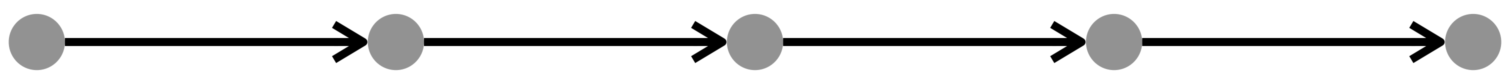
Newtonian Data Analytics

Remy Wang, Sep 28 @ Simons

W/ Mahmoud Abo Khamis, Hung Q. Ngo, Reinhard Pichler, Dan Suciu

```
R(x, y, z) :- E(x, y), T(y, z).
S(x,    z) :- E(x, y), T(y, z).
```



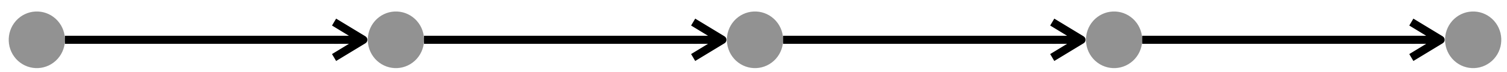
T = E =

1	2
2	3
3	4
4	5

S =

1	3
2	4
3	5

```
T(x, z) :- E(x, z).
T(x, z) :- E(x, y), T(y, z).
```



T_0 = {}, E =

1	2
2	3
3	4
4	5

T_1 =

1	2
2	3
3	4
4	5

T_2 =

1	2
2	3
3	4
4	5

1	3
2	4
3	5

```
T(x, z) :- E(x, z).
T(x, z) :- E(x, y), T(y, z).
```

Active domain: {1, 2, 3, 4, 5}

T_0 = {}, E =

1	2
2	3
3	4
4	5

T_1 =

E =

1	2
2	3
3	4
4	5

=

0 1 1
1 1 2
0 1 3
0 1 4
0 1 5 Length = 25
0 2 1
0 2 2
1 2 3
... ...

	1		1
	0		1
	0		0
	0		1
E =	0	T =	0
	0		0
	0		0
	1		1
	...		...

```
T(x, z) :- E(x, z).
T(x, z) :- E(x, y), T(y, z).
```

$$T_0 \xrightarrow{f} T_1 \xrightarrow{f} T_2 \cdots$$

$$f : \mathbb{B}^{25} \rightarrow \mathbb{B}^{25}$$

Datalog eval. = finding (least) **fixpoint** of *f*

$$f(f^*(T_0)) = f^*(T_0)$$

$$T_0 \xrightarrow{f} T_1 \xrightarrow{f} T_2 \cdots$$

Semirings

$$\begin{cases} f : \mathbb{B}^{25} \to \mathbb{B}^{25} \\ \mathbb{R}^n \to \mathbb{R}^n \\ (\mathbb{N} \cup \{\infty\})^n \to (\mathbb{N} \cup \{\infty\})^n \end{cases}$$

Definition: An algebra is called a closed semi-ring iff the following equalities are identically true:

a) $a+(b+c) = (a+b)+c$ addition is associative

b) $a+b = b+a$ addition is commutative

c) $a+0 = a$ 0 is a unit for addition

d) $a.(b.c) = (a.b).c$ multiplication is associative

e) $a.1 = 1.a = a$ 1 is a unit for multiplication

f) $a.(b+c) = a.b+a.c$

$(b+c).a = b.a+c.a$ multiplication distributes over addition

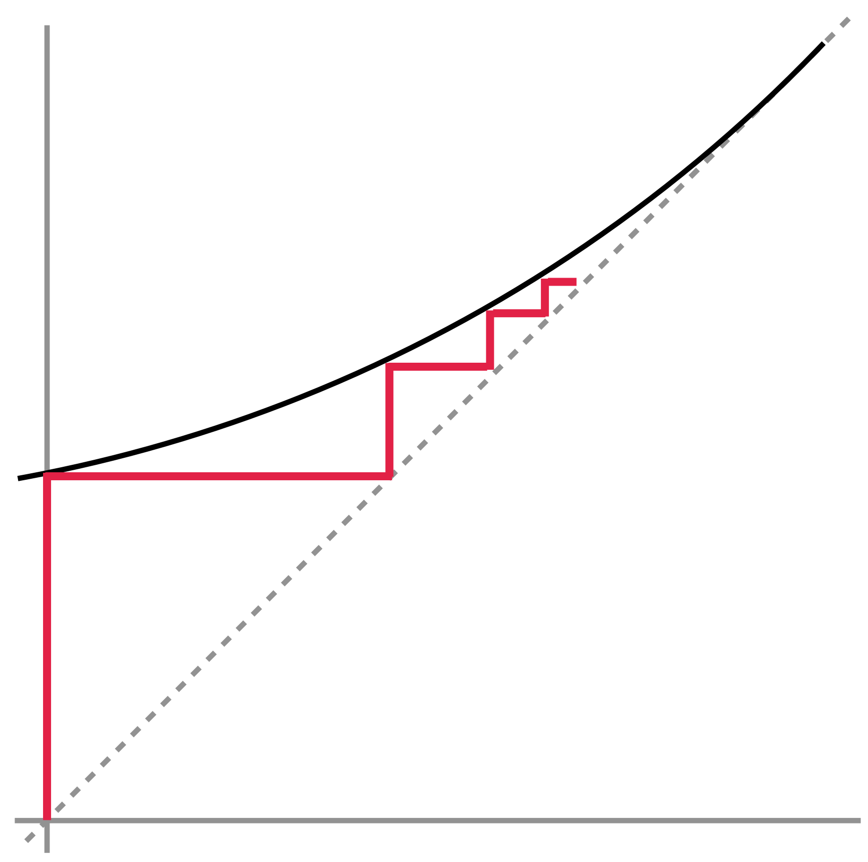
g) $a^* = 1+a.a^* = 1+a^*.a$

Lehman '77

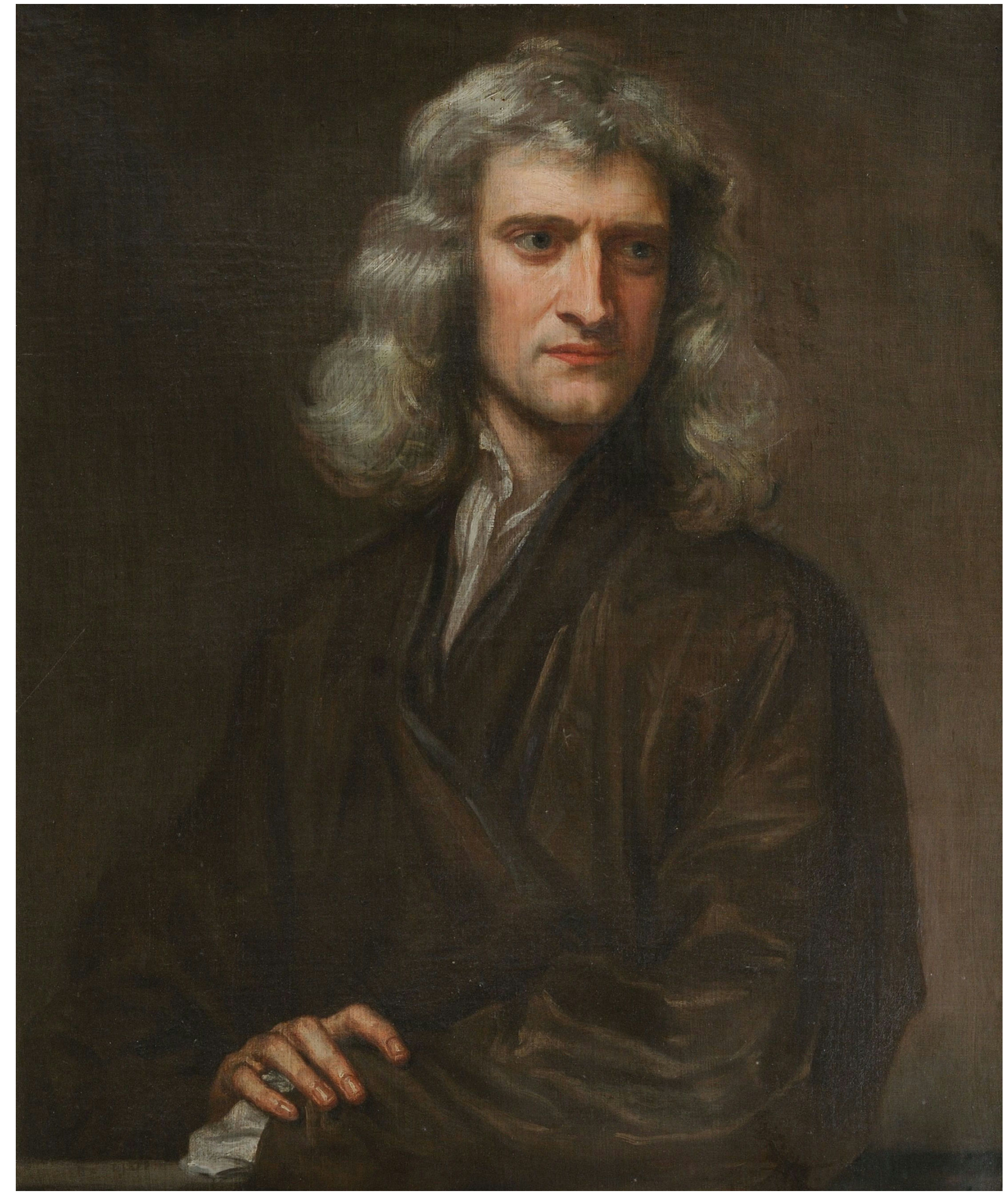
Datalog eval. = finding (least) **fixpoint** of *f*

$$f(f^{*}(T_0)) = f^{*}(T_0)$$

Naive evaluation (a.k.a. Kleene iteration)



Carl Friedrich Gauss



Isaac Newton

ALGEBRAIC STRUCTURES FOR TRANSITIVE CLOSURE

by

DANIEL J. LEHMANN

Warshall's algorithm for computing the transitive closure of a Boolean matrix, Floyd's algorithm for minimum-cost paths, Kleene's proof that every regular language can be defined by a regular expression and Gauss-Jordan's method for inverting real matrices are different interpretations of the same program scheme (with one counter and an array).[1]

```
T(x, z) :- E(x, z).
T(x, z) :- E(x, y), T(y, z).
```

$$
\begin{aligned}
T &= E + EE + EEE + \cdots \\
&= (I + E + EE + \cdots)E \\
&= E^* E
\end{aligned}
$$

```
T(x, z) :- E(x, z).
T(x, z) :- E(x, y), T(y, z).
```

```
a* = 1+a.a*
```

$$
\begin{aligned}
T &= E + EE + EEE + \cdots \\
&= (I + E + EE + \cdots)E \\
&= E^* E
\end{aligned}
$$

$$
\begin{aligned}
& I + EE^* \\
&= I + E(I + E + EE + \cdots) \\
&= (I + E + EE + \cdots) \\
&= E^*
\end{aligned}
$$

Floyd-Warshall-Kleene

```
for k in 1...n:
    A' ← new
    for i,j in 1...n:
        A'_ij ← A_ij + A_ik · (A_kk)* · A_kj
    A ← A'
```

$$\text{for } k \text{ in } 1 \ldots n:$$
$$\quad A' \leftarrow \texttt{new}$$
$$\quad \text{for } i, j \text{ in } 1 \ldots n:$$
Regex for $i \to j$ $\quad A'_{ij} \leftarrow A_{ij} + A_{ik} \cdot (A_{kk})^* \cdot A_{kj}$
$$\quad A \leftarrow A'$$

$i \to j$ $\quad i \to k$ $\quad k \to k$ $\quad k \to j$

Gaussian Elimination

A Survey of Sequential and Systolic
Algorithms for the Algebraic Path Problem

Eugene Fink
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213, USA
eugene@cs.cmu.edu

under the supervision of
Jo C. Ebergen

Research Report

Fast Algorithms for Solving Path Problems

ROBERT ENDRE TARJAN

Stanford University, Stanford, California

ABSTRACT Let $G = (V, E)$ be a directed graph with a distinguished source vertex s. The *single-source path expression problem* is to find, for each vertex v, a regular expression $P(s, v)$ which represents the set of all paths in G from s to v A solution to this problem can be used to solve shortest path problems, solve sparse systems of linear equations, and carry out global flow analysis. A method is described for computing path expressions by dividing G into components, computing path expressions on the components by Gaussian elimination, and combining the solutions This method requires $O(m\alpha(m, n))$ time on a reducible flow graph, where n is the number of vertices in G, m is the number of edges in G, and α is a functional inverse of Ackermann's function The method makes use of an algorithm for evaluating functions defined on paths in trees. A simplified version of the algorithm, which runs in $O(m \log n)$ time on reducible flow graphs, is quite easy to implement and efficient in practice

KEY WORDS AND PHRASES: Ackermann's function, code optimization, compiling, dominators, Gaussian elimination, global flow analysis, graph algorithm, linear algebra, path compression, path expression, path problem, path sequence, reducible flow graph, regular expression, shortest path, sparse matrix

$$(I - B)^* = B^{-1}$$

Proof: $(I - B)^* \cdot B$

$$= A^* \cdot (1 - A)$$

$$= A^* - A^* A$$

$$= 1$$

$$A^* = 1 + A^* A$$

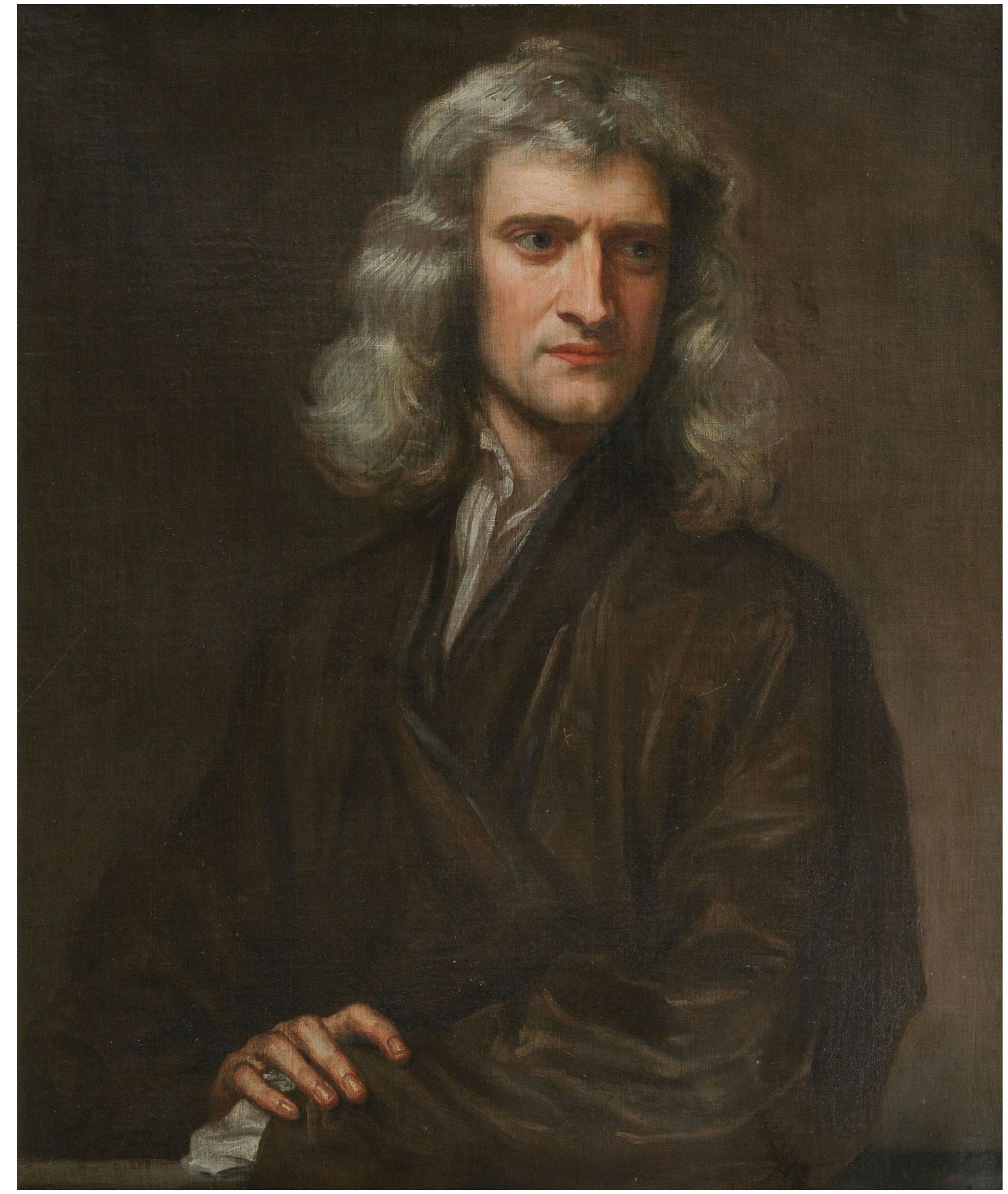
Carl Friedrich Gauss

Can compute the *closure*

$$A^* = 1 + A^* A$$

In cubic time
(sometimes "linear")

Can compute **linear** Datalog



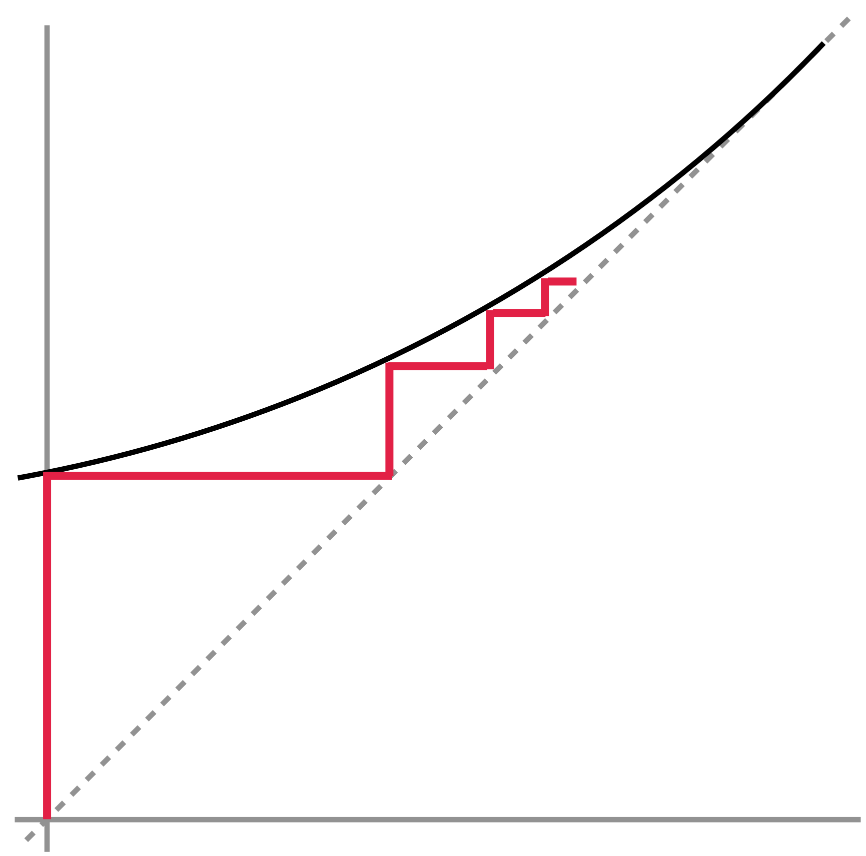
Carl Friedrich Gauss

Isaac Newton

Datalog eval. = finding (least) **fixpoint** of *f*

$$f(f^*(T_0)) = f^*(T_0)$$

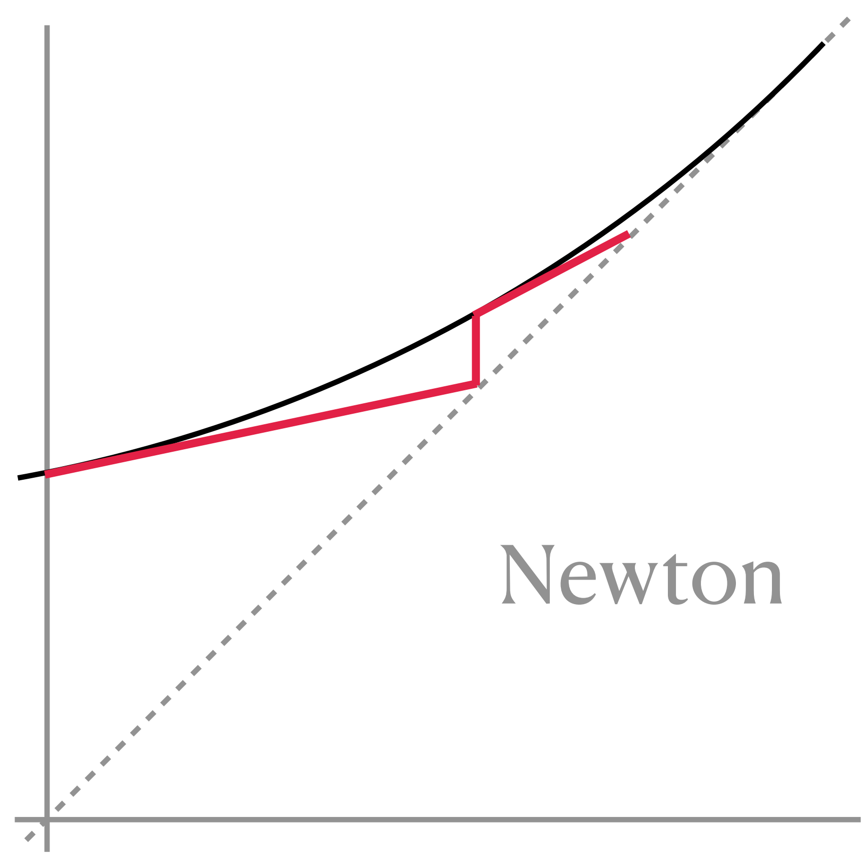
Naive evaluation (a.k.a. Kleene iteration)



Datalog eval. = finding (least) **fixpoint** of *f*

$$f(f^{*}(T_0)) = f^{*}(T_0)$$

Naive evaluation (a.k.a. Kleene iteration)



to Newton...

$$f(x) + f'(x) \cdot z \leq f(x + z)$$

$$f(x) \stackrel{\text{def}}{=} x^n$$

$$\begin{aligned}
& f(x) + f'(x) \cdot z \\
=& x^n + nx^{n-1}z \\
\leq& x^n + nx^{n-1}z + \binom{n}{2}x^{n-2}z^2 + \binom{n}{3}x^{n-3}z^3 + \cdots \\
=& (x+z)^n = f(x+z)
\end{aligned}$$

$$x + \delta = f(x)$$

↑

How to pick?

$$
\begin{aligned}
& x_{n+1} \\
=& x_n + \Delta_n \\
=& x_n + f'(x_n)\Delta_n + \delta_n \\
=& f(x_n) + f'(x_n)\Delta_n \\
\leq& f(x_n + \Delta_n) \\
=& f(x_{n+1})
\end{aligned}
$$

$$\Delta = f'(x)\Delta + \delta$$

$$x + \delta_n = f(x_n)$$

$$f(x) + f'(x) \cdot z \leq f(x + z)$$

$$+\binom{n}{2}x^{n-2}z^2 + \binom{n}{3}x^{n-3}z^3 + \cdots$$

$$T(x,y) = E(x,y) + \sum_z T(x,z) \cdot T(z,y)$$

$$\begin{aligned} \frac{\partial F(x,y)}{\partial T(u,v)} = &\sum_z \Big([(x,z) = (u,v)] \cdot T(z,y) \\ &\qquad + [(z,y) = (u,v)] \cdot T(x,z) \Big) \\ = &[x = u] \cdot T(v,y) + T(x,u) \cdot [y = v] \end{aligned}$$

$$
\begin{aligned}
\delta_n(x,y) &= \left(E(x,y) + \sum_z T_n(x,z) \cdot T_n(z,y) \right) - T_n(x,y) \\
\Delta_n(x,y) &= \delta_n(x,y) + \sum_{u,v} \frac{\partial F_n(x,y)}{\partial T(u,v)} \cdot \Delta_n(u,v) \\
&= \delta_n(x,y) + \sum_v T_n(v,y) \cdot \Delta_n(x,v) + \sum_u T_n(x,u) \cdot \Delta_n(u,y) \\
&= \delta_n(x,y) + \sum_v \Delta_n(x,v) \cdot T_n(v,y) + \sum_u T_n(x,u) \cdot \Delta_n(u,y) \\
T_{n+1}(x,y) &= T_n(x,y) + \Delta_n(x,y)
\end{aligned}
$$

$$
\begin{aligned}
\delta_0(x,y) &= E(x,y) \\
\Delta_0(x,y) &= \delta_0(x,y) + 0 = E(x,y) \\
T_1(x,y) &= 0 + \Delta_0(x,y) = E(x,y)
\end{aligned}
$$

$$\delta_n(x,y) = \left(E(x,y) + \sum_z T_n(x,z) \cdot T_n(z,y) \right) - T_n(x,y)$$

$$\begin{aligned}
\Delta_n(x,y) =& \delta_n(x,y) + \sum_{u,v} \frac{\partial F_n(x,y)}{\partial T(u,v)} \cdot \Delta_n(u,v) \\
=& \delta_n(x,y) + \sum_v T_n(v,y) \cdot \Delta_n(x,v) + \sum_u T_n(x,u) \cdot \Delta_n(u,y) \\
=& \delta_n(x,y) + \sum_v \Delta_n(x,v) \cdot T_n(v,y) + \sum_u T_n(x,u) \cdot \Delta_n(u,y) \\
T_{n+1}(x,y) =& T_n(x,y) + \Delta_n(x,y)
\end{aligned}$$

$$\delta_1(x,y) = \sum_z E(x,z) \cdot E(z,y) - E(x,y)$$

$$\Delta_1(x,y) = \delta_1(x,y) + \sum_v \Delta_1(x,v) \cdot E(v,y) + \sum_u E(x,u) \cdot \Delta_1(u,y)$$

$$T_2(x,y) = \underbrace{T_1(x,y)}_{= E} + \underbrace{\Delta_1(x,y)}_{\text{Paths} \geq 2} = E(x,y) + \Delta_1(x,y)$$