Compositional Verification of Cost and Behavior in Type Theory

Robert Harper
j.w.w. Harrison Grodin and Runming Li

IFIP WG2.8 March 2026

Supported by AFOSR MURI, NSF, and Jane Street Capital.

Motivation

Composition is the only game in town for building programs:

> let X : *interface* **be** *impl* **in** *client*

In lieu of verification X is held wholly abstract to permit change of implementation and help isolate errors.

Specification of cost and behavior is informal, if present at all, and not verified.

(Types do go a long way, though!)

Queues, Typically

A "typical" interface for queues in familiar settings:

interface PreQueue

X : **Type**

emp : X *(* no elements, O(1) *)*

enq : $\mathbb{N} \to X \to X$ *(* add to back, O(n) *)*

deq : $X \to \mathbb{N} \times X$ *(* take from front, O(1) *)*

Comments have no force ... and are often obsolete or omitted.

Bounds are worst-case? Stated as a function of what?

Compositional Verification?

For verification hiding information is a non-starter:

- Representation type cannot be hidden because ...
- ... specification of behavior requires it, and ...
- ... cost cannot be determined without behavior!

Lessons of history suggest, nothing short of a full implementation will suffice.

(cf experience with algebraic specifications)

Queues, Fully Specified

Eg, behavior of queues might be specified in terms of lists:

interface $\textsc{Queue}$

X : **Type** $= \textsc{List}\ \mathbb{N}$

empty : $X = []$

enqueue : $\mathbb{N} \to X \to X = \lambda n\ q \to q +\!\!+ [n]$

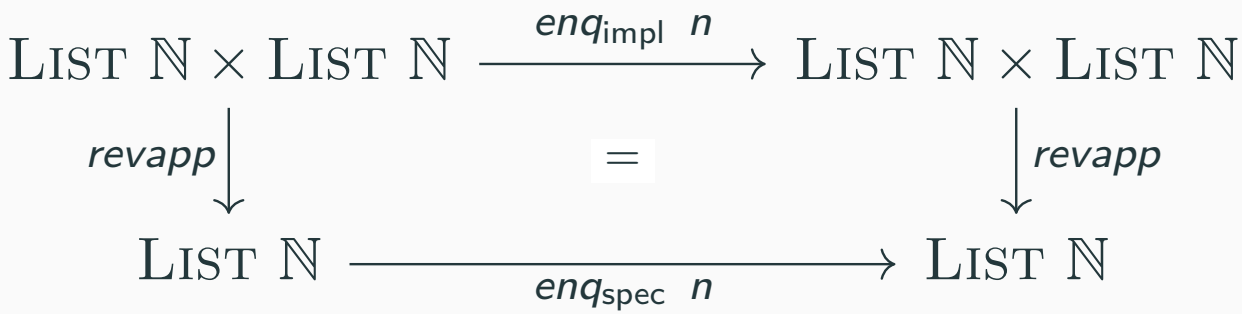
dequeue : $X \to \mathbb{N} \times X = \lambda q \to \ldots$

Client has a full understanding of the "abstraction" ...but it admits *only one* implementation!

And what about cost? Is linear time for enqueue now demanded?

Abstraction Functions

Hoare proposed abstraction functions to connect efficient to reference implementation



The abstraction function *revapp* sends (b, f) to $b \mathbin{@} rev(f)$.

Commutativity expresses correctness of the two-list implementation of enqueue relative to the specification.

$$revapp \circ (enq_{impl}\ n) = (enq_{spec}\ n) \circ revapp$$

(Each specification admits a cone of implementations over it.)

Abstraction Functions, with Cost

Hoare's method scales to account for cost, a la Calf and Decalf:

$$\begin{array}{ccc} \textsc{List}\ \mathbb{N} \times \textsc{List}\ \mathbb{N} & \xrightarrow{enq_{\text{impl}}\ n} & \mathbf{U}(\mathbf{F}(\textsc{List}\ \mathbb{N} \times \textsc{List}\ \mathbb{N})) \\ \Big\downarrow{\scriptstyle revapp} & \geq & \Big\downarrow{\scriptstyle \mathbf{U}(\mathbf{F}(revapp))} \\ \textsc{List}\ \mathbb{N} & \xrightarrow[enq_{\text{spec}}\ n]{} & \mathbf{U}(\mathbf{F}(\textsc{List}\ N)) \end{array}$$

Pre-order expresses upper bound on cost, but exact behavior.

$$\mathbf{U}(\mathbf{F}(revapp)) \circ (enq_{impl}\ n) \leq (enq_{spec}\ n) \circ revapp$$

Synthetic Formulation

Every type X is (tacitly) given by an abstraction function.

- No loss of generality: choose the identity map.
- Full range of dependent type theory constructs is available ...
- ... augmented by glueing for examples such as queues.

Synthetic formulation uses two modalities induced by a phase:

- Abstract, $\bigcirc X$: isolates specification.
- Concrete, $\bullet X$: isolates implementation.

(cf behavioral phase to suppress cost in Calf and Decalf)

Queues, Reference Interface w/Cost

Behavioral specification is augmented with (implausible!) costs:

interface $\textsc{Queue}$

$X : \mathbf{Type} \mathrel{\hat{=}} \textsc{List}\ \mathbb{N}$

$\mathsf{emp} : X \mathrel{\hat{=}} \mathbf{charge}\langle 1\rangle; []$

$\mathsf{enq} : \mathbb{N} \to X \to X \mathrel{\hat{=}} \lambda n\ q \to \mathbf{charge}\langle 1\rangle; q +\!\!+ [n]$

$\mathsf{deq} : X \to \mathbb{N} \times X \mathrel{\hat{=}} \lambda q \to \mathbf{charge}\langle |q|\rangle; \ldots$

(Here $M \mathrel{\hat{=}} N$ means $\bigcirc(M = N)$, valid only in abstract phase.)

Cost is not empirical, it is rather specified by the code!

Synthetic Formulation: Glueing

The value type of queues is given by glueing:

$$\mathsf{Glue}^{\mathcal{V}}(\bullet(\textsc{List}\ \mathbb{N} \times \textsc{List}\ \mathbb{N}), \bigcirc(\textsc{List}\ \mathbb{N}), \bullet(\eta_{\circ} \circ \mathit{revapp}))$$

Modalities isolate concrete and abstract aspects of types, even for primitive types.

Operations are implemented by a sealing effect a la modules:

$$\mathit{seal}(\mathit{enq}_{\mathit{impl}}, \mathit{enq}_{\mathit{spec}}, \mathit{prf})$$

where prf is evidence of the inequality to match interface bound.

Queue Implementation

Formulated in an ML-like notation:

$$
\begin{aligned}
&\mathit{BatchedQ} : \textsc{Queue} \\
&\mathit{BatchedQ}.X := \textsc{List}\ \mathbb{N} \times \textsc{List}\ \mathbb{N} \\
&\quad \textbf{with}\ \alpha\ (\mathit{back}, \mathit{front}) := \mathit{front} +\!\!+ \mathit{rev}(\mathit{back}) \\
&\mathit{BatchedQ}.\mathsf{emp} := ([], []) \\
&\quad \textbf{with}\ \_ : \alpha\ (\mathit{BatchedQ}.\mathsf{emp}) \mathrel{\mathring{=}} \textsc{Queue}.\mathsf{emp} \\
&\mathit{BatchedQ}.\mathsf{enq}\ n\ (\mathit{back}, \mathit{front}) := \textbf{charge}\langle 1\rangle; (n :: \mathit{back}, \mathit{front}) \\
&\quad \textbf{with}\ \_ : \alpha\ (\mathit{BatchedQ}.\mathsf{enq}\ n\ q) \mathrel{\mathring{=}} \textsc{Queue}.\mathsf{enq}\ n\ (\alpha\ q) \\
&\mathit{BatchedQ}.\mathsf{deq}\ (\mathit{back}, n :: \mathit{front}) := \\
&\quad \textbf{charge}\langle 1\rangle; (n, (\mathit{back}, \mathit{front})) \\
&\mathit{BatchedQ}.\mathsf{deq}\ (\mathit{back}, []) := \\
&\quad \textbf{charge}\langle |\mathit{back}|\rangle; \mathit{BatchedQ}.\mathsf{deq}\ ([], \mathit{rev}(\mathit{back})) \\
&\quad \textbf{with}\ \_ : (\mathbb{N} \times \alpha)\ (\mathit{BatchedQ}.\mathsf{deq}\ q) \mathrel{\mathring{=}} \textsc{Queue}.\mathsf{deq}\ (\alpha\ q)
\end{aligned}
$$

Amortization?

Weak bound for dequeue is pessimistic, but cannot be avoided.

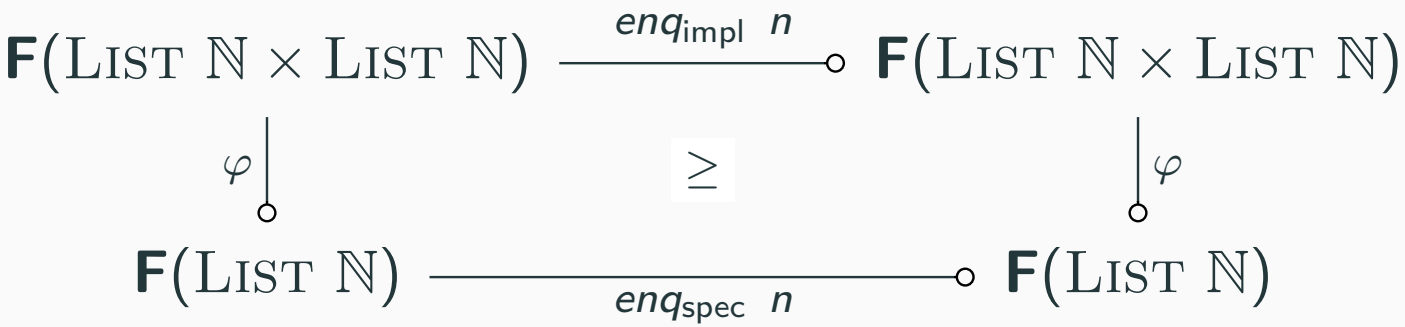
- In a structural setting queues may be used persistently, leading to replicated work.
- Rules out amortization, which relies on queues being ephemeral, allowing for eventual work.

To ensure ephemeral usage, extend Calf with a linear context at the computation level (a la EEC, LNL).

- Values remain persistent and otherwise unchanged.
- Computations may restrict usage via linear entailment.

Potential Functions as Types?

Potential functions enrich abstraction functions with cost:



where φ is the linear function

$$\lambda(\mathbf{ret}(b, f)) \multimap \mathbf{charge}\langle|b|\rangle; \mathbf{ret}(\mathit{revapp}(b, f)).$$

Linearity is used in both dimensions:

- Vertically, cost is stored potential.
- Horizontally, cost is incurred charge.

Lax commutativity expresses amortization equations relating cost and potential.

Amortized Specification Interface

Interface for ephemeral queues:

interface $\textsc{EQueue}$

$A : \mathbf{CType} \doteq \mathbf{F}(\textsc{List}\ \mathbb{N})$

$\mathsf{emp} : \mathbf{U}(A) \doteq \mathbf{charge}\langle 1\rangle; \mathbf{ret}([])$

$\mathsf{enq} : \mathbb{N} \times A \multimap A \doteq \lambda(n, \mathbf{ret}(q)) \multimap \mathbf{charge}\langle 1\rangle; q +\!\!+ [n]$

$\mathsf{deq} : A \multimap \mathbb{N} \times A \doteq \lambda q \multimap \mathbf{charge}\langle 1\rangle; \ldots$

Unit cost for dequeue reflects use of linearity to express amortized bounds.

Potential Functions, Synthetically

Extend computation types with $\bigcirc A$, $\bullet A$, and glueing to specify amortized interfaces.

$$\mathsf{Glue}^{\mathcal{C}}(\bullet \mathbf{F}(\textsc{List}\ \mathbb{N} \times \textsc{List}\ \mathbb{N}), \bigcirc \mathbf{F}(\textsc{List}\ \mathbb{N}), \bullet(\eta_{\circ} \circ \varphi))$$

The operations themselves remain (largely) unchanged, only the analysis differs!

Amortized Queues

Implementation of amortized queues:

$$
\begin{aligned}
&\mathit{BatchedQ} : \textsc{EQueue} \\
&\mathit{BatchedQ}.X := \mathbf{F}(\textsc{List}\ \mathbb{N} \times \textsc{List}\ \mathbb{N}) \\
&\quad \textbf{with}\ \varphi(\textbf{ret}(b, f)) \ := \textbf{charge}\langle |b| \rangle; \textbf{ret}(\mathit{revapp}(b, f)) \\
&\mathit{BatchedQ}.\mathsf{emp} := \textbf{ret}([], []) \\
&\quad \textbf{with}\ \_ : \varphi\ (\mathit{BatchedQ}.\mathsf{emp}) \mathrel{\dot{=}} \textsc{EQueue}.\mathsf{emp} \\
&\mathit{BatchedQ}.\mathsf{enq}\ n\ (\mathit{back}, \mathit{front}) := \textbf{charge}\langle 1 \rangle; (n :: \mathit{back}, \mathit{front}) \\
&\quad \textbf{with}\ \_ : \varphi\ (\mathit{BatchedQ}.\mathsf{enq}\ n\ q) \mathrel{\dot{=}} \textsc{EQueue}.\mathsf{enq}\ n\ (\varphi\ q) \\
&\vdots
\end{aligned}
$$

Ongoing Work

Variant of "big O" notation for $f, g : X \to \mathbb{N}$:

- Define $f \preceq g$ iff $f \leq c + k\,g$ for some $c \geq 0$ and $k > 0$.
- Smoothly accounts for multi-variate functions, other domains.

Define a modality, $\mathcal{O}(A)$, on computation types.

- Add **scale**$\langle k \rangle; e$ to **charge**$\langle c \rangle; e$ to express inequation.
- Rescaling is permitted (and sensible) only in abstract phase!

With this modality the informal interface becomes formal!

Ongoing Work

Interpret Hoffmann's AARA in Calf w/amortization.

- Use computation glueing to add potential to incurred costs.
- RAML as a tactic for automatic cost inference in Calf.

Verification of meta-theory.

- Calf itself: CPP 26.

Library of case studies:

- RBT's, Parallel Scan, Splay Tree Merge (PoPL 26).

Probabilistic bounds?

Summary

Compositional verification of cost and behavior!

But why only now?

- Univalence equates equivalent representations.
- Phases distinguish abstraction from implementation (and behavior from cost).
- Inequality of computations to express lax bounds.
- Linearity provides for amortization.

Thanks!

Calf web page for links to papers, case studies, mechanized meta-theory:

`https://calfproject.github.io/calf/`

(See the papers linked there for citations and comparisons to related work.)