# Amortized Analysis via Coalgebra

Harrison Grodin and Robert Harper

MFPS 2024

Carnegie Mellon University

# Acknowledgements

This work was improved through discussions with Max New, Yue Niu, and David Spivak and collaboration with Zachary Battleman, Runming Li, Parth Shastri, and Jonathan Sterling.

Sponsored by AFOSR grant FA9550-21-0009 (Tristan Nguyen, PM) and by NSF grant CCF-1901381.

**In a category of algebras,**

**amortized analyses are coalgebra morphisms.**

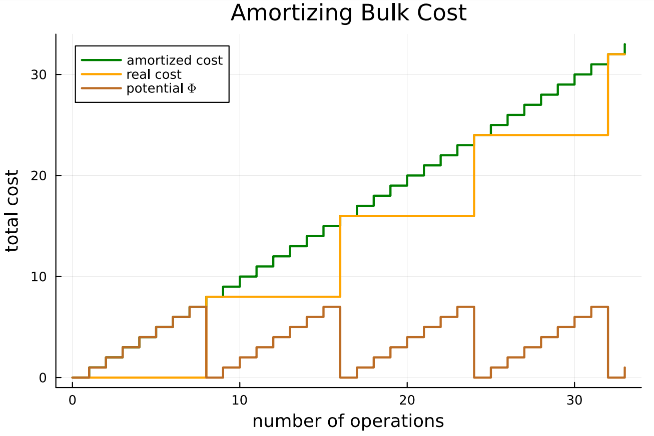
# Background

# Amortized Analysis

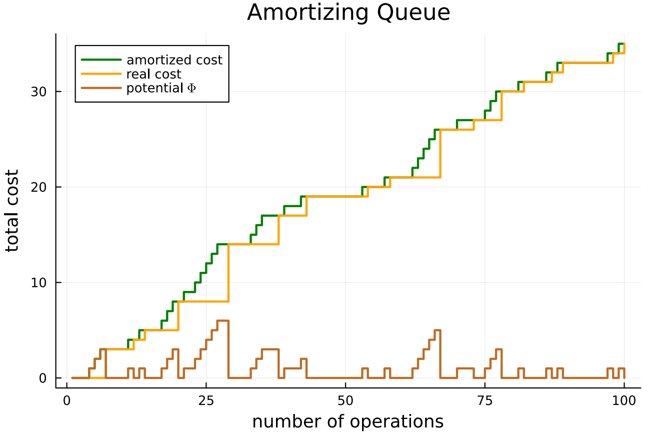
*In many uses of data structures, a* **sequence of operations**, *rather than just a single operation, is performed, and we are interested in the* **total time of the sequence**, *rather than in the times of the individual operations.*

—*Tarjan, 1985*

# Amortized Analysis (cont.)



# Amortized Analysis (cont.)



# Amortized Analysis: Potential Method

Let $d, d' \in D$ be states of a data structure. For each operation:

$$\textbf{amortized cost} = \textbf{real cost} + \Phi(d') - \Phi(d)$$

Here, $\Phi : D \to \mathbb{Z}$ is maps states to "potential", extra imagined up-front cost to offset big operations.

- Cheap operations save potential: $\Phi(d') > \Phi(d)$.
- Expensive operations spend potential: $\Phi(d') < \Phi(d)$.

# Abstract Cost Analysis via the Writer Monad

**calf** is an effectful dependent type theory for studying the cost and behavior of algorithms and data structures.

**Example**

$$
\begin{aligned}
&\mathit{isort} : \mathsf{list}(E) \rightharpoonup \mathsf{F}(\mathsf{list}(E)) \\
&\mathit{isort}\ [] = \mathsf{ret}([]) \\
&\mathit{isort}\ (x :: xs) = \\
&\quad \mathsf{bind}\, xs' \leftarrow \mathit{isort}\ xs\, \mathsf{in} \\
&\quad \mathit{insert}\ x\ xs'
\end{aligned}
$$

# Abstract Cost Analysis via the Writer Monad (cont.)

For effects, **calf** is "polarized" (à la CBPV/EEC/LNL).

**Cost Annotation**

To instrument a program with cost $c$, effect $\mathsf{charge}\langle \$c \rangle$.

**Key Idea**

Effects commute with computations: effects now are effects later.

$$\mathsf{charge}\langle \$c \rangle \,;\, (\lambda x.\ e) = \lambda x.\ (\mathsf{charge}\langle \$c \rangle \,;\, e)$$

$$\mathsf{charge}\langle \$c \rangle \,;\, (e_1, e_2) = ((\mathsf{charge}\langle \$c \rangle \,;\, e_1), (\mathsf{charge}\langle \$c \rangle \,;\, e_2))$$

# Abstract Cost Analysis via the Writer Monad (cont.)

**Semantics**

Category of cost algebras, $\mathbf{Alg}(T)$, where $T$ is the writer monad $\mathbb{C} \times (-)$, using adjunction $\mathsf{F} \dashv \mathsf{U} : \mathbf{Alg}(T) \to \mathcal{C}$.

Notation:

$$\cfrac{\cfrac{\mathsf{F}A \to X}{A \rightharpoonup X}}{A \to \mathsf{U}X}$$

$$\frac{\delta_{\$} : A \to \mathbb{C} \qquad \delta_{\circ} : A \to B}{\delta : A \rightharpoonup \mathsf{F}B}$$

# Coalgebraic Semantics of Data Structures

**Definition**

A *signature* is an endofunctor $\Sigma : \mathbf{Alg}(T) \to \mathbf{Alg}(T)$.

**Example**

The signature for queues

$$\Sigma X = (E \rightharpoonup X) \times (\mathsf{F}1 + (E \ltimes X))$$

power — copower

provides two operations

$$\begin{aligned} \text{enqueue} &: E \rightharpoonup X \\ \text{dequeue} &: \mathsf{F}1 + (E \ltimes X) \end{aligned}$$

where $X$ is the "state type".

# Coalgebraic Semantics of Data Structures (cont.)

**Definition**

A $\Sigma$-*coalgebra* $(D,\ \delta : D \to \Sigma D)$ is an implementation of $\Sigma$.

**Example**

With signature $\Sigma$ for queues as before:

- carrier $D = \mathsf{F}(\mathsf{list}(E))$, and
- transition map

$$\delta : D \to (E \rightharpoonup D) \times (\mathsf{F}1 + (E \ltimes D))$$

implements the operations.

# Coalgebraic Semantics of Data Structures (cont.)

**Definition (morphism of $\Sigma$-coalgebras)**

A morphism $(D, \delta) \to (S, \sigma)$ is a morphism $\Phi : D \to S$ that preserves the $\Sigma$-coalgebra structure:

$$\begin{array}{ccc} D & \xrightarrow{\delta} & \Sigma D \\ \downarrow{\scriptstyle \Phi} & & \downarrow{\scriptstyle \Sigma\Phi} \\ S & \xrightarrow{\sigma} & \Sigma S \end{array}$$

**Key Idea**

The specification $S$ *simulates* the data structure $D$.

# Basic Examples

# Example: Bulk Cost

**Specification (carrier F1)**

Charges $1 every cycle:

$$\sigma : 1 \rightharpoonup \mathsf{F}1$$
$$\sigma\ * = \mathsf{charge}\langle \$1 \rangle \ ; \mathsf{ret}(*)$$

# Example: Bulk Cost

**Specification (carrier $\mathsf{F}1$)**

Charges \$1 every cycle:

$$\sigma : 1 \rightharpoonup \mathsf{F}1$$
$$\sigma \, * = \mathsf{charge}\langle \$1 \rangle \, ; \mathsf{ret}(*)$$

**Implementation (carrier $\mathsf{F}(\mathsf{Fin}_8)$)**

Charges \$8 every 8 cycles:

$$\delta : \mathsf{Fin}_8 \rightharpoonup \mathsf{F}(\mathsf{Fin}_8)$$
$$\delta \; 7 = \mathsf{charge}\langle \$8 \rangle \, ; \mathsf{ret}(0)$$
$$\delta \; d = \mathsf{ret}(\mathsf{suc} \; d)$$

# Example: Bulk Cost (cont.)

Coalgebra morphism $\Phi : \mathsf{Fin}_8 \rightharpoonup \mathsf{F}1$ must satisfy:

$$\Phi \mathbin{;} \sigma = \delta \mathbin{;} \Sigma\Phi$$

# Example: Bulk Cost (cont.)

Coalgebra morphism $\Phi : \mathsf{Fin}_8 \rightharpoonup \mathsf{F1}$ must satisfy:

$$\Phi \,;\, \sigma = \delta \,;\, \Sigma\Phi$$

Since $\mathsf{U}(\mathsf{F1}) \cong \mathbb{C}$, equivalently $\Phi : \mathsf{Fin}_8 \to \mathbb{C}$:

$$\Phi(d) + \sigma_{\$} = \delta_{\$}(d) + \Phi(\delta_{\circ}(d))$$

# Example: Bulk Cost (cont.)

Coalgebra morphism $\Phi : \mathsf{Fin}_8 \rightharpoonup \mathsf{F1}$ must satisfy:

$$\Phi \mathbin{;} \sigma = \delta \mathbin{;} \Sigma\Phi$$

Since $\mathsf{U}(\mathsf{F1}) \cong \mathbb{C}$, equivalently $\Phi : \mathsf{Fin}_8 \to \mathbb{C}$:

$$\Phi(d) + \sigma_{\$} = \delta_{\$}(d) + \Phi(\delta_{\circ}(d))$$

When $\mathbb{C} = \mathbb{Z}$:

amortized cost

$$\sigma_{\$} = \delta_{\$}(d) + \Phi(\delta_{\circ}(d)) - \Phi(d)$$

real cost

new state $d'$

# Example: Bulk Cost (cont.)

Coalgebra morphism $\Phi : \mathsf{Fin}_8 \rightharpoonup \mathsf{F}1$ must satisfy:

$$\Phi \,;\, \sigma = \delta \,;\, \Sigma\Phi$$

Since $\mathsf{U}(\mathsf{F}1) \cong \mathbb{C}$, equivalently $\Phi : \mathsf{Fin}_8 \to \mathbb{C}$:

$$\Phi(d) + \sigma_{\$} = \delta_{\$}(d) + \Phi(\delta_{\circ}(d))$$

When $\mathbb{C} = \mathbb{Z}$:

$$\underbrace{\sigma_{\$}}_{\text{amortized cost}} = \underbrace{\delta_{\$}(d)}_{\text{real cost}} + \Phi(\underbrace{\delta_{\circ}(d)}_{\text{new state } d'}) - \Phi(d)$$

For example, $\Phi(d) = \$d$.

# Example: Batched Queue

$$(l_i, l_o)$$

$l_i$: inbox list; $l_o$: outbox list

- Enqueue to inbox;
- dequeue from outbox;
- move inbox to outbox when outbox empty.

$([4, 3], [1, 2])$

↓ enqueue 5

$([5, 4, 3], [1, 2])$

↓ dequeue 1

$([5, 4, 3], [2])$

↓ dequeue 2

$([5, 4, 3], [])$

↓ dequeue 3

$([], [4, 5])$

↓ enqueue 6

$([6], [4, 5])$

# Example: Batched Queue (cont.)

**Specification (carrier $\mathsf{F}(\mathsf{list}(E))$)**

Charges \$1 per enqueue, \$0 per dequeue:

$$\sigma : \mathsf{list}(E) \rightharpoonup \Sigma(\mathsf{F}(\mathsf{list}(E)))$$
$$\sigma\ .\mathsf{enqueue}\ l\ e = \mathsf{charge}\langle \$1 \rangle \mathbin{;} \mathsf{ret}(l \mathbin{+\!\!+} [e])$$
$$\sigma\ .\mathsf{dequeue} = \cdots$$

**Implementation (carrier $\mathsf{F}(\mathsf{list}(E)^2)$)**

Charges \$0 per enqueue, \$0 (usually) or $\$n$ (rarely) per dequeue.

# Example: Batched Queue (cont.)

**Specification (carrier $\mathsf{F}(\mathsf{list}(E))$)**

Charges \$1 per enqueue, \$0 per dequeue:

$$\sigma : \mathsf{list}(E) \rightharpoonup \Sigma(\mathsf{F}(\mathsf{list}(E)))$$
$$\sigma\ .\mathsf{enqueue}\ l\ e = \mathsf{charge}\langle \$1 \rangle\ ;\ \mathsf{ret}(l \mathbin{+\!\!+} [e])$$
$$\sigma\ .\mathsf{dequeue} = \cdots$$

**Implementation (carrier $\mathsf{F}(\mathsf{list}(E)^2)$)**

Charges \$0 per enqueue, \$0 (usually) or $\$n$ (rarely) per dequeue.

Let $\Phi(l_i, l_o) = \mathsf{charge}\langle \$(\mathsf{length}(l_i)) \rangle\ ;\ \mathsf{ret}(l_o \mathbin{+\!\!+} \mathsf{reverse}(l_i))$.

potential function (annotating $\mathsf{length}(l_i)$) — integrated behavior (annotating $l_o \mathbin{+\!\!+} \mathsf{reverse}(l_i)$)

# Amortizing Other Effects

## Non-Commutative Cost Models

Choosing $\mathbb{C} = \text{String}$, amortized string printing is buffering:

$$\Phi(\text{“hello”}) \mathbin{+\!\!+} \text{“world”} = \text{“hellowor”} \mathbin{+\!\!+} \Phi(\text{“ld”})$$

old buffer — spec print — impl print — new buffer

“Potential function” $\Phi$ is the inclusion, flushing the buffer.

# Amortizing Other Effects

## Non-Commutative Cost Models

Choosing $\mathbb{C} = \text{String}$, amortized string printing is buffering:

$$\Phi(\text{“hello”}) +\!\!+ \text{“world”} = \text{“hellowor”} +\!\!+ \Phi(\text{“ld”})$$

old buffer — spec print — impl print — new buffer

“Potential function” $\Phi$ is the inclusion, flushing the buffer.

## Randomized Amortized Analysis

Using monad $\mathcal{D}(\mathbb{C} \times (-))$, amortize randomness.

# Amortizing Other Effects

## Non-Commutative Cost Models

Choosing $\mathbb{C} = \text{String}$, amortized string printing is buffering:

$$\Phi(\text{"hello"}) + \text{"world"} = \text{"hellowor"} + \Phi(\text{"ld"})$$

old buffer — spec print — impl print — new buffer

"Potential function" $\Phi$ is the inclusion, flushing the buffer.

## Randomized Amortized Analysis

Using monad $\mathcal{D}(\mathbb{C} \times (-))$, amortize randomness.

## Expected Amortized Analysis

Using monad $\mathbb{C} \times \mathcal{D}(-)$, expected amortized analysis.

# Composition of Potential Functions

Potential functions are coalgebra morphisms, so they compose.

**Example**

For bulk cost amortization:

$$(D_{16}, \delta_{16}) \xrightarrow{\Phi'} (D_8, \delta_8) \xrightarrow{\Phi} (S, \sigma)$$

# Composition of Potential Functions (cont.)

To compose data structures with different signatures:

$$\int^{\Sigma} \mathbf{Coalg}(\Sigma)$$

**Example**

Amortized queues implemented via a pair of amortized stacks.

$$\mathsf{asQueue} : \Sigma_{\mathsf{Stacks}} \to \Sigma_{\mathsf{Queue}}$$

$$\Phi : \mathsf{asQueue}(D_{\mathsf{stacks}}, \delta_{\mathsf{stacks}}) \to (S_{\mathsf{queue}}, \sigma_{\mathsf{queue}})$$

# Generalizations

# Lax Amortized Analysis

Sometimes, the amortized cost is an overestimate:

$$\textbf{amortized cost} \geq \textbf{real cost} + \Phi(d') - \Phi(d)$$

In some cases, the change in potential is less than the spec.

**Key Idea**

Upgrade to bicategories: programs are ordered by inequality.

# Lax Amortized Analysis (cont.)

**Definition (colax morphism of Σ-coalgebras)**

A *colax morphism* $(D, \delta) \to (S, \sigma)$ is a morphism $\Phi : D \to S$ that "colaxly" preserves the Σ-coalgebra structure:

$$\begin{array}{ccc} D & \xrightarrow{\ \delta\ } & \Sigma D \\ {\scriptstyle \Phi}\downarrow & \overset{\varphi}{\Swarrow} & \downarrow{\scriptstyle \Sigma\Phi} \\ S & \xrightarrow[\ \sigma\ ]{} & \Sigma S \end{array}$$

Here, 2-cell $\varphi : (\Phi \mathbin{;} \sigma) \Leftarrow (\delta \mathbin{;} \Sigma\Phi)$ is a proof of inequality.

Choose $\mathcal{C} = \mathbf{Poset}$, letting all types but $\mathbb{C} = \omega$ be discrete.

$$\Phi(d) + \sigma_{\$} \geq \delta_{\$}(d) + \Phi(\delta_{\circ}(d))$$

# Lax Amortized Analysis (cont.)

**Remark**

A 2-cell $\Phi \leq \Phi'$ justifies that $\Phi$ is a tighter analysis than $\Phi'$.

**Example**

For bulk cost, both

$$\Phi(d) = \$d$$
$$\Phi'(d) = \$(d + 1)$$

are coalgebra morphisms. Now, observe that $\Phi \leq \Phi'$.

# Splitting Potential

**Example**

Some operations split data structures into parts:

$$\Sigma X = X \otimes X$$

Informally, for multiple outputs:

total output potential

$$\sigma_{\$} \geq \delta_{\$}(d) + \sum_i \Phi(\delta_{\circ}(d)_i) - \Phi(d)$$

Made formal when $T$ is commutative, using map

$$\mathsf{F}A \otimes \mathsf{F}B \xrightarrow{\sim} \mathsf{F}(A \times B)$$

to add potential.

# Splitting Potential (cont.)

## Example

Let $\Phi : \mathsf{F}A \to \mathsf{F}1$:

$$\begin{array}{ccccc} \mathsf{F}A & \xrightarrow{\delta} & \mathsf{F}A \otimes \mathsf{F}A & & \\ \downarrow{\scriptstyle \Phi} & \geq & \downarrow{\scriptstyle \Phi \otimes \Phi} & & \\ \mathsf{F}1 & \xrightarrow{\sigma} & \mathsf{F}1 \otimes \mathsf{F}1 & \overset{+}{=\!=\!=} & \mathsf{F}1 \end{array}$$

In other words:

$$\Phi(d) + \sigma_{\$} \geq \delta_{\$}(d) + \sum_{i \in \{1,2\}} \Phi(\delta_{\circ}(d)_i)$$

# Combining Potential

Some data structures, e.g. queues, support an append operation:

$$\frac{X \otimes X \to X}{X \to (X \multimap X)}$$

But, $\Sigma X = X \multimap X$ is not functorial!

Instead, use a profunctor $\Sigma : \mathbf{Alg}(T)^{\mathrm{op}} \times \mathbf{Alg}(T) \to \mathbf{Set}$. Here:

$$\Sigma(X^-, X^+) = (X^- \otimes X^-) \multimap X^+$$

As desired, this gives us:

$$\sum_{i \in \{1,2\}} \Phi(d_i) + \sigma_\$ \geq \delta_\$(d) + \Phi(\delta_\circ(d))$$

total input potential

# Conclusion

# Conclusion

## Foundation

In a category of cost algebras, a **coalgebra morphism** is a generalized **potential function** of amortized analysis.

- Integrates cost and behavior;
- Provides a theory of composition for amortized analyses;
- Elegantly supports amortization of arbitrary effects;
- Simplifies formalization.

## Extensions

- Inexact amortized analysis expressed via bicategories;
- Splitting potential expressed via monoidal products;
- Combining potential expressed via profunctors.