Denotational Recurrence Extraction for Amortized Analysis

Joe Cutler  Dan Licata  Norman Danner

Wesleyan University
Informal Recurrence Extraction

mergeSort : int list → int list
mergeSort [] = []
mergeSort xs = let (l, r) = split xs in
merge (mergeSort l, mergeSort r)

\[ T_{\text{mergeSort}}(n) = T_{\text{split}}(n) + T_{\text{merge}}(n) + 2T_{\text{mergeSort}}\left(\frac{n}{2}\right) \]

How do we make this informal process formal?
Formally Extracting Recurrences

Source Language

$$\Gamma \vdash M : A$$

Recurrence Language

$$\langle \Gamma \rangle \vdash \| M \| : \mathbb{C} \times \langle A \rangle$$

Functions get translated to recurrences in the traditional sense

$$\langle A \to B \rangle = \langle A \rangle \to \mathbb{C} \times \langle B \rangle$$

$$\langle A \times B \rangle = \langle A \rangle \times \langle B \rangle$$

$$\ldots$$

$$\| \lambda x. M \| = (0, \lambda x. \| M \|)$$

$$\|(M, N)\| = (\pi_1 \| M \| + \pi_1 \| N \|, (\pi_2 \| M \|, \pi_2 \| N \|))$$

Monadic translation to recurrence language into writer monad

Cost to evaluate M

Result of running M (size or use-cost)
Proving Extraction Correctness

Cost-Indexed Big-Step Operational Semantics

$$M \downarrow^n v$$

Bounding Theorem

For $M : A$ if $M \downarrow^n v$, then $n \leq \pi_1 \|M\|$ and $v \sqsubseteq^A_{\text{val}} \pi_2 \|M\|

Size-Abstraction Semantics

$$\sum \mathcal{C} \xrightarrow{[]} \text{Poset}$$
Prior Work & Limitations

This technique works for:

<table>
<thead>
<tr>
<th>STLC</th>
<th>Inductive Types</th>
<th>PCF</th>
<th>Let-Polymorphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>[PLPV ’13]</td>
<td>[ICFP ’15]</td>
<td>[POPL ’20]</td>
<td>[arxiv:2002.07262]</td>
</tr>
</tbody>
</table>

but it can’t handle...

Amortized Analysis
This Work:
Amortized Analysis by
Formal Recurrence Extraction

Examples in the Paper:

Binary Counter

```
1 0 1 1
```

Splay Trees

```
```


Binary Counter

type bit = 0 | 1

inc : bit list → bit list
inc [] = [1]
icc (0 :: bs) = 1 :: bs
inc (1 :: bs) = 0 :: (inc bs)

set : nat → bit list
set 0 = []
set (S n) = inc (set n)

\[ T_{inc}(0) = 1 \]
\[ T_{inc}(n) = \max(1, 1 + T_{inc}(n - 1)) \]

\[ T_{set}(0) = 0 \]
\[ T_{set}(n) = T_{inc}(\log n) + T_{set}(n - 1) \]

\[ T_{inc}(n) \in O(n) \]
\[ T_{set}(n) \in O(n \log_2 n) \]

But we can do better!

\[ T_{set}(n) \in O(n) \]
Binary Counter, Formally

\[
\begin{align*}
\text{inc} : \text{bit list} & \rightarrow \text{bit list} \\
\text{inc} [\ ] & = [1] \\
\text{inc} (0 :: \text{bs}) & = 1 :: \text{bs} \\
\text{inc} (1 :: \text{bs}) & = 0 :: (\text{inc} \text{bs})
\end{align*}
\]

\[
\begin{align*}
\|	ext{inc}\|_c : \text{bit list} & \rightarrow \mathbb{C} \\
\|	ext{inc}\|_c [\ ] & = 1 \\
\|	ext{inc}\|_c (0 :: \text{bs}) & = 1 \\
\|	ext{inc}\|_c (1 :: \text{bs}) & = 1 + (\|	ext{inc}\|_c \text{bs})
\end{align*}
\]

\[
\begin{align*}
\llbracket \text{bit list} \rrbracket & = \mathbb{N} \\
\llbracket b :: \text{bs} \rrbracket & = 1 + \llbracket \text{bs} \rrbracket
\end{align*}
\]

\[
\begin{align*}
\llbracket \|	ext{inc}\|_c \rrbracket (n) & \in O(n) \\
\llbracket \|	ext{set}\|_c \rrbracket (n) & \in O(n \log_2 n)
\end{align*}
\]
# Amortized Analysis

<table>
<thead>
<tr>
<th>Cons Ops</th>
<th>Credits Created</th>
<th>Credits Spent</th>
<th>Amortized Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
New Source Language $\lambda^A$

**Credits in Context**

$\Gamma \vdash_c M : A$

(Affine type system!)

**Credit Modality**

$!_c A$

(Graded modal types!)

**Attaching Credits**

$\Gamma \vdash_a M : A$

$\Gamma \vdash_{a+c} \text{save}_c(M) : !_c A$

**Transferring Credits**

$\Gamma \vdash_a M : !_c A$

$\Gamma, x : A \vdash_{b+c} N : C$

$\Gamma \vdash_{a+b} \text{transfer} !_c x = M \text{ to } N : C$

**Creating Credits**

$\Gamma \vdash_{a+c} M : A$

$\Gamma \vdash_a \text{create}_c(M) : A$

**Spending Credits**

$\Gamma \vdash_a M : A$

$\Gamma \vdash_{a+c} \text{spend}_c(M) : A$
Binary Counter in $\lambda^A$

type bit = unit $\oplus$ !1unit

inc : bit list $\rightarrow$ bit list
inc [] = [create₁(inr(save₁()))]
inc ((inl _) :: bs) = (create₁(inr(save₁())) :: bs)
inc ((inr x) :: bs) = transfer _ = x to spend₁ ((inl ()) :: (inc bs))
Extracting Amortized Recurrences

Creating Credits Incurs a Cost
\[ \| \text{create}_a(M) \| = (a + \pi_1 \| M \|, \pi_2 \| M \|) \]

Spending Credits Frees up Cost
\[ \| \text{spend}_a(M) \| = (-a + \pi_1 \| M \|, \pi_2 \| M \|) \]

Extraction Erases the Modality
\[ \langle !c.A \rangle = \langle A \rangle \]
\[ \| \text{save}_c(M) \| = \| M \| \]
\[ \| \text{transfer} \ x = M \text{ to } N \| = \text{let } (c, x) = \| M \| \text{ in } (c + \pi_1 \| N \|, \pi_2 \| N \|) \]
Binary Counter... Again

type bit = unit \oplus \mathsf{!}_1 \text{unit}

\lambda A . \begin{align*}
\text{inc} : \text{bit list} &\rightarrow \text{bit list} \\
\text{inc} [] &= [\text{create}_1 (\text{inr} (\text{save}_1 ()))] \\
\text{inc} ((\text{inl} \_ :: \text{bs}) &= (\text{create}_1 (\text{inr} (\text{save}_1 ()))) :: \text{bs} \\
\text{inc} ((\text{inr} x :: \text{bs}) &= \text{transfer} \_ = x \text{ to spend}_1 ((\text{inl} ()) :: (\text{inc bs})))
\end{align*}

\lambda C . \begin{align*}
\| \text{inc} \|_c &: (\text{unit} + \text{unit}) \text{ list} \rightarrow (\text{unit} + \text{unit}) \text{ list} \\
\| \text{inc} \|_c [] &= 2 \\
\| \text{inc} \|_c (\text{inl} \_ :: \text{bs}) &= 2 \\
\| \text{inc} \|_c (\text{inr} \_ :: \text{bs}) &= \| \text{inc} \|_c \text{ bs}
\end{align*}

\text{posets} \quad \| \text{inc} \|_c (n) = 2 \quad \Rightarrow \quad \| \text{set} \|_c (n) = 2n \in O(n)
Proving Extraction Correctness

Amortized Cost Indexed
Big-Step
Operational Semantics

\[ M \Downarrow^n v \]

Bounding Theorem
For \( M : A \) if \( M \Downarrow^n v \),
then \( n \leq \pi_1 \| M \| \)
and \( v \sqsubseteq^A_{\text{val}} \pi_2 \| M \| \)

Key Corollary
For closed terms typed in a context with no credits,
recurrence-predicted amortized cost
is a bound on real evaluation cost
Thank you!

$!^{cA}$  
- Affine type system & modality for tracking credits

$\|M\|$  
- Automatic recurrence extraction translation

$n \leq \pi_1 \|M\|$  
- Correctness proof relative to operational semantics by logical relations

- Expressive enough to handle non-trivial analyses like splay trees  
  (not handled by existing techniques)