   |   |   |   | 4 |
|   |   |   |   |   |   |   | 5 |
|   |   |   |   |   |   |   | 6 |
|   |   |   |   |   |   |   | 7 |

*“from”*

each cell holds a map from nonterminals to values, plus backpointers

# Classic (Chart) View

---



*“from”*

each cell holds a map from nonterminals to values, plus backpointers

# Probabilistic CKY

---

$\text{constit}(X, I, I) \text{ max} = \underline{\text{word}}(W, I) \times \underline{\text{unary}}(X, W).$

$\text{constit}(X, I, K) \text{ max} = \text{constit}(Y, I, J) \times \text{constit}(Z, J+1, K) \times \underline{\text{binary}}(X, Y, Z).$

$\text{goal} \text{ max} = \text{constit}(\text{"S"}, 1, N) \times \underline{\text{length}}(N).$

Set values to be probabilities.

“Max-times” semiring.

# Probabilistic CKY With Log Trick

---

$\text{constit}(X, I, I) \text{ max} = \underline{\text{word}}(W, I) + \underline{\text{unary}}(X, W).$

$\text{constit}(X, I, K) \text{ max} = \text{constit}(Y, I, J) + \text{constit}(Z, J+1, K) + \underline{\text{binary}}(X, Y, Z).$

$\text{goal} \text{ max} = \text{constit}(\text{"S"}, 1, N) + \underline{\text{length}}(N).$

Set values to be log-probabilities.

“Max-plus” semiring.

# A Few Comments

---

- Can include the “best tree” in the semiring, too.
  - Or use backpointers.
- All of the following are different representations of the same thing:
  - proof
  - hyperpath to goal
  - CNF-CFG derivation
  - dense graphical models with recursive structure