Dynamic programming using histomorphisms

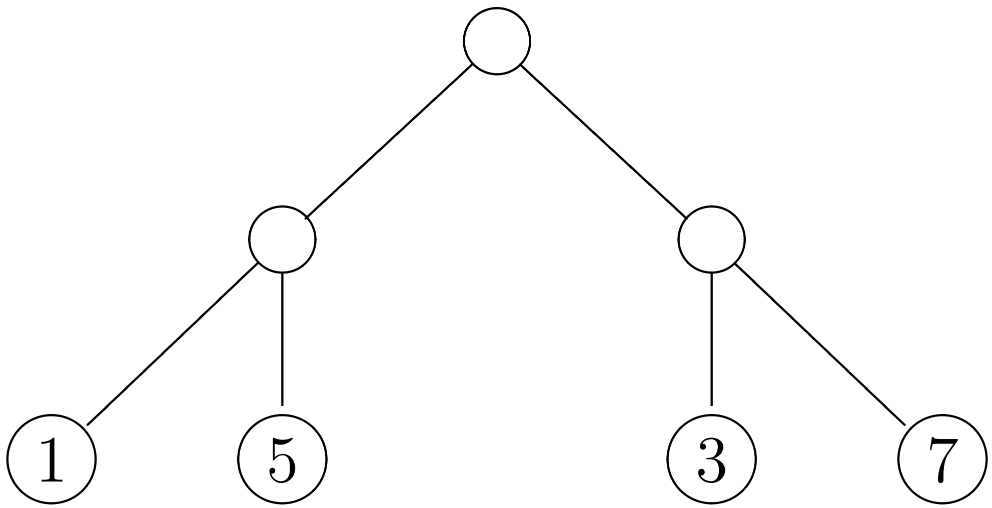
Jevgeni Kabanov

Viinistu, 2005

Catamorphism (fold)

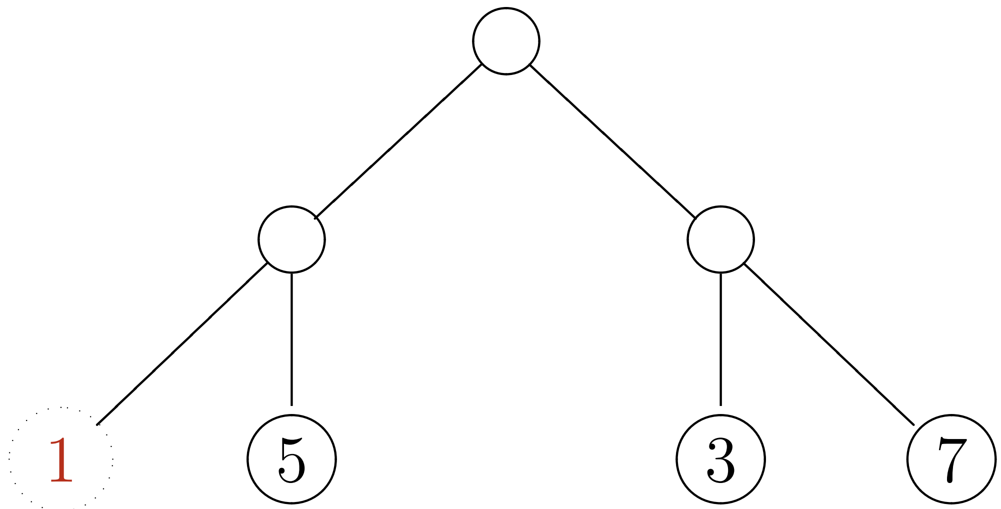
- Structural recursion combinator
- Generic *foldr* (Haskell)
- Eats (folds) trees from bottom-up, producing combined result
- Similar to Visitor pattern in OOP, but doesn't update structures

Sum fold animation

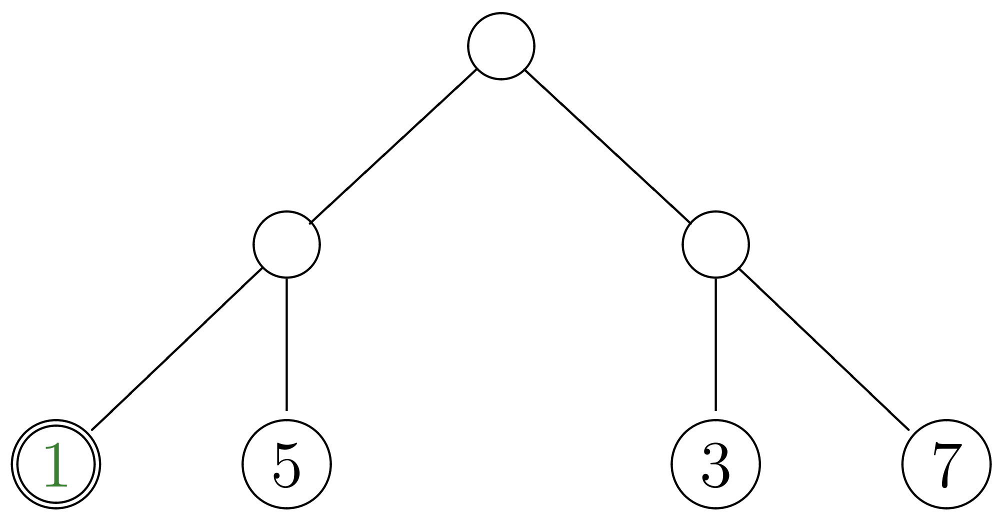


Let's count this tree sum...

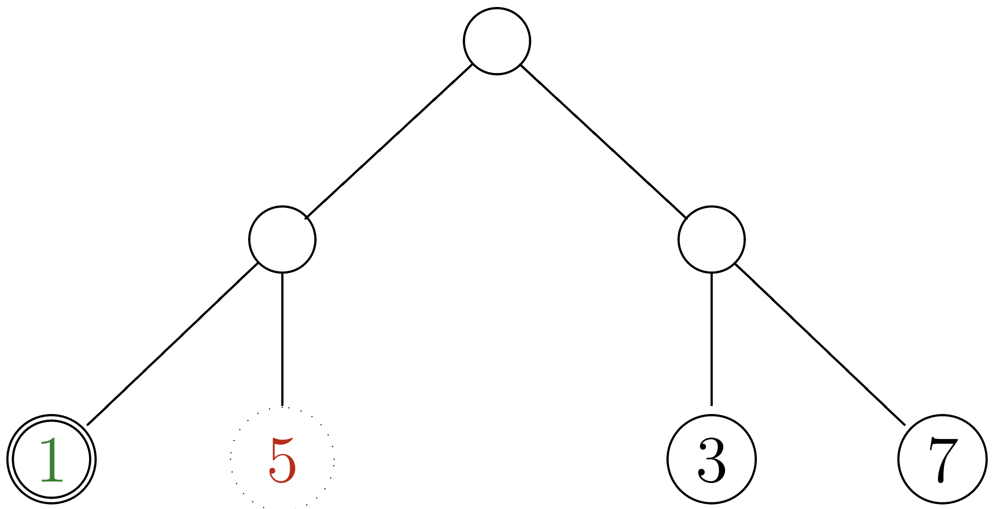
Sum fold animation



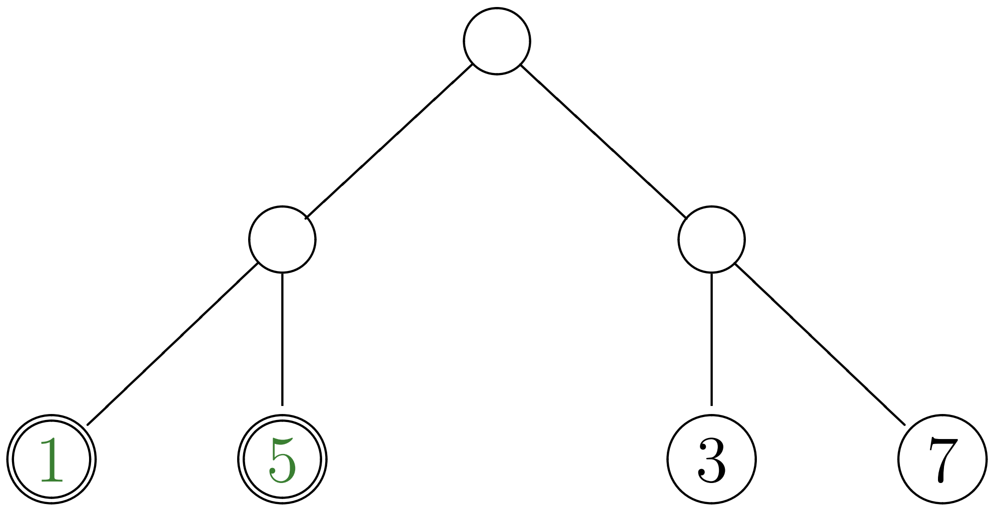
Sum fold animation



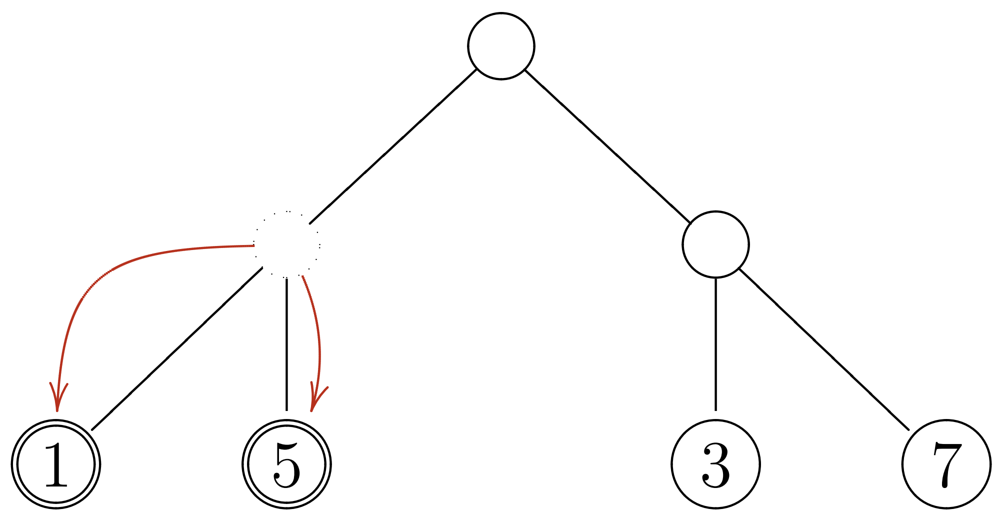
Sum fold animation



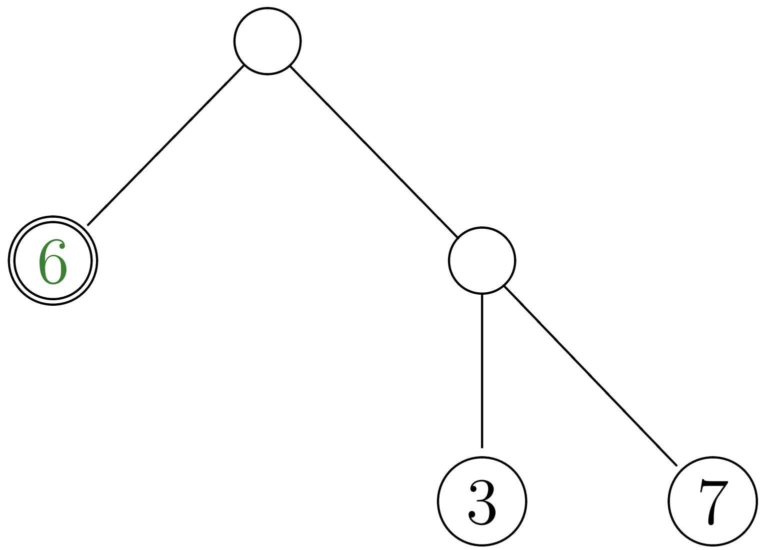
Sum fold animation



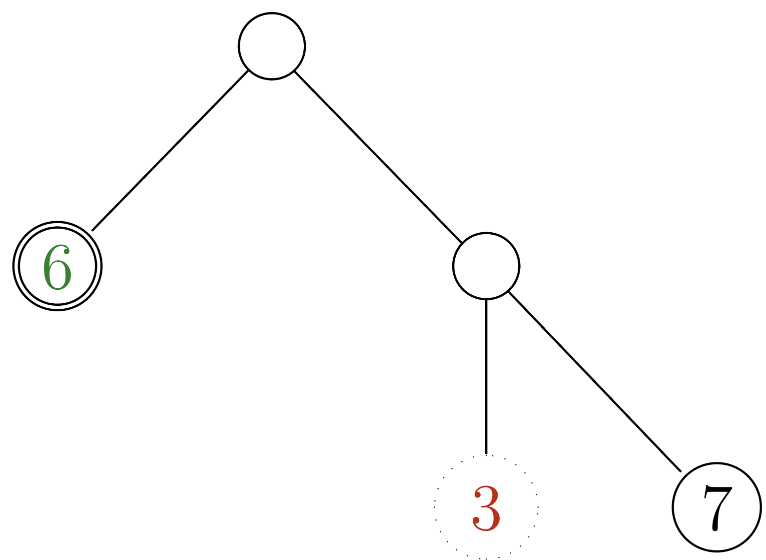
Sum fold animation



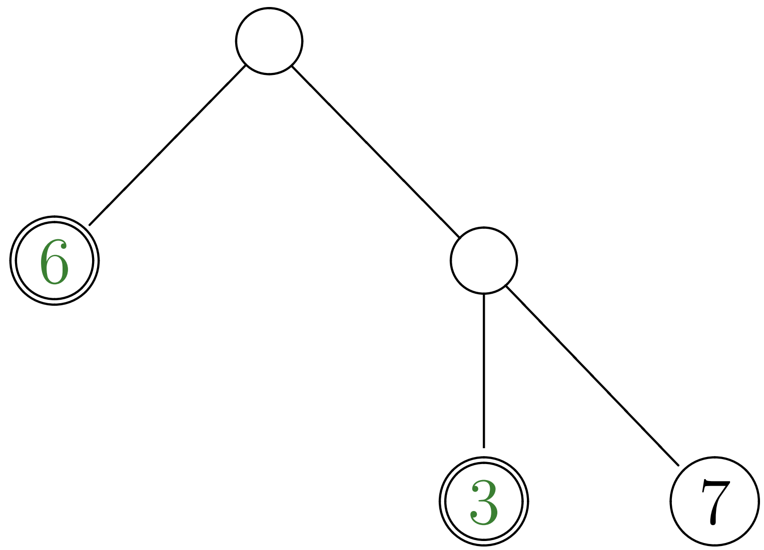
Sum fold animation



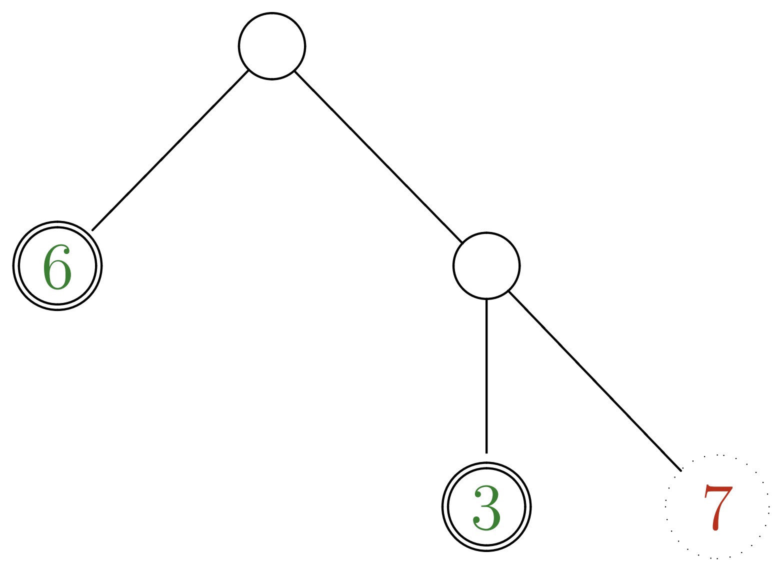
Sum fold animation



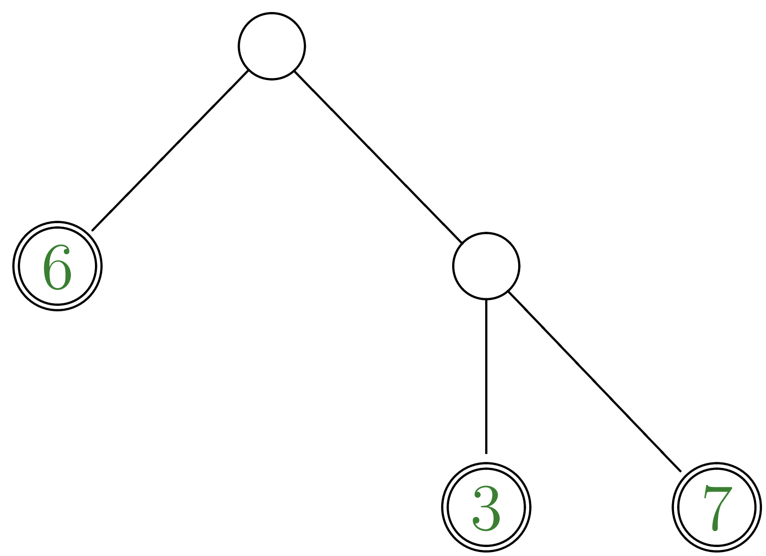
Sum fold animation



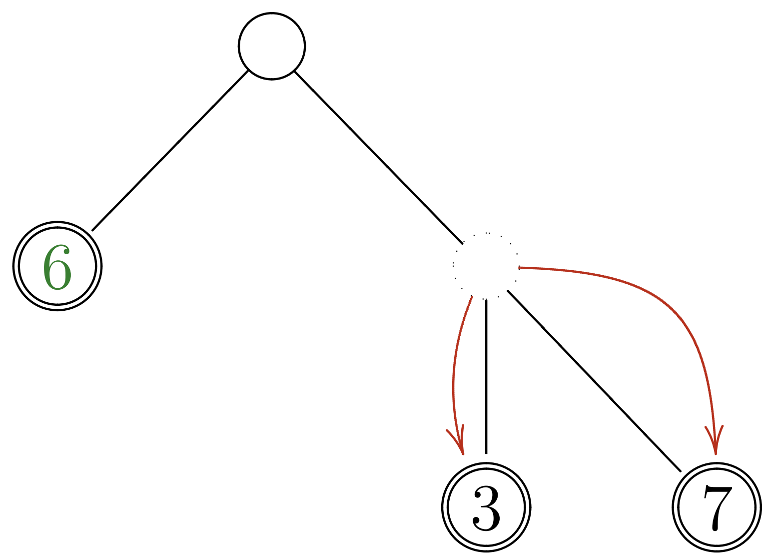
Sum fold animation



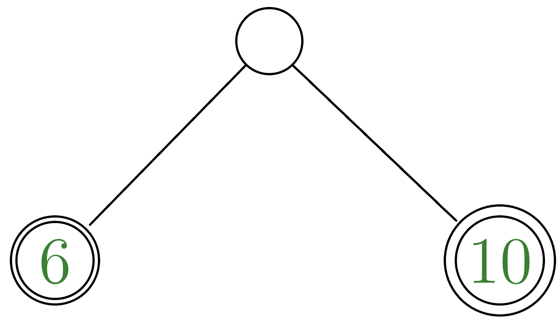
Sum fold animation



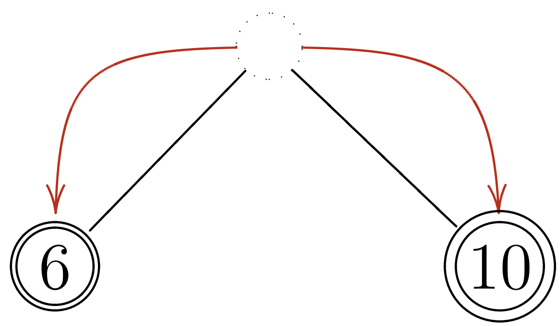
Sum fold animation



Sum fold animation



Sum fold animation



Sum fold animation

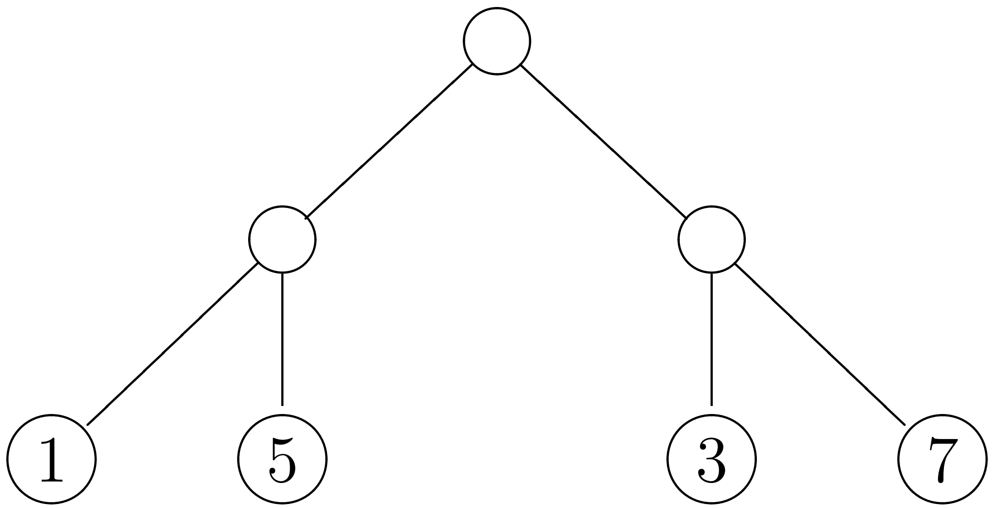
16

And the result is $16 = 1 + 5 + 3 + 7$

Histomorphism

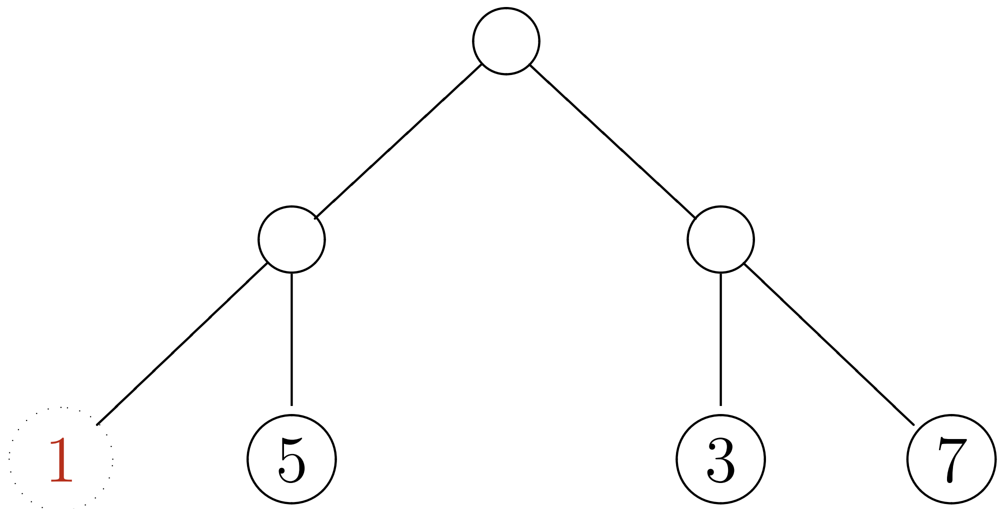
- Introduced by Varmo & Tarmo in 1999
- Course-of-value structural recursion combinator
- Inspired by dynamic programming technique
- Moves bottom-up annotating the tree with results
- Allows to reuse sub(-sub)* node results
- Finally collapses the tree producing the end result

Funny sum histo animation

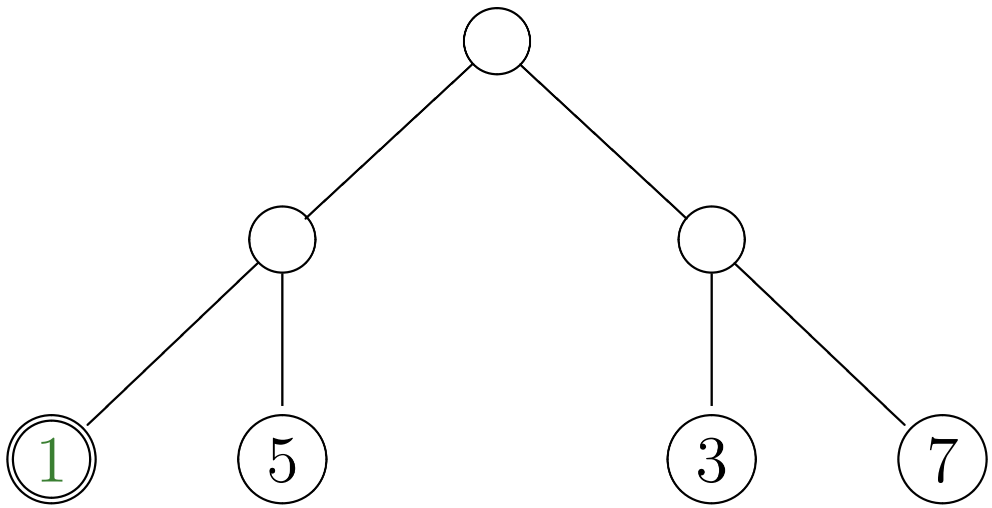


Let's count this tree (funny) sum...

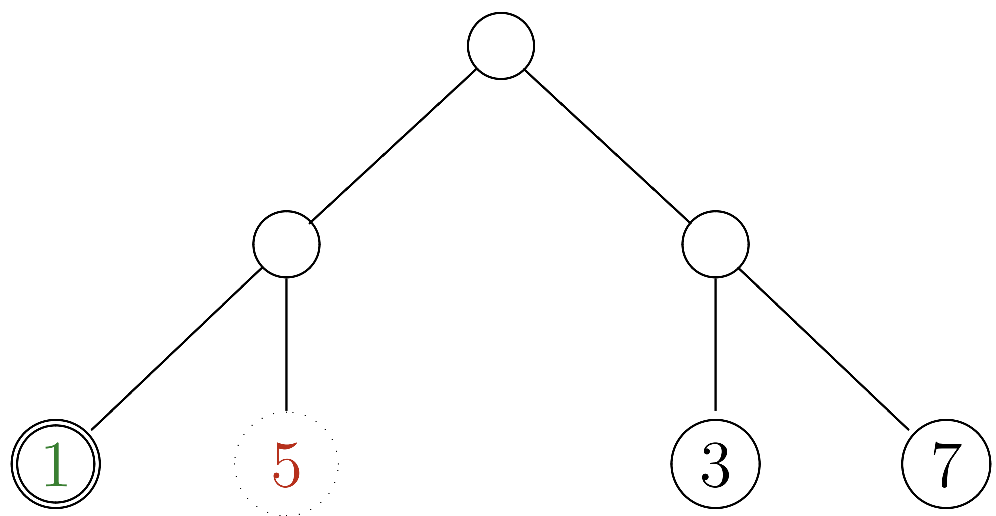
Funny sum histo animation



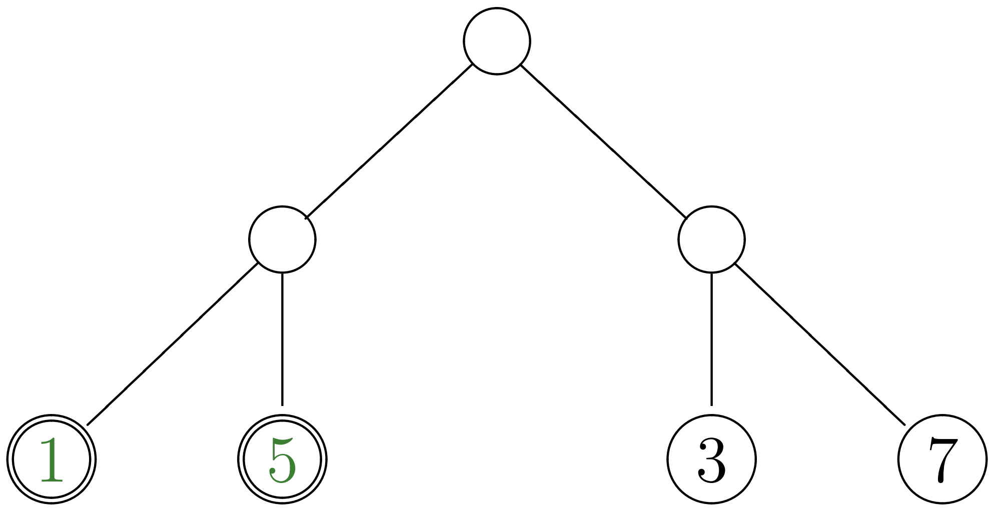
Funny sum histo animation



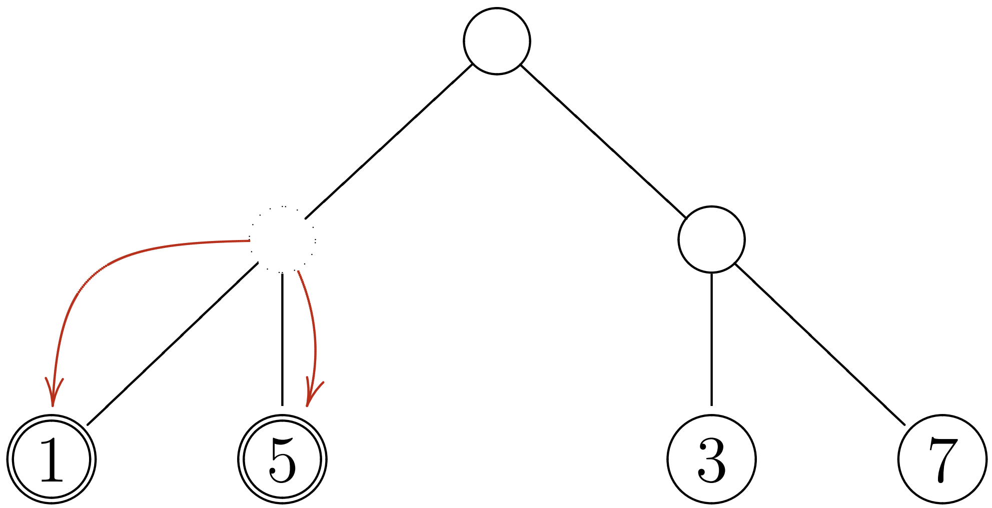
Funny sum histo animation



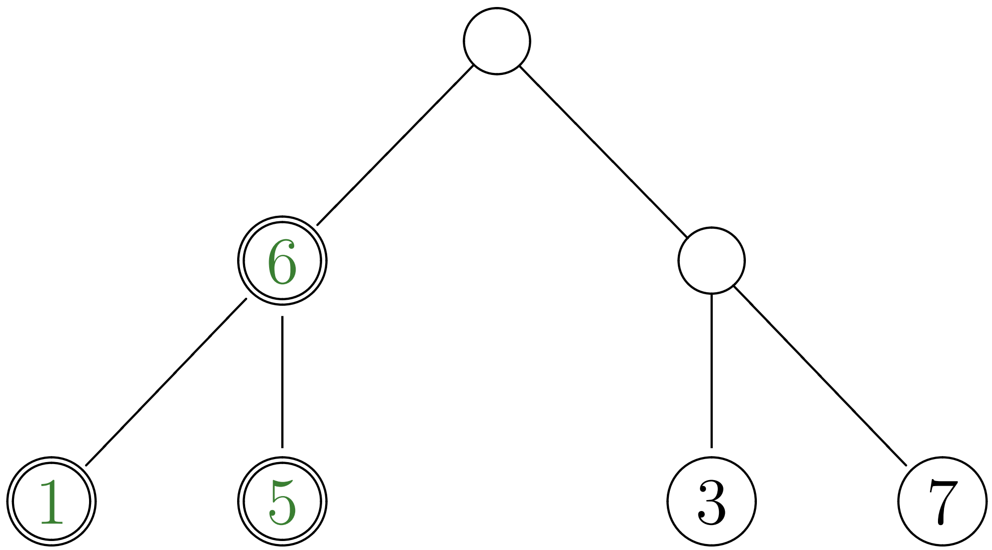
Funny sum histo animation



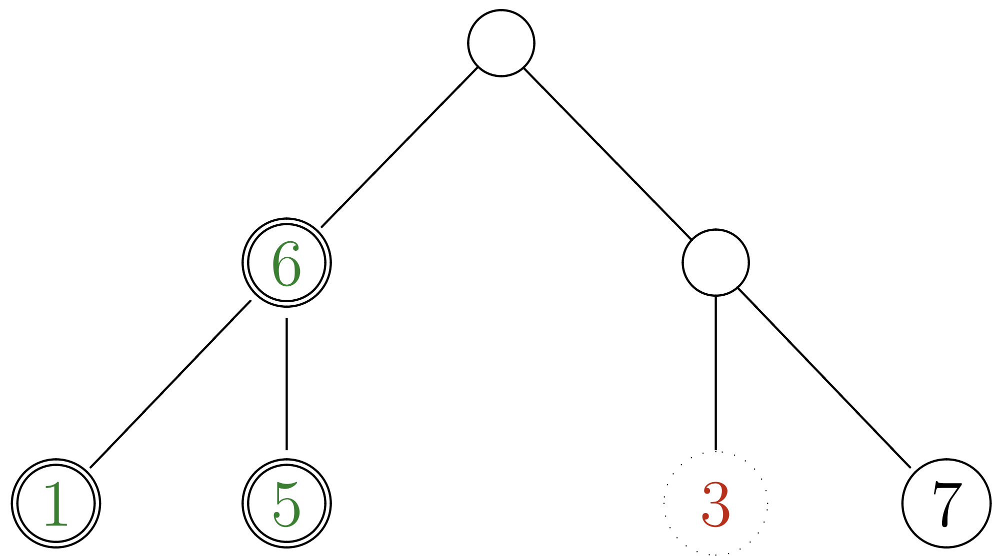
Funny sum histo animation



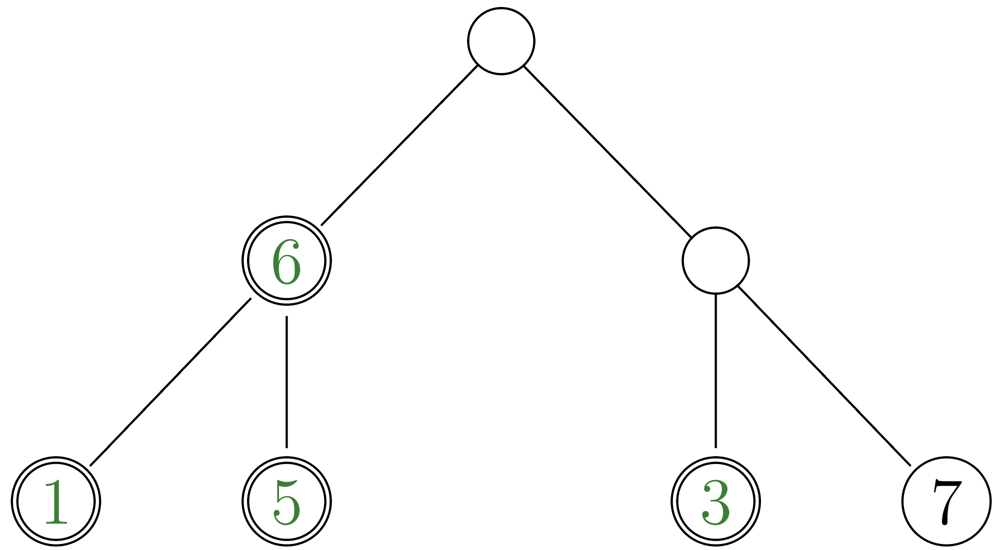
Funny sum histo animation



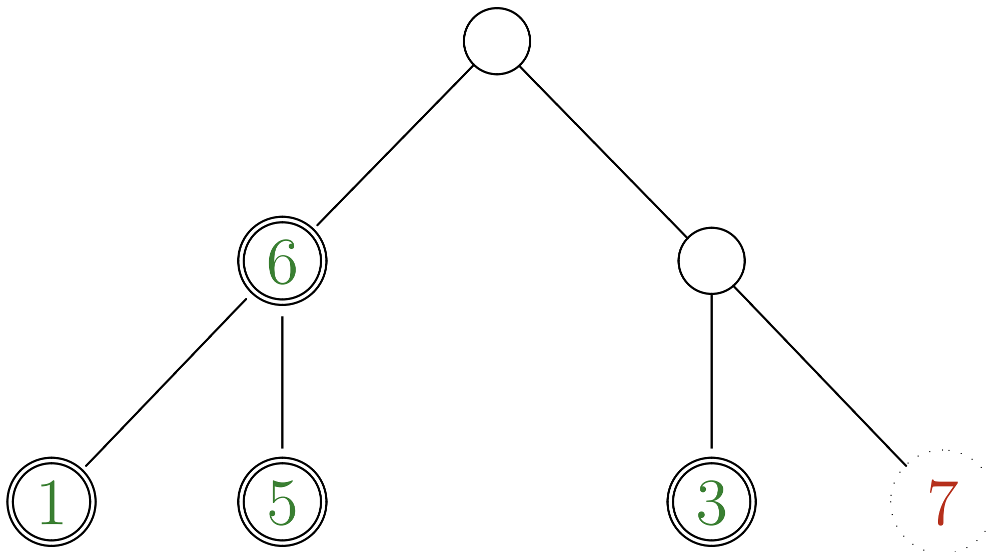
Funny sum histo animation



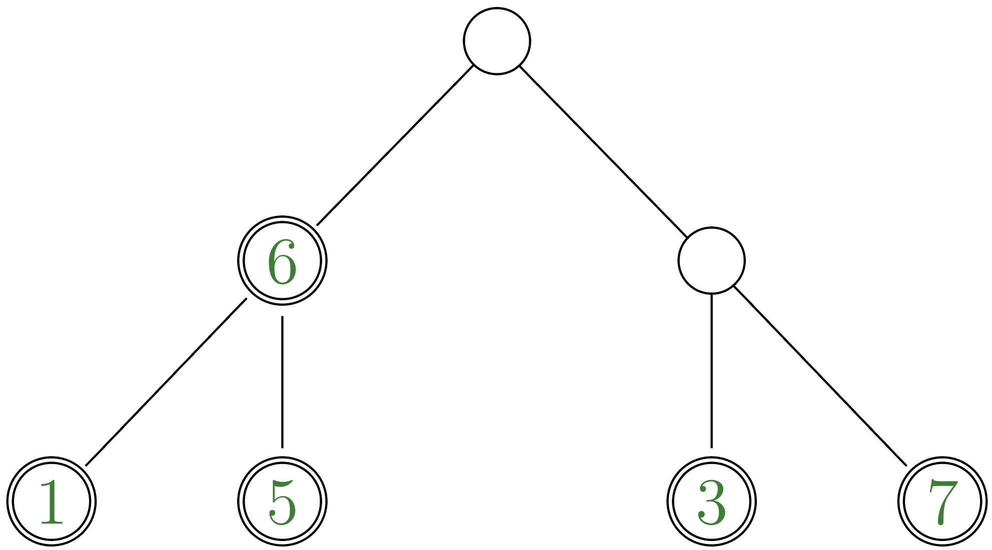
Funny sum histo animation



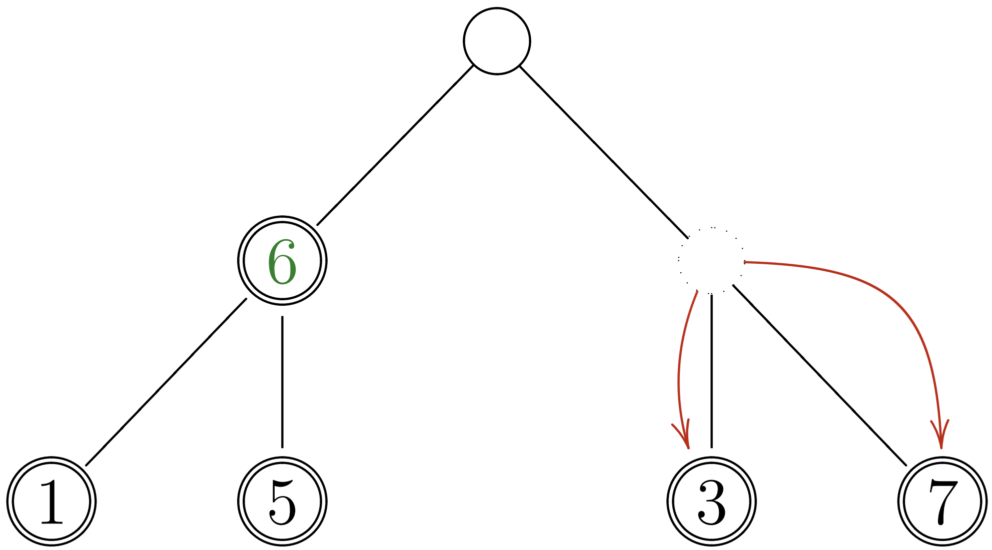
Funny sum histo animation



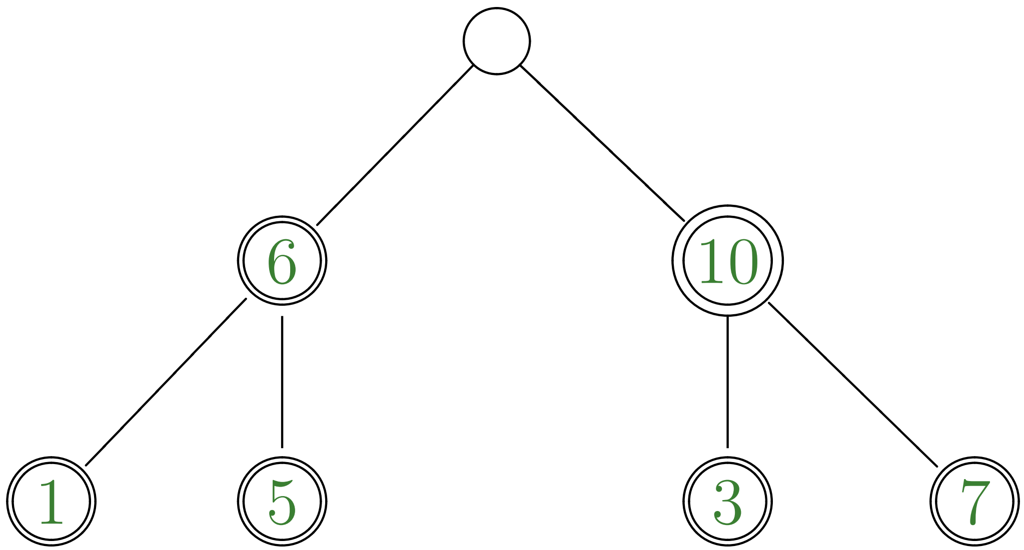
Funny sum histo animation



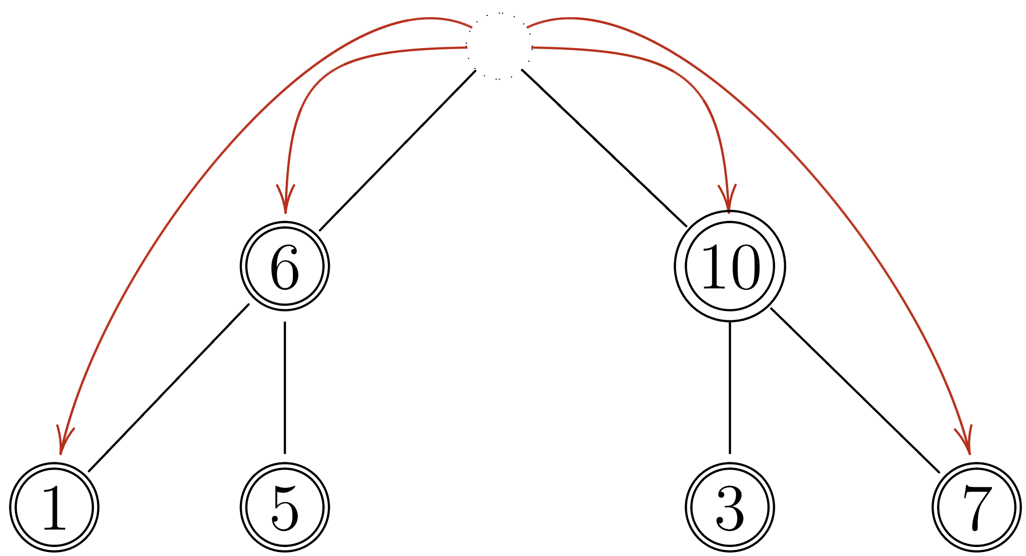
Funny sum histo animation



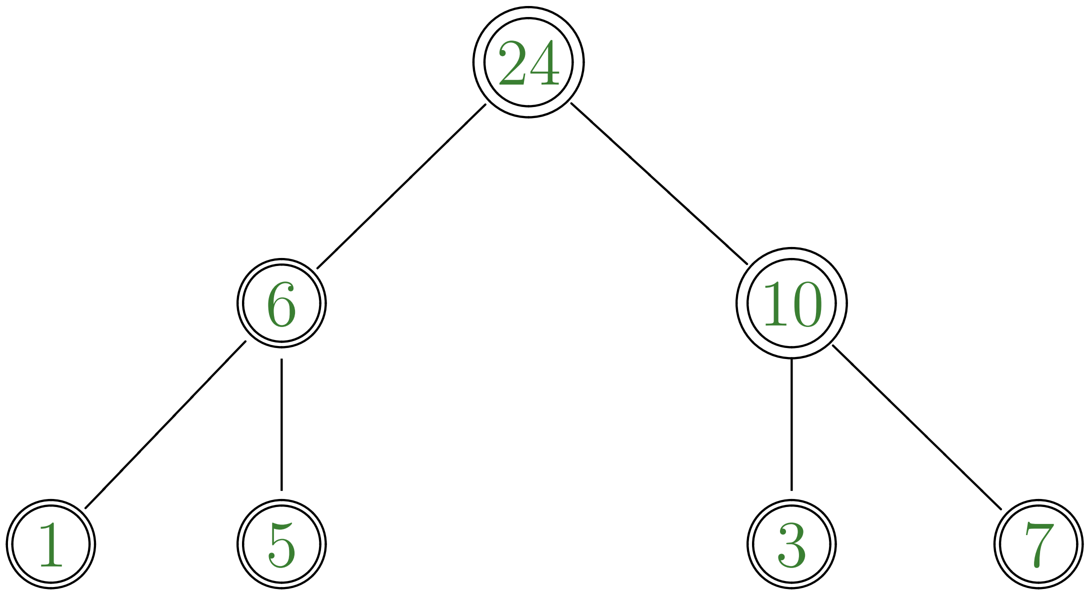
Funny sum histo animation



Funny sum histo animation



Funny sum histo animation



Funny sum histo animation

24

And the result is $24 = 1 \times 2 + 5 + 3 + 7 \times 2$

Generic hylomorphism

- General recursion combinator
- 2 stages:
  1. Build an intermediate structure using *unfold*
  2. Collapse the intermediate structure using *fold*
- The intermediate structure corresponds to the implicit call tree
- The intermediate structure does not really have to be built

Dynamic hylomorphism

- Dynamic recursion combinator
- The *fold* is replaced by the histomorphism

$$
\begin{tikzcd}
\mathrm{F}A \arrow[d, "\mathrm{F}[\![(\varphi)]\!]"'] & A \arrow[l, "\varphi"'] \arrow[d, "{[\![(\varphi)]\!]}"] \arrow[dd, bend left=60, "f"] \\
\mathrm{F}\mu\mathrm{F} \arrow[r, "\mathtt{in}"] \arrow[d, "\mathrm{F}[\![(\langle \{\!|\,\psi\,|\!\}, \mathtt{in}^{-1} \rangle)]\!]"'] & \mu\mathrm{F} \arrow[d, "\{\!|\,\psi\,|\!\}"] \\
\mathrm{F}(\mathrm{F}^{\nu}(B)) \arrow[r, "\psi"] & B
\end{tikzcd}
$$

Challenges

- Histomorphism expressive power
- Dynamic hylomorphism expressive power
- Properties of transformation to dynamic recursion
- Deriving dynamic definition

Case study

- Fibonacci numbers
- Binary partition number
- Levenshtein (Edit) distance
- Longest common subsequence

- Only first two can be defined as pure histomorphisms
- General recursion is needed

Inspiration

Fibonacci dependency tree

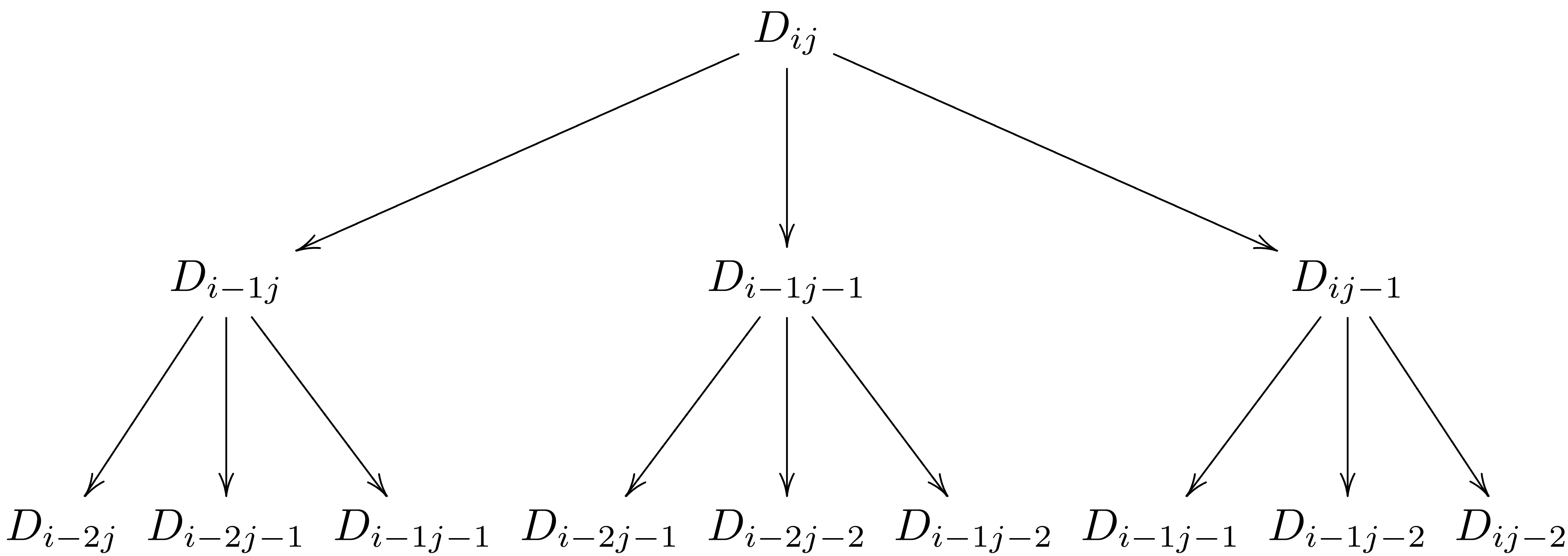
$$
\begin{tikzcd}
 & & & n \arrow[dl] \arrow[dr] & & & \\
 & & n-1 \arrow[dll] \arrow[d] & & n-2 \arrow[d] \arrow[drr] & & \\
n-2 & & n-3 & & n-3 & & n-4
\end{tikzcd}
$$

Collapsed dependency graph

$$
\begin{tikzcd}
n \arrow[r] \arrow[rr, bend left] & n-1 \arrow[r] \arrow[rr, bend left] & n-2 \arrow[r] \arrow[rr, bend left] & n-3 \arrow[r] & \cdots
\end{tikzcd}
$$

Inspiration (2)

Levenshtein (Edit) distance dependency tree



Inspiration (3)

Levenshtein (Edit) distance collapsed dependency graph

$$
\begin{tikzcd}
 & & & D_{ij} \arrow[dl] \arrow[dr] \arrow[dd] & & & \\
 & & D_{i-1j} \arrow[dl] \arrow[dd] \arrow[dr] & & D_{ij-1} \arrow[dl] \arrow[dd] \arrow[dr] & & \\
 & D_{i-2j} \arrow[dl] \arrow[dr] & & D_{i-1j-1} \arrow[dl] \arrow[dr] & & D_{ij-2} \arrow[dl] \arrow[dr] & \\
D_{i-3j} & & D_{i-2j-1} & & D_{i-1j-2} & & D_{ij-3}
\end{tikzcd}
$$

Transformation

- Original definition: $f = \psi \circ \mathrm{T} f \circ \varphi$
- Dynamic definition: $f = \psi \circ \sigma \circ \mathrm{T}' [\![(\langle f, \mathtt{in}^{-1} \rangle)]\!] \circ \varphi'$
  - φ' generates more compact intermediate structure
  - T' defines the structure recursive pattern
  - σ restores one level of the old structure
  - σ and T' are uniquely determined by φ'
- The consumer (algebra) part is preserved
- The producer (coalgebra) part is consistently updated

Dependency algebra

Let

- Original dependency producers: $h_i : A \to A$
- Dynamic dependency producers: $h'_j : A \to A$
- Projections: $\pi_i : \mathrm{T}^\nu(C) \to \mathrm{T}^\nu(C)$,
  $\pi_i = [\mathtt{in}, \mathrm{out}_i \circ \mathrm{outr}] \circ \mathtt{in}^{-1}$
- Deep projections:
  $\pi_i^* = \mathrm{outl} \circ \pi'_{k_l} \circ \pi'_{k_{l-1}} \circ \cdots \circ \pi'_{k_2} \circ \mathrm{out}_{k_1}$
- Induction indicator: $p : A \to \mathrm{Bool}$

Dependency algebra (2)

Then

- $\varphi = (id + \langle id, h_1, h_2, \ldots, h_n \rangle) \circ p?$
- $\varphi' = (id + \langle id, h'_1, h'_2, \ldots, h'_m \rangle) \circ p'?$
- $\sigma = [\mathrm{inl}, (\mathrm{out}_0 + \langle \mathrm{out}_0, \pi^*_1, \pi^*_2, \ldots, \pi^*_n \rangle) \circ (p \circ \mathrm{out}_0)?]$

And φ' has to satisfy following for each $i \in I$, each $s \in S$:

$$P(s, i) = \langle k_1, k_2, \ldots, k_l \rangle \in J^*$$

$$\mathrm{outl} \circ \pi_i \circ [\![(\langle id, \varphi \rangle)]\!] = \mathrm{outl} \circ \pi'_{k_l} \circ \pi'_{k_{l-1}} \circ \cdots \circ \pi'_{k_1} \circ [\![(\langle id, \varphi' \rangle)]\!]$$

$$h_i(s) = h'_{k_l} \circ h'_{k_{l-1}} \circ \cdots \circ h'_{k_1}(s)$$

Future work

- More categorical approach to transformation
- A solid proof for dependency algebra
- (Semi)-automatical derivation for restricted cases