Multivariate Amortized Resource Analysis

Jan Hoffmann, Klaus Aehlig, and Martin Hofmann
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?

clock cycles, heap space, power, ...
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?

- Asymptotic bounds are often insufficient (embedded systems & hard real-time systems)

clock cycles, heap space, power, ...
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?

- Asymptotic bounds are often insufficient (embedded systems & hard real-time systems)
- Concrete constant factors for specific hardware

clock cycles, heap space, power, ...
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?

- Asymptotic bounds are often insufficient (embedded systems & hard real-time systems)
- Concrete constant factors for specific hardware
- Manual analyses are tedious & error-prone

clock cycles, heap space, power, ...
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?

- Asymptotic bounds are often insufficient (embedded systems & hard real-time systems)

- Concrete constant factors for specific hardware

- Manual analyses are tedious & error-prone

- High-level programming languages may make analyses more complex
Static Resource Analysis

What is the resource consumption of a program as a function of its input sizes?

- Asymptotic bounds are often insufficient (embedded systems & hard real-time systems)
- Concrete constant factors for specific hardware
- Manual analyses are tedious & error-prone
- High-level programming languages may make analyses more complex

⇒ we would like automatic methods for static resource analysis
<table>
<thead>
<tr>
<th>Method</th>
<th>Strengths</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariant Generation</td>
<td>C++ code</td>
<td>Gulwani et al. (POPL’09)</td>
</tr>
<tr>
<td></td>
<td>Numeric programs</td>
<td>Gulavani et al. (CAV’08)</td>
</tr>
<tr>
<td>Recurrence Relations</td>
<td>Java bytecode</td>
<td>Albert et al. (VMCAI’11)</td>
</tr>
<tr>
<td></td>
<td>Flexibility</td>
<td>Albert et al. (ESOP’07)</td>
</tr>
<tr>
<td>WCET Analysis</td>
<td>machine code</td>
<td>Wilhelm et al. (TECS’08)</td>
</tr>
<tr>
<td></td>
<td>low-level features</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(caches, pipelines, ...)</td>
<td></td>
</tr>
<tr>
<td>Amortized Analysis</td>
<td>Recursion</td>
<td>Jost et al. (POPL’10)</td>
</tr>
<tr>
<td></td>
<td>Nested data structures</td>
<td>Hofmann et al. (POPL’03)</td>
</tr>
</tbody>
</table>

**Automatic Methods for Static Resource Analysis**
<table>
<thead>
<tr>
<th>Method</th>
<th>Strengths</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invariant Generation</td>
<td>C++ code</td>
<td>Gulwani et al. (POPL’09)</td>
</tr>
<tr>
<td></td>
<td>Numeric programs</td>
<td>Gulavani et al. (CAV’08)</td>
</tr>
<tr>
<td>Recurrence Relations</td>
<td>Java bytecode</td>
<td>Albert et al. (VMCAI’11)</td>
</tr>
<tr>
<td></td>
<td>Flexibility</td>
<td>Albert et al. (ESOP’07)</td>
</tr>
<tr>
<td>WCET Analysis</td>
<td>machine code</td>
<td>Wilhelm et al. (TECS’08)</td>
</tr>
<tr>
<td></td>
<td>low-level features</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(caches, pipelines, ...)</td>
<td></td>
</tr>
<tr>
<td>Amortized Analysis</td>
<td>Recursion</td>
<td>Jost et al. (POPL’10)</td>
</tr>
<tr>
<td></td>
<td>Nested data structures</td>
<td>Hofmann et al. (POPL’03)</td>
</tr>
</tbody>
</table>

Automatic Methods for Static Resource Analysis | Overview
Amortized Resource Analysis

- Automatic type-based analysis: No annotations required
- Run-time behavior of programs is unaffected
- Generic in the resource: heap space, stack space, clock cycles, ...

- Today: **Polynomial bounds** for first-order functional programs
- Also: linear bounds for:    - object-oriented programs (ESOP’06, CSL’09)
  - higher-order functional programs (POPL’10)

- Efficient type inference based on linear programming
Bird’s Eye View

Program Source

- Amortized Analysis
- Resource Bounds

- apply to

- Compiler

- Machine Code

Type-Based Amortized Analysis

Dienstag, 1. Februar 2011
Bird’s Eye View

Program Source

Amortized Analysis

Resource Bounds

Compiler

Hardware

Machine Code

Type-Based Amortized Analysis

Dienstag, 1. Februar 2011
Bird's Eye View

Program Source

Amortized Analysis

Resource Bounds

Machine-Code Snippets

Compiler

Hardware

apply to

Machine Code

Dienstag, 1. Februar 2011
Bird’s Eye View

Program Source

Amortized Analysis

WCET Tool

Machine-Code Snippets

Compiler

Resource Bounds

Hardware

apply to

Machine Code

Type-Based Amortized Analysis

Dienstag, 1. Februar 2011
Bird’s Eye View

Type-Based Amortized Analysis
Type-Based Amortized Analysis

- Assign potential functions to data structures
  \( \Phi(state) \geq 0 \)
  - States are mapped to non-negative numbers

- Potential pays the resource consumption and the potential at the following program point
  \( \Phi(before) \geq \Phi(after) + cost \)

\( \downarrow \) telescoping

\( \Phi(initial\ state) \geq \sum cost \)

- Initial potential is an upper bound
Type-Based Amortized Analysis

- Assign potential functions to data structures
  - States are mapped to non-negative numbers

- Potential pays the resource consumption and the potential at the following program point

- Initial potential is an upper bound

\[ \Phi(state) \geq 0 \]
\[ \Phi(before) \geq \Phi(after) + cost \]
\[ \Phi(initial\ state) \geq \sum cost \]

Type Systems for automatic analysis

- Fix a format of potential functions
- Develop type rules that manipulate potential functions
Linear Potential Functions (POPL’03)

<table>
<thead>
<tr>
<th>List types:</th>
<th>$L^q(A)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential functions:</td>
<td>$\Phi(\ell:L^q(int)) = q \cdot</td>
</tr>
</tbody>
</table>

| Function types:   | $A^{q/q'} \to B$  |
Linear Potential Functions (POPL’03)

List types: \( L^q(A) \)

Potential functions: \( \Phi(\ell:L^q(int)) = q \cdot |\ell| \)

Non-negative rational number

Function types: \( A^{q/q'} \rightarrow B \)
Linear Potential Functions (POPL’03)

List types: \( L^q(A) \)

Potential functions: \( \Phi(\ell : L^q(int)) = q \cdot |\ell| \)

Function types: \( A^{q/q'} \rightarrow B \)

- Non-negative rational number
- Constant input potential
- Linear input potential
Linear Potential Functions (POPL’03)

Function types:
\[ A^{q/q'} \rightarrow B \]

List types:
\[ L^q(A) \]

Potential functions:
\[ \Phi(\ell:L^q(int)) = q \cdot |\ell| \]

- Non-negative rational number
- Constant input potential
- Constant output potential
- Linear input potential
- Linear output potential
Example: Evaluation Steps of Append

\[
\begin{align*}
append &\colon (L(int), L(int)) \longrightarrow L(int) \\
append(xs, ys) &\equiv \text{match } xs \text{ with } \text{nil} \rightarrow ys \\
& \quad \mid (x::xs') \rightarrow x::append(xs', ys)
\end{align*}
\]
Example: Evaluation Steps of Append

```markdown
append: (L(int),L(int)) ----> L(int)

append(xs,ys) = match xs with  
  | nil -> ys  
  | (x::xs') -> x::append(xs',ys)
```

Number of evaluation steps in the worst case: $8|xs| + 3$
Example: Evaluation Steps of Append

append: \((L(int), L(int)) \rightarrow L(int)\)

\[
\text{append}(xs, ys) = \begin{cases} 
\text{match } xs \text{ with } & | \text{nil } \rightarrow ys \\
& | (x::xs') \rightarrow x::\text{append}(xs', ys)
\end{cases}
\]

Number of evaluation steps in the worst case: \(8|xs| + 3\)

Possible typings (type rules justify both):

\[
\text{append: } (L^8(int), L^0(int)) \rightarrow L^0(int)
\]

\[
\text{append: } (L^{16}(int), L^8(int)) \rightarrow L^8(int)
\]
Example: Evaluation Steps of Append

```
append: (L(int),L(int)) ----> L(int)

append(xs,ys) = match xs with | nil -> ys  
  | (x::xs’) -> x::append(xs’,ys)
```

Number of evaluation steps in the worst case: $8|xs| + 3$

Possible typings (type rules justify both):

```
append: (L (int),L (int)) ----> L (int)
append: (L (int),L (int)) ----> L (int)
```

To type inner occurrence in $append(append(a,b),c)$
Benefits of Linear Amortized Analysis

• Often conceptually simpler than maintaining a global counter

• No explicit reasoning about sizes of data

• Naturally deals with nested data structures

• Recognizes if data is scattered over different locations (partitions in quick sort)

• Highly compositional

• Efficient type inference based on linear programming
How to Extend to Polynomial Bounds?

First idea:

Potential functions are polynomials with non-negative coefficients

But:

1. We want to express bounds that contain \textit{binomial coefficients}

2. We want to express \textit{multivariate bounds} such as $n^2 \cdot m$

3. We want to incorporate the \textit{individual sizes} of inner data structures
1. Binomial Coefficients

• Many tight bounds contain binomial coefficients

\[ \sum_{i=1}^{n-1} i = \binom{n}{2} \]

• Typical example: insertion sort

• Such bounds cannot be expressed using the usual basis

• Non-negative linear combinations of binomials are the largest class \( C \) of polynomials such that

\[ p(n) \geq 0 \quad \text{for all} \ p \in C \]

if \( p \in C \) then \( \Delta p \in C \) where \( \Delta p(n) = p(n + 1) - p(n) \in C \)
2. Multivariate Bounds

- Multivariate bounds are essential for obtaining good results in practice

- Example: sort inner lists with insertion sort

  \[
  \text{sortAll: } L(L(\text{int})) \rightarrow L(L(\text{int}))
  \]

- Resource bound: \(O(nm^2)\)

  where \(n\) is the length of the outer list
  \(m\) is the maximal length of the inner lists
2. Multivariate Bounds

• Multivariate bounds are essential for obtaining good results in practice

• Example: sort inner lists with insertion sort

\[
\text{sortAll: L(L(int))} \rightarrow \text{L(L(int))}
\]

• Resource bound: \( O(nm^2) \)

where \( n \) is the length of the outer list

\( m \) is the maximal length of the inner lists

Over-approximating sizes of inner data leads to loose bounds when composing functions.
3. Incorporate Individual Sizes

Example: sortAll in a composed function

1. Function split: group values by their keys

2. Function sortAll: sort each list of values

\[ O(nm^2) \]
3. Incorporate Individual Sizes

**Example:** `sortAll` in a composed function

1. **Function `split`:** group values by their keys

2. **Function `sortAll`:** sort each list of values

Run time: $O(k^2)$ where $k$ is the length of the initial list
3. Incorporate Individual Sizes

Example: `sortAll` in a composed function

1. Function `split`: group values by their keys
2. Function `sortAll`: sort each list of values

Run time: \( O(k^2) \) where \( k \) is the length of the initial list

Naive overall bounding results in cubic bound:
3. Incorporate Individual Sizes

Example: sortAll in a composed function

1. Function split: group values by their keys

2. Function sortAll: sort each list of values

Run time: $O(k^2)$ where $k$ is the length of the initial list

Naive overall bounding results in cubic bound:

- length $k$: $O(k)$
- split: $O(k)$
- sortAll: $O(nm^2)$ with $n \leq k$ and $m \leq k$
3. Incorporate Individual Sizes

Example: sortAll in a composed function

1. Function split: group values by their keys
   ![Diagram of split function]

2. Function sortAll: sort each list of values
   ![Diagram of sortAll function]

Run time: $O(k^2)$ where $k$ is the length of the initial list

Naive overall bounding results in cubic bound:

- length $k$: $O(k)$
- split function: $O(k)$
- sortAll function: $O(k^3)$

$n \leq k$

$m \leq k$
3. Incorporate Individual Sizes

Example: sortAll in a composed function

1. Function split: group values by their keys

2. Function sortAll: sort each list of values

Run time: $O(k^2)$ where $k$ is the length of the initial list

Naive overall bounding results in cubic bound:

$O(k^3)$
3. Incorporate Individual Sizes

Example: `sortAll` in a composed function

1. Function `split`: group values by their keys

2. Function `sortAll`: sort each list of values

Run time: $O(k^2)$

Naive overall bounding results in cubic bound:

- length $k$ \(\rightarrow\) $O(k)$ \(\rightarrow\) $O(nm^2)$ \(\rightarrow\) $O(k^3)$

Our system correctly infers a quadratic bound.

$n \leq k$

$m \leq k$
Our Solution: Resource Polynomials

- Take into account the sizes of all inner data structures
- Represent a wide range of relations between different parts of the input
- Express tight bounds for many examples
- Generalize non-negative linear combinations of binomials
Resource Polynomials: Definition

Map data structures to non-negative rational numbers
\[ p : [A] \rightarrow \mathbb{Q}_0^+ \]

Are non-negative linear combinations of the following base polynomials:

\[ \mathcal{P}(\text{Int}) = \{ a \mapsto 1 \} \]

\[ \mathcal{P}(A_1, A_2) = \{ (a_1, a_2) \mapsto p_1(a_1) \cdot p_2(a_2) \mid p_i \in \mathcal{P}(A_i) \} \]

\[ \mathcal{P}(L(A)) = \{ [a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \cdots < j_k \leq n} \prod_{i=1}^{k} p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A) \} \]
Resource Polynomials: Definition

Map data structures to non-negative rational numbers

\[ p : [A] \to \mathbb{Q}_0^+ \]

Are non-negative linear combinations of the following base polynomials:

\[ \mathcal{P}(Int) = \{ a \mapsto 1 \} \]

\[ \mathcal{P}(A_1, A_2) = \{ (a_1, a_2) \mapsto p_1(a_1) \cdot p_2(a_2) \mid p_1, p_2 \in \mathcal{P}(A_i) \} \]

\[ \mathcal{P}(L(A)) = \{ [a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \cdots < j_k \leq n} \prod_{i=1}^k p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A) \} \]

Important innovation: sigma-pi formula for data structures
Resource Polynomials: Examples

\[ \mathcal{P}(L(A)) = \{[a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \cdots < j_k \leq n} \prod_{i=1}^{k} p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A)\} \]
Resource Polynomials: Examples

\[ [a_1, \ldots, a_n] \mapsto 8n + 3 \]

\[ \mathcal{P}(L(A)) = \left\{ [a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \cdots < j_k \leq n} \prod_{i=1}^{k} p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A) \right\} \]
Resource Polynomials: Examples

\[ [a_1, \ldots, a_n] \mapsto 8n + 3 \]

\[ [a_1, \ldots, a_n] \mapsto 36 \binom{n}{3} + 16 \binom{n}{2} + 20n + 3 \]

\[ \mathcal{P}(L(A)) = \{ [a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \ldots < j_k \leq n} \prod_{i=1}^{k} p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A) \} \]
Resource Polynomials: Examples

\[ [a_1, \ldots, a_n] \mapsto 8n + 3 \]

\[ [a_1, \ldots, a_n] \mapsto 36 \binom{n}{3} + 16 \binom{n}{2} + 20n + 3 \]

\[ ([a_1, \ldots, a_n], [b_1, \ldots, b_m]) \mapsto 39mn + 6m + 21n + 19 \]

\[ \mathcal{P}(L(A)) = \{[a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \cdots < j_k \leq n} \prod_{i=1}^{k} p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A) \} \]
Resource Polynomials: Examples

\[ [a_1, \ldots, a_n] \mapsto 8n + 3 \]

\[ [a_1, \ldots, a_n] \mapsto 36 \binom{n}{3} + 16 \binom{n}{2} + 20n + 3 \]

\[ ([a_1, \ldots, a_n], [b_1, \ldots, b_m]) \mapsto 39mn + 6m + 21n + 19 \]

\[ [[a_1^1, \ldots, a_{m_1}^1], \ldots, [a_1^n, \ldots, a_{m_n}^n]] \mapsto 18 \binom{n}{2} + 12n + 3 + \sum_{1 \leq i < j \leq n} 12m_i \]

\[ \mathcal{P}(L(A)) = \{ [a_1, \ldots, a_n] \mapsto \sum_{1 \leq j_1 < \cdots < j_k \leq n} \prod_{i=1}^{k} p_i(a_{j_i}) \mid k \in \mathbb{N}, p_i \in \mathcal{P}(A) \} \]
Automatic Computation of the Bounds

1. Fix a **maximal degree** of resource polynomials

2. **Annotate** each type with (yet unknown) coefficients for resource polynomials

   Example for degree 2: \(((L(int), L(int)), q_{0,0}, q_{1,0}, q_{2,0}, q_{1,1}, q_{0,1}, q_{0,2})\)

   General case: **index system** that enumerates resource polynomials

3. Extract **linear constraints** for the coefficients during type inference

4. Solve the constraints with an LP solver
Automatic Computation of the Bounds

1. **Fix a maximal degree** of resource polynomials

2. **Annotate** each type with (yet unknown) coefficients for resource polynomials

   Example for degree 2: \((L(int), L(int)), q_{0,0}, q_{1,0}, q_{2,0}, q_{1,1}, q_{0,1}, q_{0,2}\)

   General case: **index system** that enumerates resource polynomials

3. **Extract linear constraints** for the coefficients during type inference

4. **Solve the constraints with an LP solver**
Experimental Evaluation
<table>
<thead>
<tr>
<th>Description</th>
<th>Computed Bound</th>
<th>Actual Behavior</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick Sort (Integers)</td>
<td>$12n^2 + 14n + 3$</td>
<td>$O(n^2)$</td>
<td>0.1 s</td>
</tr>
<tr>
<td>Split and Sort</td>
<td>$16n^2 + 46n + 9$</td>
<td>$O(n^2)$</td>
<td>2.1 s</td>
</tr>
<tr>
<td>Insertion Sort (Strings)</td>
<td>$8n^2m + 8n^2 - 8nm + 4n + 3$</td>
<td>$O(n^2m)$</td>
<td>0.91 s</td>
</tr>
<tr>
<td>Duplicate Elimination</td>
<td>$6n^2m + 9n^2 - 6nm + 3n + 3$</td>
<td>$O(n^2m)$</td>
<td>0.97 s</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>$28xmn + 32xm + 2x + 14n + 21$</td>
<td>$O(xmn)$</td>
<td>1.96 s</td>
</tr>
<tr>
<td>Longest Common Subsequence</td>
<td>$39nm + 6m + 21n + 19$</td>
<td>$O(nm)$</td>
<td>0.36 s</td>
</tr>
<tr>
<td>Subtrees</td>
<td>$4n^2 + 19n + 3$</td>
<td>$O(n^2)$</td>
<td>0.06 s</td>
</tr>
</tbody>
</table>

**Evaluation-Step Bounds**
Quick Sort for Integers

Evaluation-step bound vs. measured behavior
Longest Common Subsequence

Evaluation-step bound vs. measured behavior

measured worst-case steps

39xy + 6y + 21x + 19
Insertion Sort for Strings

Evaluation-step bound vs. measured behavior
Matrix Multiplication with Transposition

Evaluation-step bound vs. measured behavior

measured worst-case cost

$28x^2y + 32xy + 16x + 21$
measured worst-case cost

$15x^2y + 16xy + 15x + 3$

Matrix Multiplication with Accumulation  Evaluation-step bound vs. measured behavior
Conclusion

Multivariate Amortized Resource Analysis

- **Precise:** bounds are resource polynomials
- **Efficient:** inference via linear programming
- **Reliable:** formal soundness proof of the bounds
- **Verifiable:** type derivation is a certificate

Future Work

- Non-polynomial bounds
- Programs with garbage collection
- Imperative programs with loops and arrays
Resource Aware ML
http://raml.tcs.ifi.lmu.de/

I am looking for a post-doc position.
uni@hoffjan.de