Fast Left Kan Extensions Using The Chase

Ryan Wisnesky, David I. Spivak

Conexus AI

Applied Category Theory at Oxford 2019

$$\Sigma \dashv \Delta \dashv \Pi$$

Introduction

- Left Kan extensions have many uses:
  - to enumerate finitely-presented algebraic structures such as monoids;
  - to construct semi-decision procedures for Thue systems;
  - to compute the cosets of a group;
  - to compute the orbits of a group action;
  - to compute quotients of sets, and more.
- We present a new algorithm for computing left Kan extensions based on the venerable "chase" algorithm from relational database theory.
- Part 1 defines left Kan extensions, the chase, and how to compute the former using the latter.
- Part 2 describes a particular chase algorithm implementation and gives performance results.

Left Kan Extensions

- Given functors $F : C \to D$ and $J : D \to \mathsf{Set}$, define:

$$\Delta_F : \mathsf{Set}^D \to \mathsf{Set}^C \;:=\; - \circ F \qquad C \xrightarrow{F} D \underbrace{\xrightarrow{J}}_{\Delta_F(J)} \mathsf{Set}$$

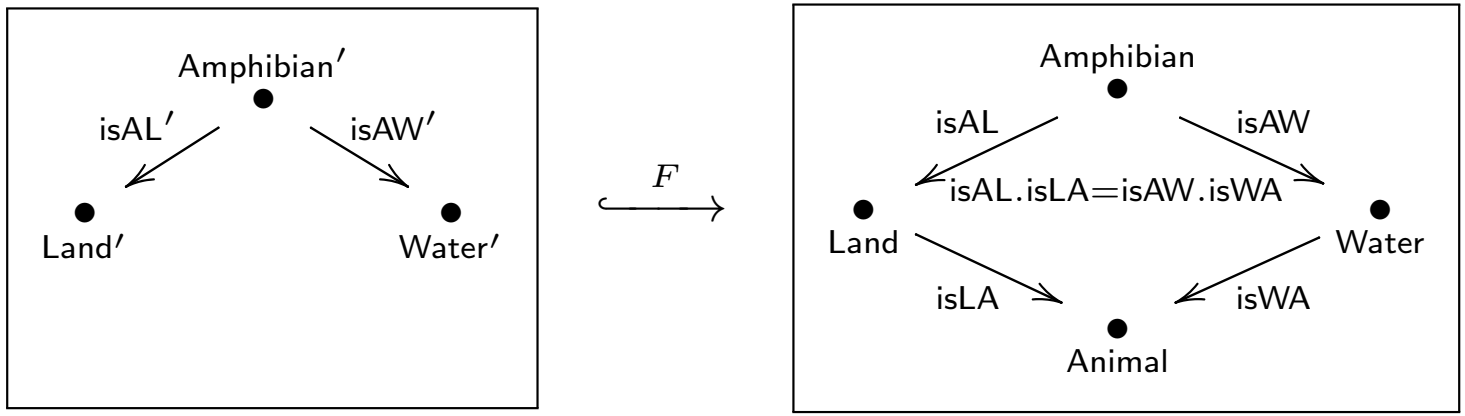
- Δ_F has a left adjoint, $\Sigma_F : \mathsf{Set}^C \to \mathsf{Set}^D$.
- For each functor $I : C \to \mathsf{Set}$, the functor $\Sigma_F(I) : D \to \mathsf{Set}$ is the *left Kan extension* of I along F.
- Left Kan extensions always exist, but $\Sigma_F(I)(d)$ may be infinite for particular d.
- Computing finite left Kan extensions when C, D, and F are finitely presented and $I(c)$ is finite for every c is a semi-computable problem first studied by Carmody et al.
  - The formula for left Kan extensions quantifies over infinitely morphisms.
  - When D is finite, the problem is computable.

Motivation: Kan Extensions and Data Migration

- In data migration, $F : C \to D$ is interpreted as a "schema mapping", and I as a C-database to be migrated to D.
- Left Kan algorithms bear striking resemblances to *chase* algorithms from relational data migration.
- Our contribution is to formalize the above observation, enabling fast computation of left Kan extensions via the fast chase engines built by the database community.
- Our in-house chase engine is part of the open-source categorical query language CQL, see `http://categoricaldata.net`.
- Δ_F also has a right adjoint related to joins, which we do not discuss.

Example

- The functor $F : C \to D$ is:



- The functor $I : C \to$ Set is:

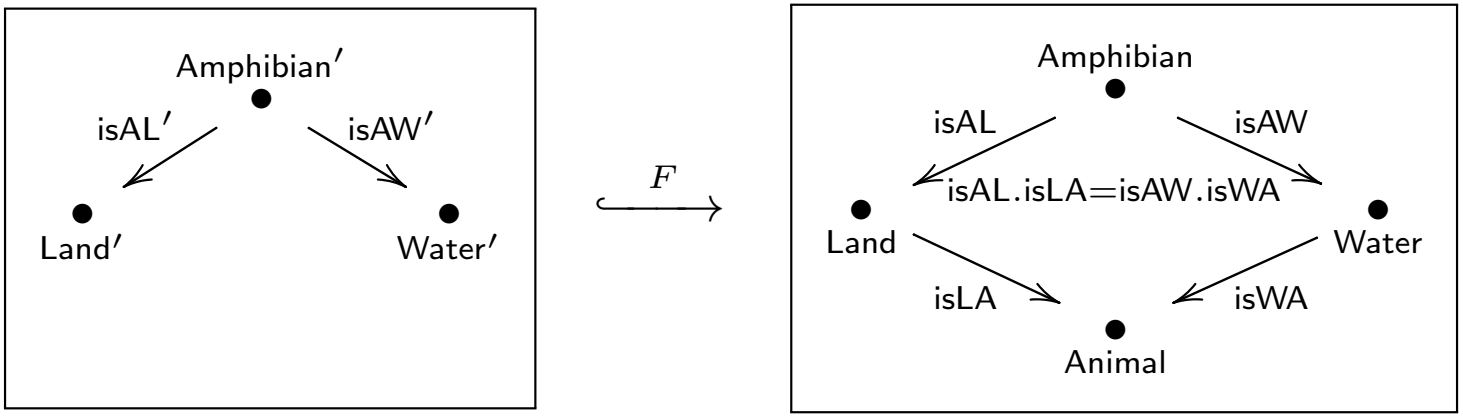
Amphibian'	isAL'	isAW'
gecko	lizard	salamander
frog	toad	newt

Land'
lizard
toad
human
cow
horse

Water'
fish
salamander
newt
dolphin

- Frogs are both toads and newts, and geckos are both lizards and salamanders, so we expect $5 + 4 - 2 = 7$ animals in $\Sigma_F(I)$.

Example, continued



$\Sigma_F(I) =$

Amphibian	isAL	isAW
gecko	lizard	salamander
frog	toad	newt

Land	isLA
lizard	gecko
toad	frog
human	human
cow	cow
horse	horse

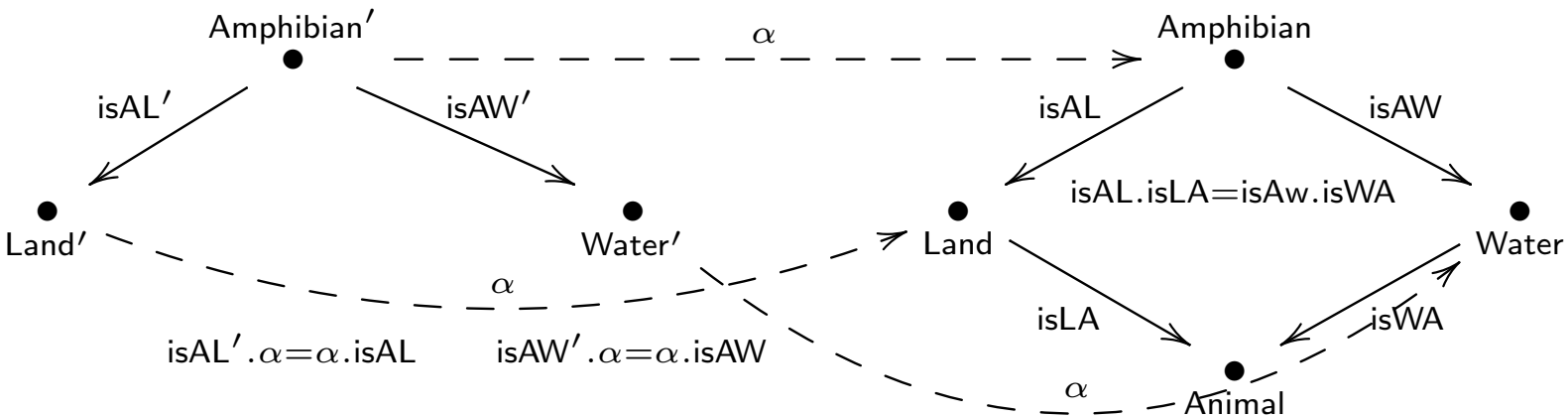
Water	isWA
fish	fish
salamander	gecko
newt	frog
dolphin	dolphin

Animal
fish
frog
dolphin
human
cow
horse
gecko

The Collage of a Functor

- The *collage* $\mathsf{col}(F)$ of a functor $F : C \to D$ is the category:

$$\mathsf{Ob}(\mathsf{col}(F)) := \mathsf{Ob}(C) + \mathsf{Ob}(D)$$
$$\mathsf{Hom}(\mathsf{col}(F)) := \mathsf{Hom}(C) + \mathsf{Hom}(D) + \{\alpha_c : c \to F(c),\ \forall c \in \mathsf{Ob}(C)\}$$
$$F(f) \circ \alpha_c = \alpha_{c'} \circ f,\ \forall f : c \to c' \in \mathsf{Hom}(C)$$



Regular and Finite Limit Theories

- A *regular theory* consists of
  - a set $s_1, \ldots, s_j$ of *sorts*,
  - a set $p_1, \ldots, p_k$ of *relation symbols*, and
  - a set of *axioms* of the form:

$$\forall (x_0 : s_0) \cdots (x_n : s_n).\, \phi(x_0, \cdots, x_n) \Rightarrow \exists (x_{n+1} : s_{n+1}) \cdots\ (x_m : s_m).\, \psi(x_0, \ldots, x_m)$$

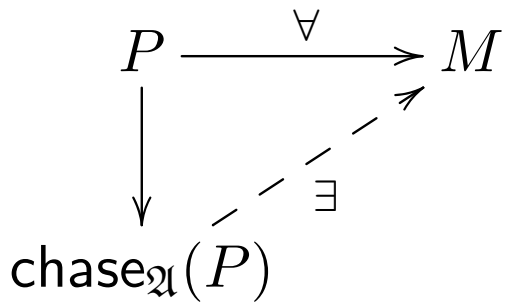
where ϕ and ψ are conjunctions of:

- truth, or
- assertions $x = x'$, for variables of the same sort, or
- assertions $p(x, \ldots, x')$, for variables of appropriate sort.

- By replacing every $\exists$ with $\exists!$, we obtain *finite limit theories*.

The Chase

- Given regular theory $\mathfrak{A}$ and pre-model P, to *chase* P by $\mathfrak{A}$ is to construct a model $\mathrm{chase}_{\mathfrak{A}}(P)$ and morphism $p : \kappa \to \mathrm{chase}_{\mathfrak{A}}(\kappa)$ such that for any model M and morphism $m : P \to M$, there is a possibly non-unique morphism $h : \mathrm{chase}_{\mathfrak{A}}(\kappa) \to M$ such that $h \circ p = m$.



- In database theory, $\mathrm{chase}_{\mathfrak{A}}(\kappa)$ is called a *universal solution*.
- Warning: Database theory frequently departs from model theory!
- Key lemma: on a finite limit theory, the morphism h is unique.

Left Kan Extensions Using the Chase

- Main result: left Kan extensions can be computed by the chase:

$$\mathsf{chase}_{\mathsf{col}(F)}(\Delta_i(I)) \cong (I, \Sigma_F(I), \eta_I : I \to \Delta_F(\Sigma_F(I))$$

  where η is the unit of the $\Sigma_F \dashv \Delta_F$ adjunction and $i : C \hookrightarrow \mathsf{col}'(F)$ is the inclusion of C into "$\mathsf{col}(F)$ without equations".
- Algorithm:
  - First, consider $I : C \to \mathsf{Set}$ as a pre-model of $\mathsf{col}(F)$.
  - Second, chase, to obtain a model of $\mathsf{col}(F)$.
  - Finally, project from the model of $\mathsf{col}(F)$ to obtain $\Sigma_F(I)$ and η_I.

CQL's Chase Engine

- Fast chase engines exist, but currently none meet CQL's needs:
  - support for all of finite limit logic (requirement)
  - emitting a *term model* as output (nice to have)
  - determinism (nice to have)
  - unfailing: no ID / null distinction (requirement)
- So, we built our own algorithm, resembling a parallel version of Carmody et al's algorithm.
- It is performant enough to execute multi-gb real world data science use cases (joint work with Kris Brown).

Input Specification

The data below comes from functors $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ and $\mathcal{I} : \mathcal{C} \to \text{Set}$:

- A finite set C of *nodes*.
- For each $c_1, c_2 \in C$, a finite set $C(c_1, c_2)$ of *edges*.
- For each $c_1, c_2 \in C$, a finite set $CE(c_1, c_2)$ of pairs of *paths* $c_1 \to c_2$.
- A finite set D of *nodes*.
- For each $d_1, d_2 \in D$, a finite set $D(d_1, d_2)$ of *edges*.
- For each $d_1, d_2 \in D$, a finite set $DE(d_1, d_2)$ of pairs of paths $d_1 \to d_2$.
- A function $F : C \to D$.
- For each $c_1, c_2 \in C$, a function F_{c_1,c_2} from edges in $C(c_1, c_2)$ to paths $F(c_1) \to F(c_2)$ in D.
- For each $c \in C$, a set $I(c)$.
- For each edge $g : c_1 \to c_2 \in C$, a function $I(c_1) \to I(c_2)$.

Output Specification

- Our chase runs in rounds, possibly forever, transforming a state until a fixed point is reached.
- Termination is undecidable, but conservative criteria exist.
- The output state consists of:
  1. For each $d \in D$, a set $J(d)$, initialized to $\bigsqcup_{\{c \in C \mid F(c)=d\}} I(c)$.
  2. For each $d \in D$, an equivalence relation $\sim_d \subseteq J(d) \times J(d)$, initialized to identity.
  3. For each $f : d_1 \to d_2 \in D$, a relation $J(f) \subseteq J(d_1) \times J(d_2)$, initialized to empty.
  4. For each $c \in C$, a function $\eta(c) : I(c) \to J(F(c))$, initialized to the injections from the first item, i.e., $\eta(c)(x) = (c, x)$.

The Chase Step

- Given a path $p : d_1 \to d_2$ in D, we may *evaluate* p on any $x \in J(d_1)$ to $p(x) \subseteq J(d_2)$.
- Each round consists of the following sequence of actions:
  1. For each edge $g : d_1 \to d_2$ in D and $x \in J(d_1)$ for which there does not exist $y \in J(d_2)$ with $(x, y) \in J(g)$, add a fresh symbol g(x) to $J(d_2)$ and add $(x, \mathsf{g(x)})$ to $J(g)$.
  2. For each $p = q$ in $DE(d_1, d_2)$ and $x \in J(d_1)$, update $\sim_{d_2}$ to include $\{(x', x'') | x' \in p(x), x'' \in q(x)\}$.
  3. Same as above, but for each $p = q$ in col(F).
  4. For each (x, y) and (x, y') in $J(f)$ for some $f : d_1 \to d_2$ in D with $y \neq y'$, update $\sim_{d_2}$ to include (y, y').
  5. Replace each element in J and η with its equivalence class representative from $\sim$.

Example Chase I

Land	isLA
lizard	
toad	
human	
cow	
horse	

Water	isWA
fish	
dolphin	
salamander	
newt	

Amphibian	isAL	isAW
gecko		
frog		

Animal

$\Downarrow (\exists!)$

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA(cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
dolphin	isWA(dolphin)
salamander	isWA(salamander)
newt	isWA(newt)

Amphibian	isAL	isAW
gecko	isAL(gecko)	isAW(gecko)
frog	isAL(frog)	isAW(frog)

Animal

Example Chase II

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA(cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
dolphin	isWA(dolphin)
salamander	isWA(salamander)
newt	isWA(newt)

Amphibian	isAL	isAW
gecko	isAL(gecko)	isAW(gecko)
frog	isAL(frog)	isAW(frog)

Animal

$\Downarrow (\alpha)$

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA(cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
dolphin	isWA(dolphin)
salamander	isWA(salamander)
newt	isWA(newt)

Amphibian	isAL	isAW
gecko	lizard	salamander
frog	toad	newt

Animal

Example Chase III

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA(cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
dolphin	isWA(dolphin)
salamander	isWA(salamander)
newt	isWA(newt)

Amphibian	isAL	isAW
gecko	lizard	salamander
frog	toad	newt

Animal

$\Downarrow (\exists!)$

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA(cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
dolphin	isWA(dolphin)
newt	isWA(newt)
salamander	isWA(salamander)

Amphibian	isAL	isAW
gecko	lizard	salamander
frog	toad	newt

Animal
isLA(lizard)
isLA(toad)
isLA(human)
isLA(cow)
isLA(horse)
isWA(fish)
isWA(dolphin)
isWA(salamander)
isWA(newt)

Example Chase IV

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA(cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
salamander	isWA(salamander)
newt	isWA(newt)
dolphin	isWA(dolphin)

Amphibian	isAL	isAW
gecko	lizard	salamander
frog	toad	newt

Animal
isLA(lizard)
isLA(toad)
isLA(human)
isLA(cow)
isLA(horse)
isWA(fish)
isWA(dolphin)
isWA(salamander)
isWA(newt)

$\Downarrow (DE)$

Land	isLA
lizard	isLA(lizard)
toad	isLA(toad)
human	isLA(human)
cow	isLA (cow)
horse	isLA(horse)

Water	isWA
fish	isWA(fish)
salamander	isWA(salamander)
newt	isWA(newt)
dolphin	isWA(dolphin)

Amphibian	isAL	isAW
gecko	lizard	salamander
frog	toad	newt

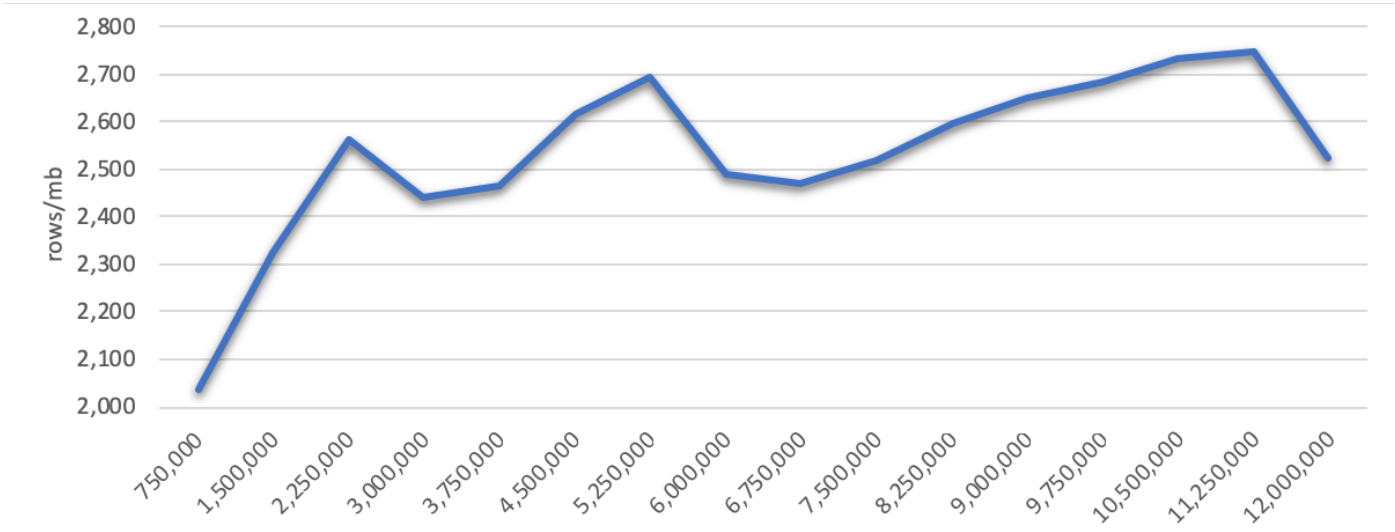
Animal
isLA(lizard)
isLA(toad)
isLA(human)
isLA (cow)
isLA(horse)
isWA(fish)
isWA(dolphin)

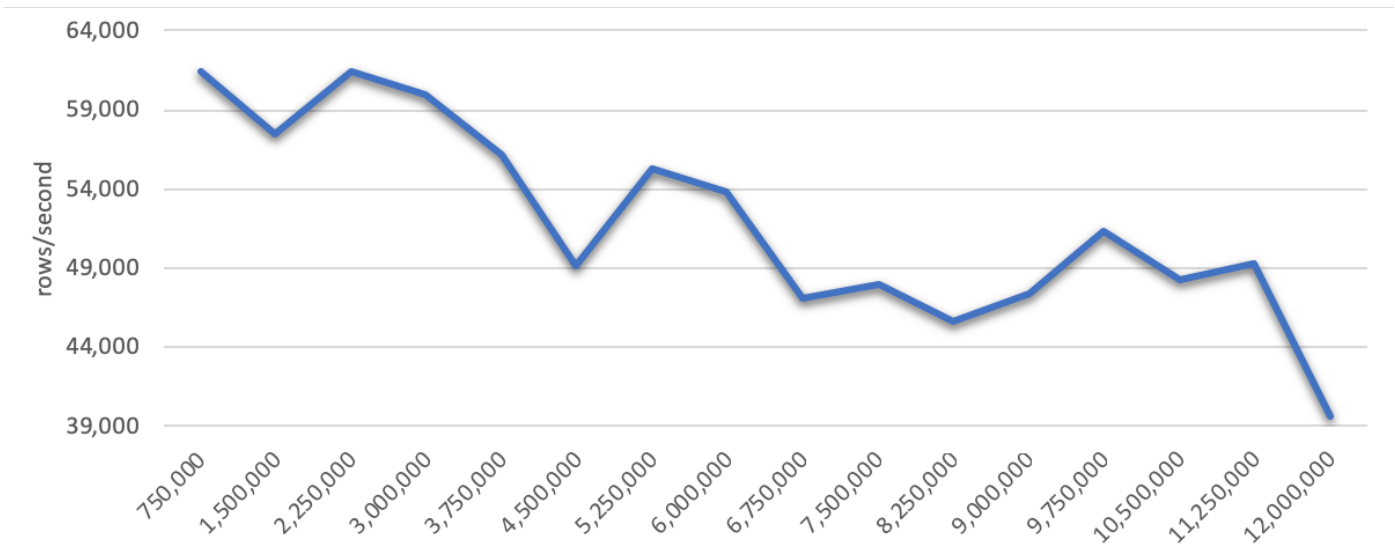
Implementation in CQL

- Our CQL implementation minimizes memory use by using either cardinalities or lists instead of sets:
  1. For each $d \in D$, a number $J(d) \geq 0$.
  2. For each $d \in D$, a list of length $J(d)$, where each element has the form (c, x, p), for some $c \in C$, $x \in I(c)$, and $p : F(c) \to d$.
  3. For each $d \in D$, a union-find data structure based on path-compressed trees $\sim_d \subseteq J(d) \times J(d)$.
  4. For each edge $f : d_1 \to d_2 \in D$, a list of length $J(d_1)$, each element of which is a number between 0 and $J(d_2)$.
  5. For each $c \in C$, a function $\eta(c) : I(c) \to J(F(c))$.
- The above state is more precisely considered as a functor to the *skeleton* of the category of sets.

CQL Benchmarks

- Scalability results on random instances of the example on a 13" 2018 MacBook Air with a 1.6ghz i5 CPU and 16gb RAM on Oracle Java 11:





Conclusion

- Left Kan algorithms bear striking resemblances to chase algorithms from relational data migration.
- We formalized the above observation, enabling, in theory, fast computation of left Kan extensions via the fast chase engines built by the database community.
- In practice, no current chase engines are right for CQL so we built a parallel version of Carmody et al's algorithm.
- Our in-house chase engine is part of the open-source categorical query language CQL, see `http://categoricaldata.net`.
- It is performant enough to execute multi-gb real world data science use cases (joint work with Kris Brown).
- We are looking for collaborators and/or customers! `info@conexus.ai`