Automatic Resource Bound Analysis and Linear Optimization

Jan Hoffmann

Carnegie Mellon

Beyond worst-case analysis

Beyond worst-case analysis

… of algorithms

Beyond worst-case analysis

… of algorithms

Resource bound analysis

Beyond worst-case analysis

… of algorithms

Resource bound analysis

… of programs / software

Motivation: Why analyze resource usage of programs?

Resource Usage in Safety-Critical Systems

Memory Usage

Timing

Resource Usage in Safety-Critical Systems

Memory Usage


About Us • Subscribe to Newsletters

DESIGN CENTERS • TOOLS & LEARNING • COMMUNITY • ED

Home > Automotive Design Center > How To Article

Toyota's killer firmware: Bad design and its consequences

Michael Dunn -October 28, 2013

109 Comments

On Thursday October 24, 2013, an Oklahoma court ruled against Toyota in a case of unintended acceleration that lead to the death of one the occupants. Central to the trial was the Engine Control Module's (ECM) firmware.

Unintended acceleration in Toyota cars in the US 2005-2009.

Timing

Resource Usage in Safety-Critical Systems

Memory Usage


About Us • Subscribe to Newsletters

EDN NETWORK

DESIGN CENTERS TOOLS & LEARNING COMMUNITY ED

Home > Automotive Design Center > How To Article

Toyota's killer firmware: Bad design and its consequences

Michael Dunn -October 28, 2013

109 Comments

Share 277 8+1 932 Tweet 724 Like 3.8k

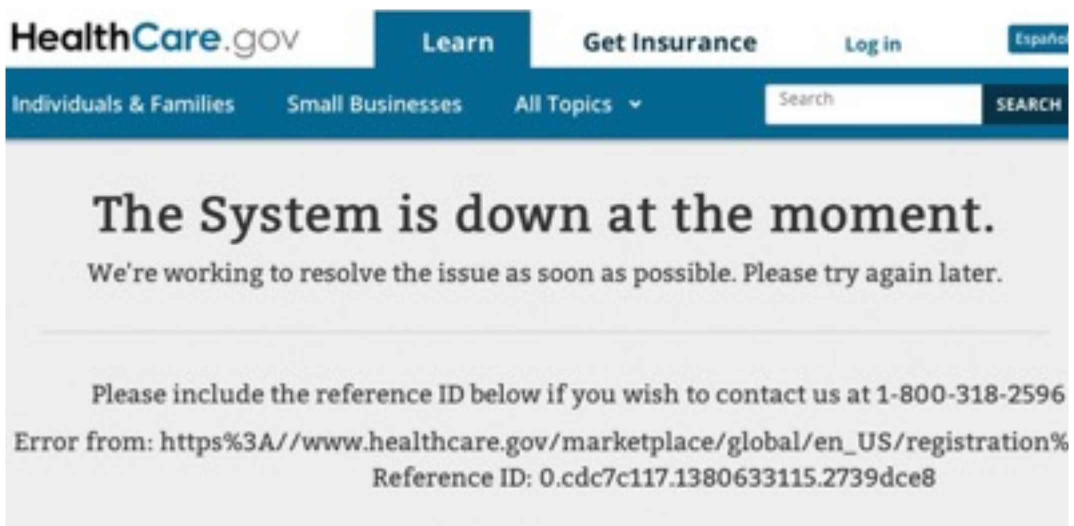
On Thursday October 24, 2013, an Oklahoma court ruled against Toyota in a case of unintended acceleration that lead to the death of one the occupants. Central to the trial was the Engine Control Module's (ECM) firmware.

Unintended acceleration in Toyota cars in the US 2005-2009.

Timing

ICE 3 Velaro D delivery delayed by one year because of software performance issues in 2013.

Performance Bugs are Common and Expensive

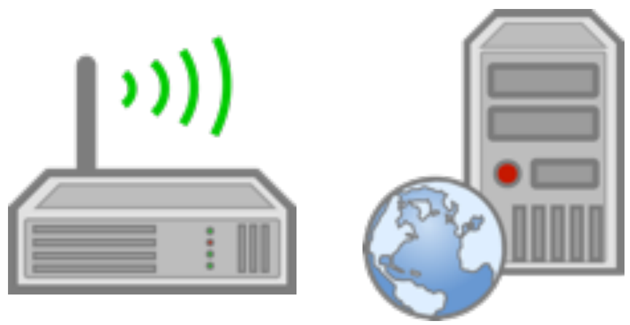


HealthCare.gov debacle has been mainly caused by performance issues.

Google Apps

Google Apps Developer: "Cannot test for performance bugs with regression test".

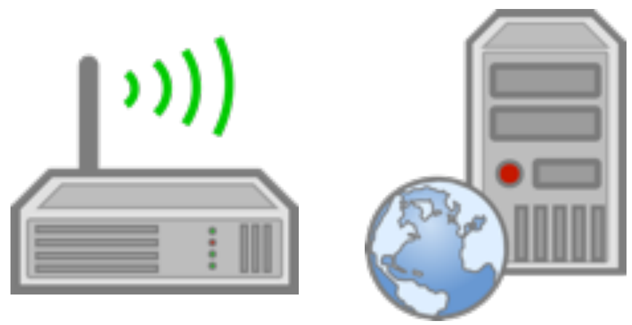
Software Security



Algorithmic Complexity Attacks

Side-Channel Attacks

Software Security



Algorithmic Complexity Attacks

Side-Channel Attacks

Space/Time Analysis for Cybersecurity (STAC)

October 2014
$ 53M program

Our team:

GRAMMATECH
Carnegie Mellon
Yale
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Static Resource Bound Analysis

Given:	A program P
Question:	What is the worst-case resource consumption of P as a function of the size of its inputs?

Static Resource Bound Analysis

Clock cycles, heap space, power, ...

Given: A program P

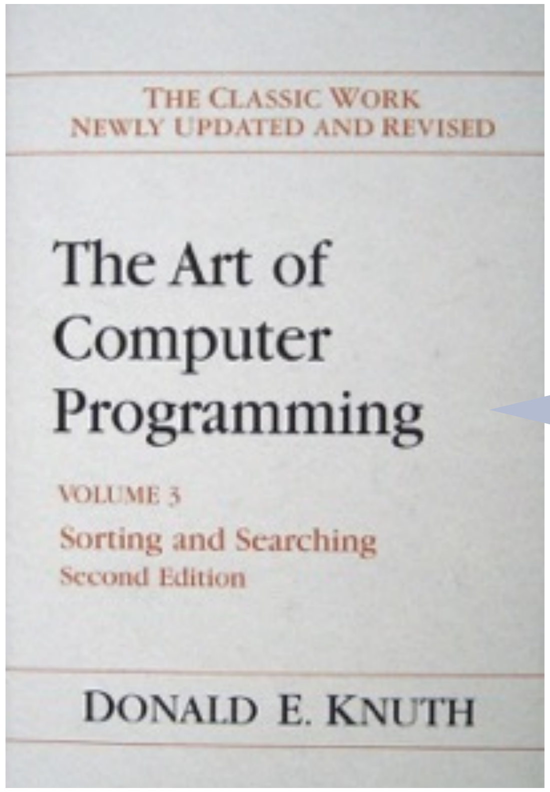
Question: What is the worst-case resource consumption of P as a function of the size of its inputs?

Static Resource Bound Analysis

Given: A program P

Clock cycles, heap space, power, ...

Question: What is the worst-case resource consumption of P as a function of the size of its inputs?



Not only asymptotic bounds but concrete constant factors.

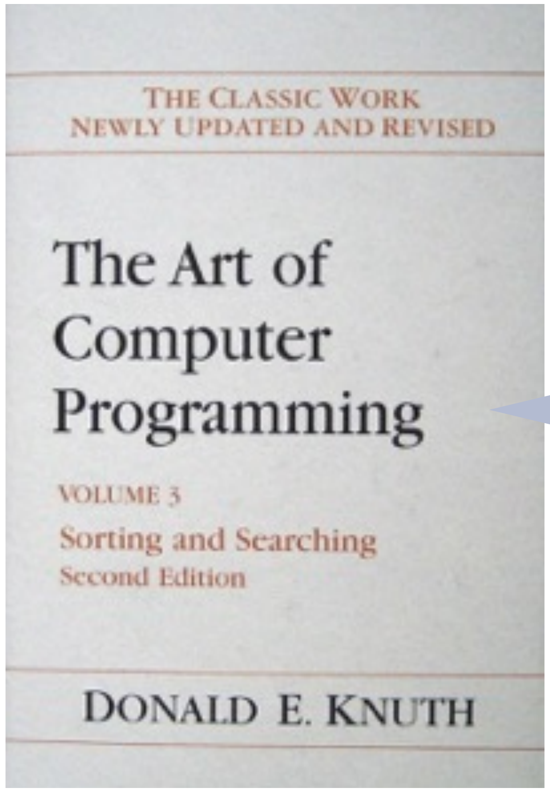
Automatic

Static Resource Bound Analysis

Given: A program P

Clock cycles, heap space, power, ...

Question: What is the worst-case resource consumption of P as a function of the size of its inputs?



Not only asymptotic bounds but concrete constant factors.

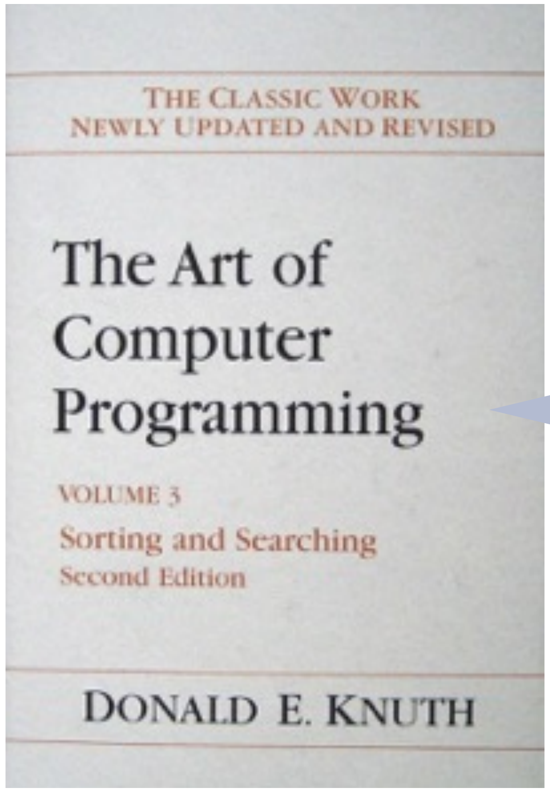
Automatic Static Resource Bound Analysis

Undecidable problem

Given: A program P

Question: What is the worst-case resource consumption of P as a function of the size of its inputs?

Clock cycles, heap space, power, ...



Not only asymptotic bounds but concrete constant factors.

Research Challenges

Model and predict resource usage at development time.

Source code

Research Challenges

Model and predict resource usage at development time.

Compiler

Libraries

Source code

Run-time system

Hardware

Research Challenges

Help developers to reason about quantitative properties.

- **Computer support:** Modeling, specification, verification, automation and user interaction

- **Compositionally:** Track size changes and specify resource-usage of library code

- **Language features:** Concurrency, higher-order, data structures, …

Why is this Related?

1. Beyond worst-case analysis *of algorithms*

- Automation
- Concrete (non-asymptotic) bounds for specific hardware

2. We face similar challenges in resource-bound analysis

- “The worst-case behavior doesn’t happen in practice.”

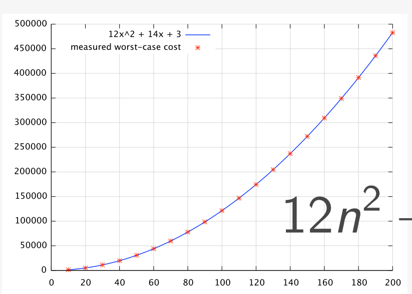
3. We reduce bound inference to linear optimization

- Linear programs we get are solvable in linear time in practice
- Theoretical worst-case of the algorithm is exponential (simplex)

Outline

- Motivation ✓
- **How does automatic resource bound analysis work?**
- How **well** does automatic resource bound analysis work? (implementation and experiments)
- What are the properties of the LP instances that we get?

Source Code



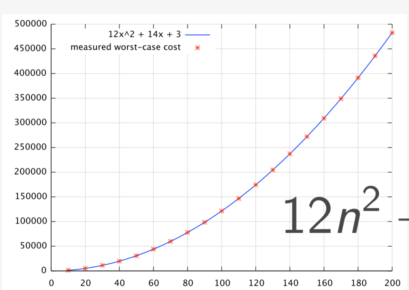
$$12n^2 + 14n + 3$$

Resource Bound

Bird's Eye View

Type-Based Resource Analysis

Source Code



$12n^2 + 14n + 3$

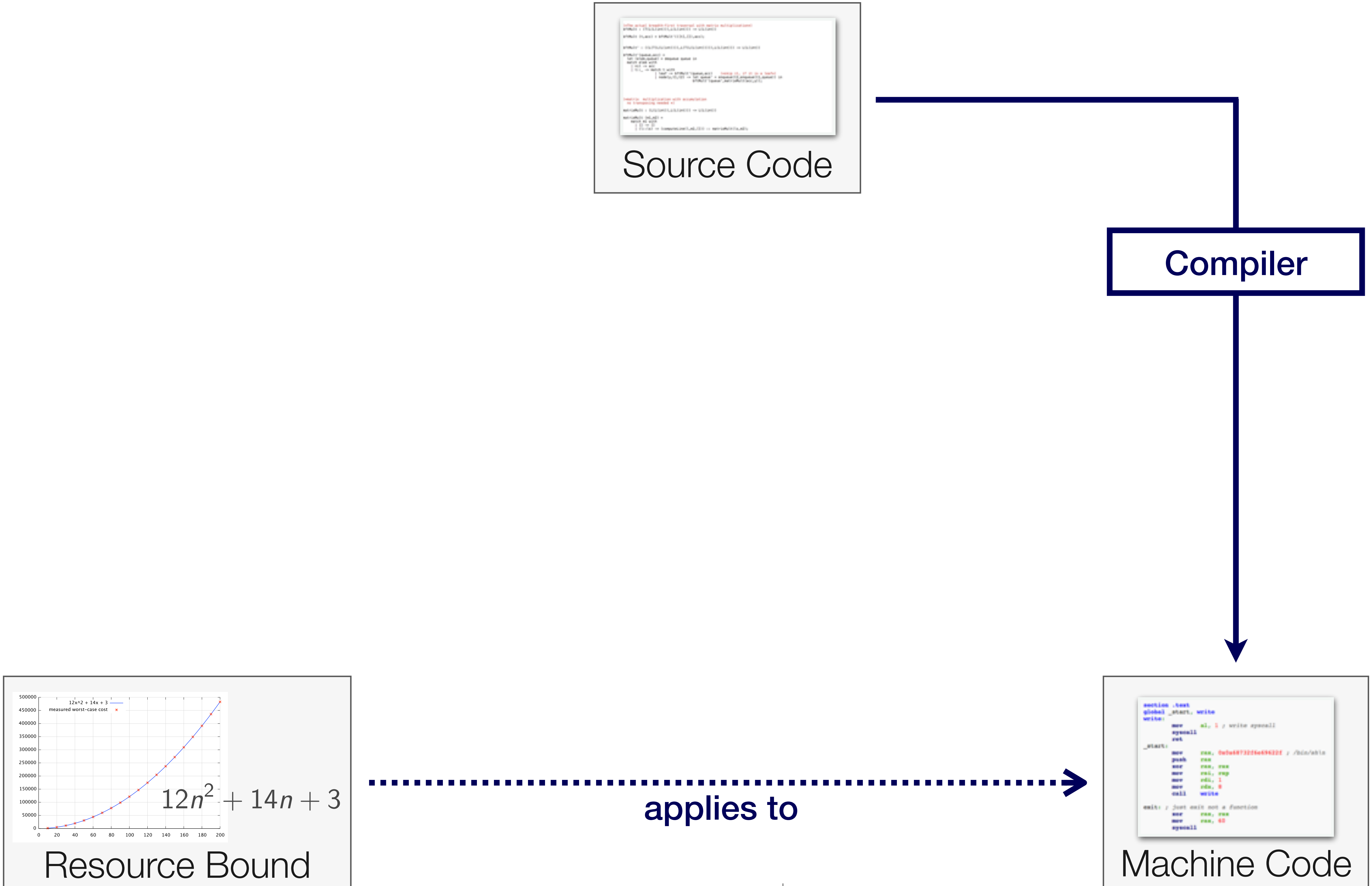
Resource Bound

applies to

Machine Code

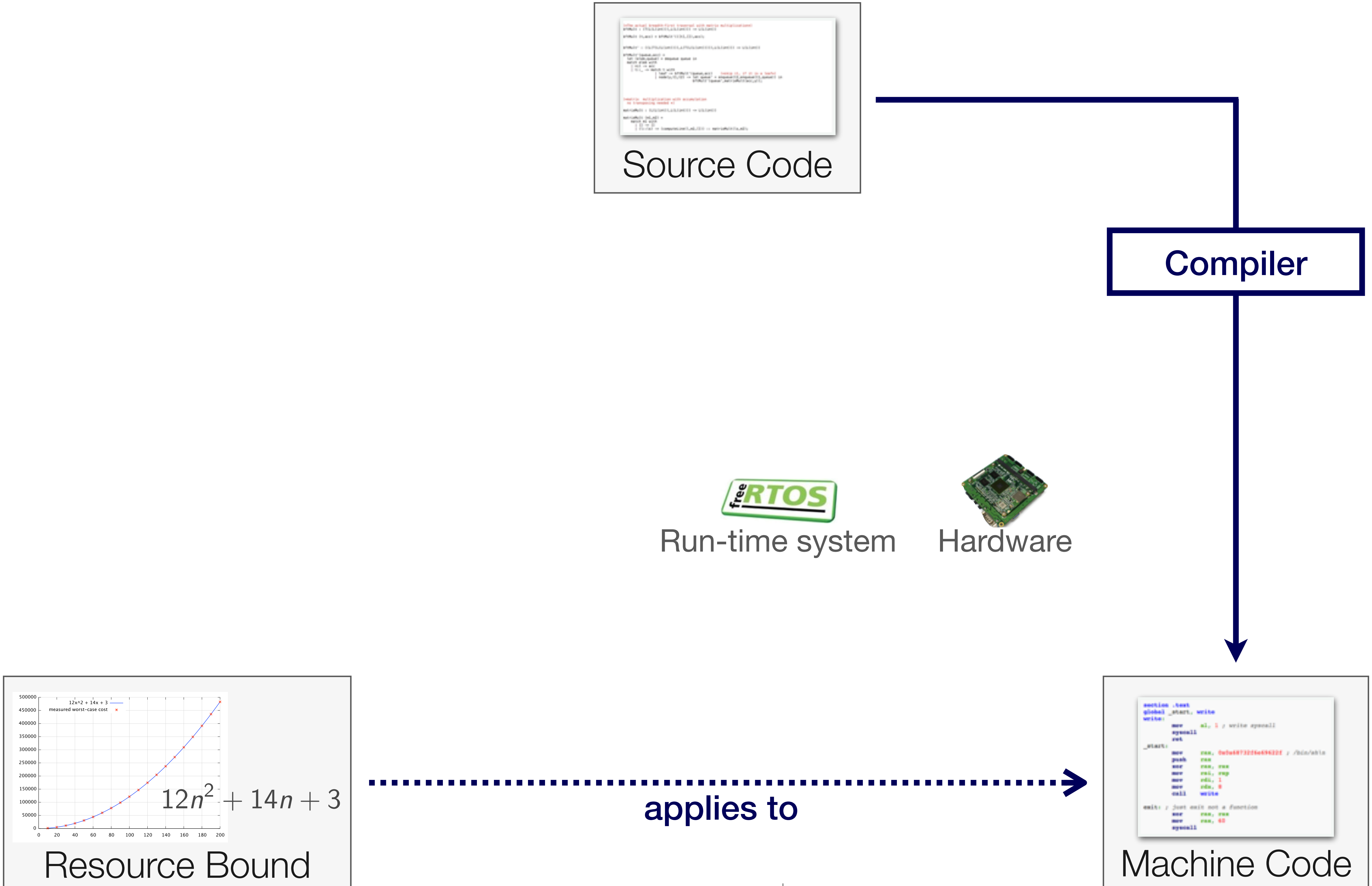
Bird's Eye View

Type-Based Resource Analysis



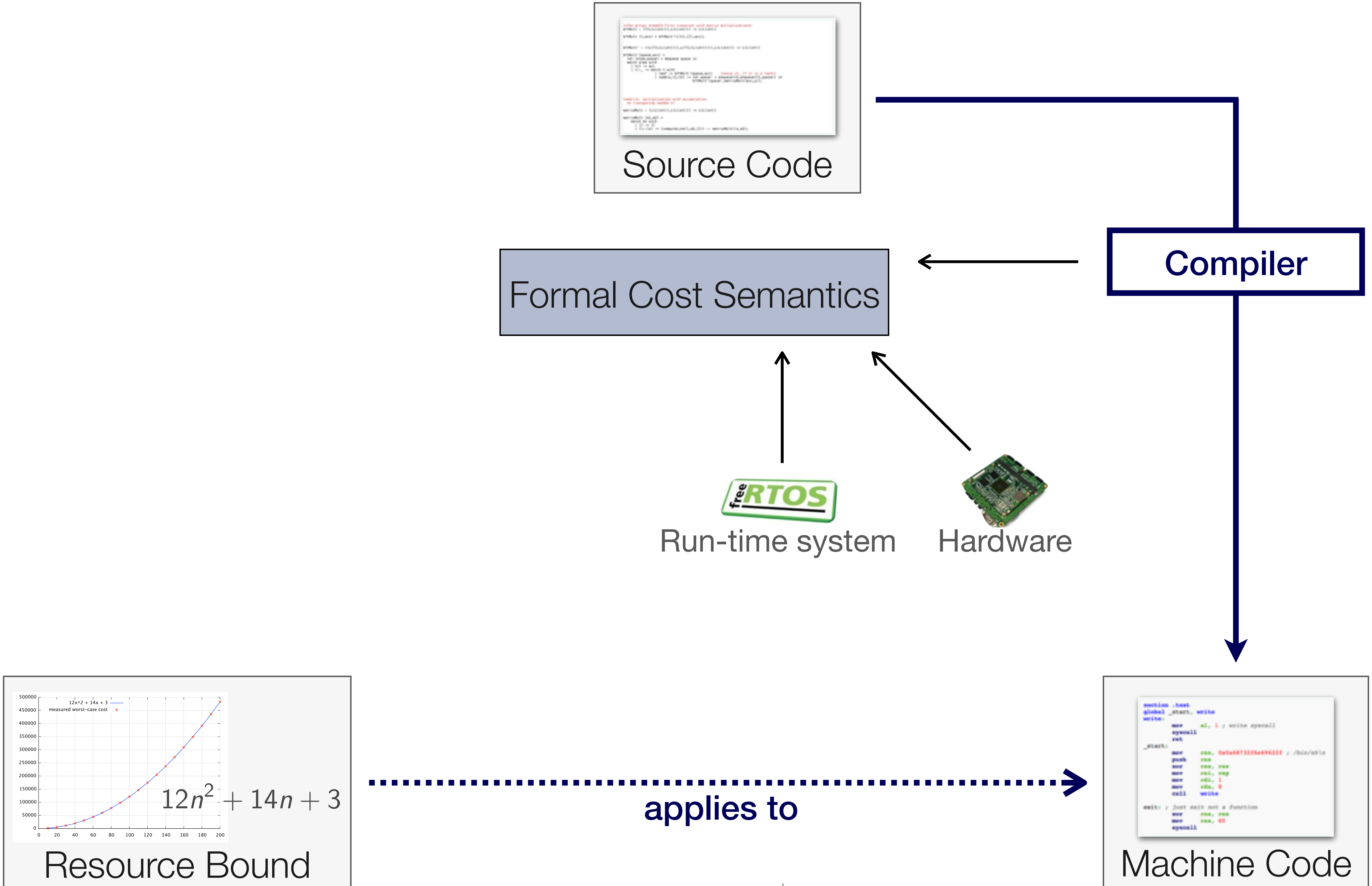
Bird’s Eye View

Type-Based Resource Analysis



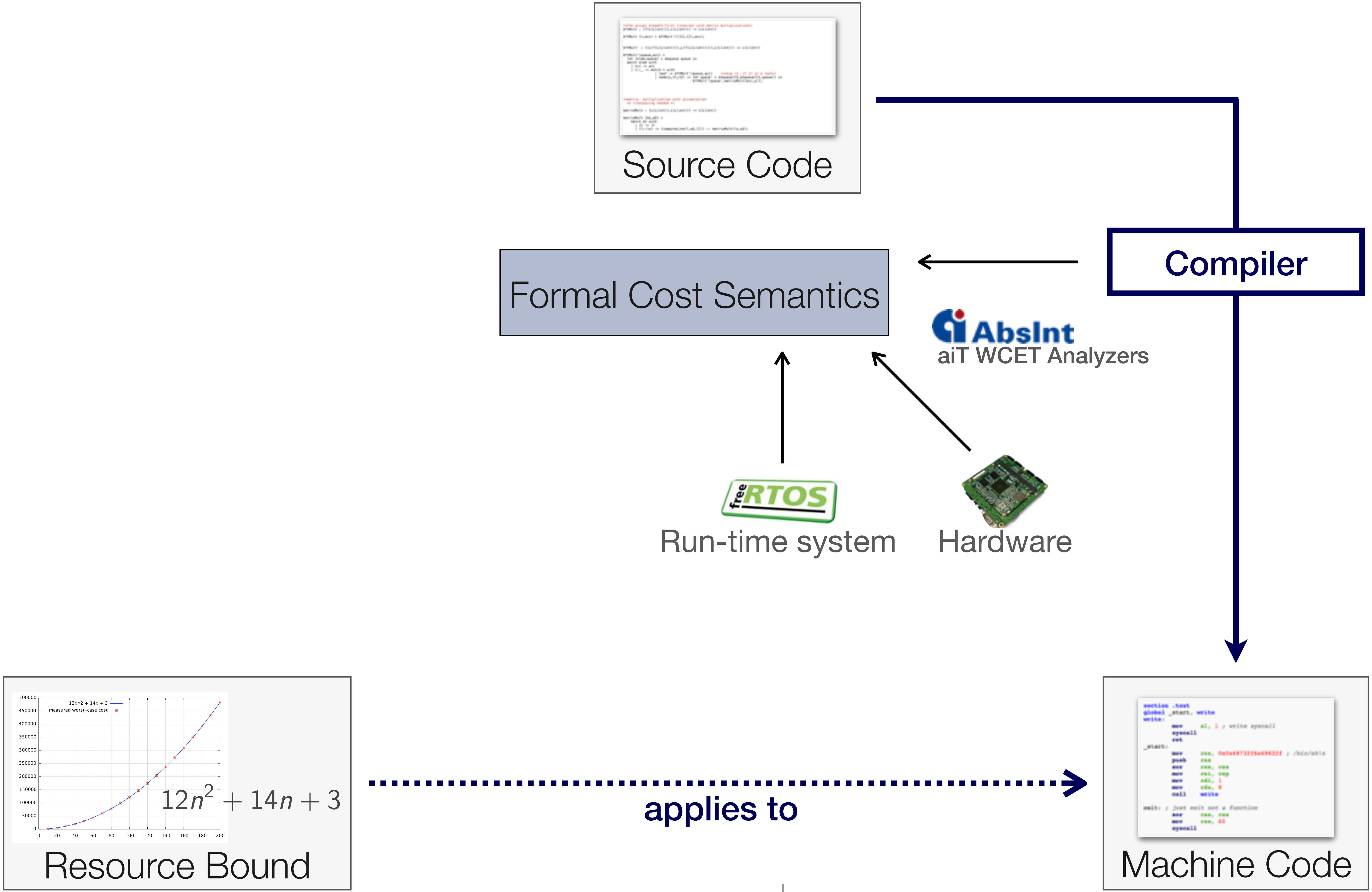
Bird’s Eye View

Type-Based Resource Analysis



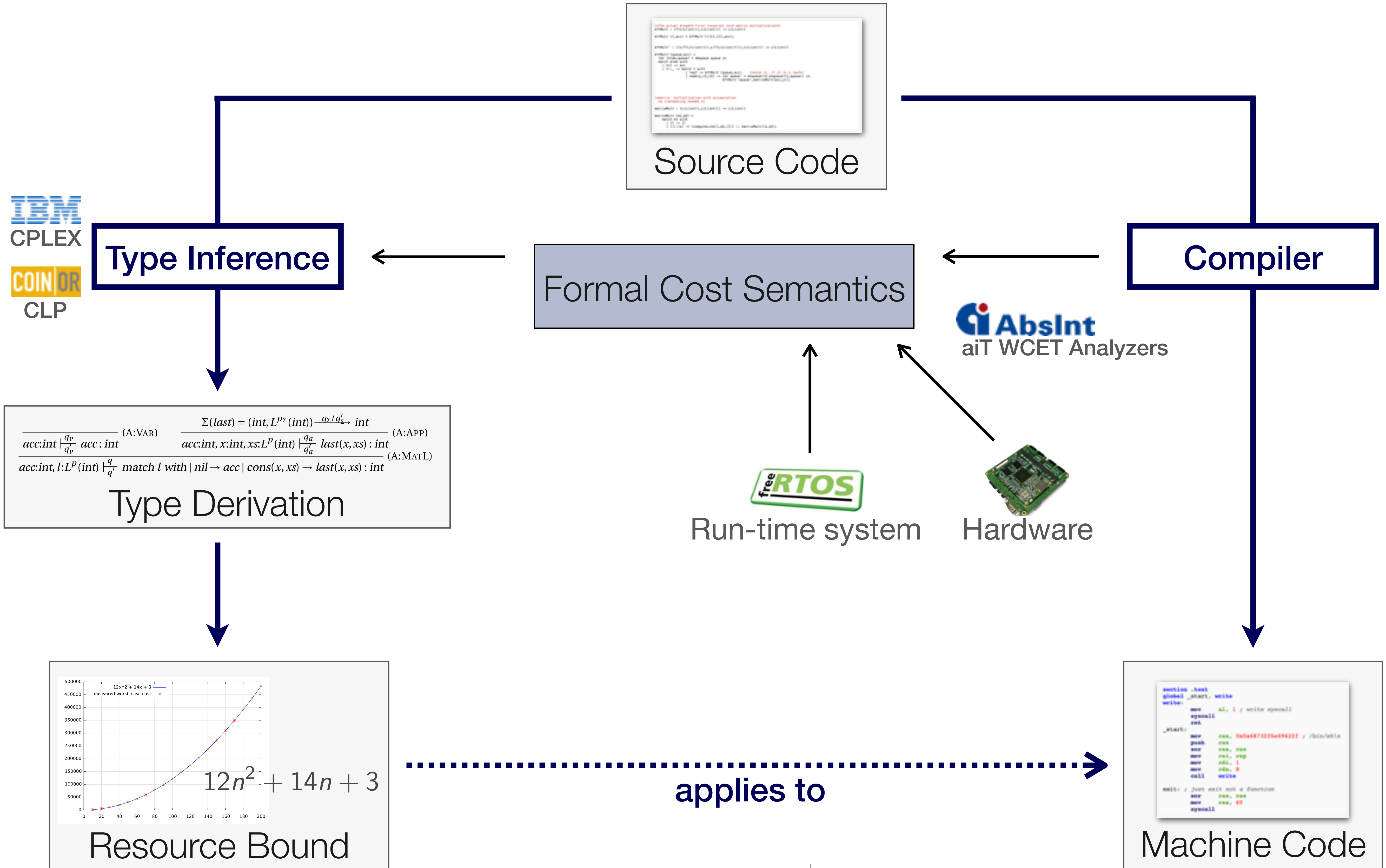
Bird’s Eye View

Type-Based Resource Analysis



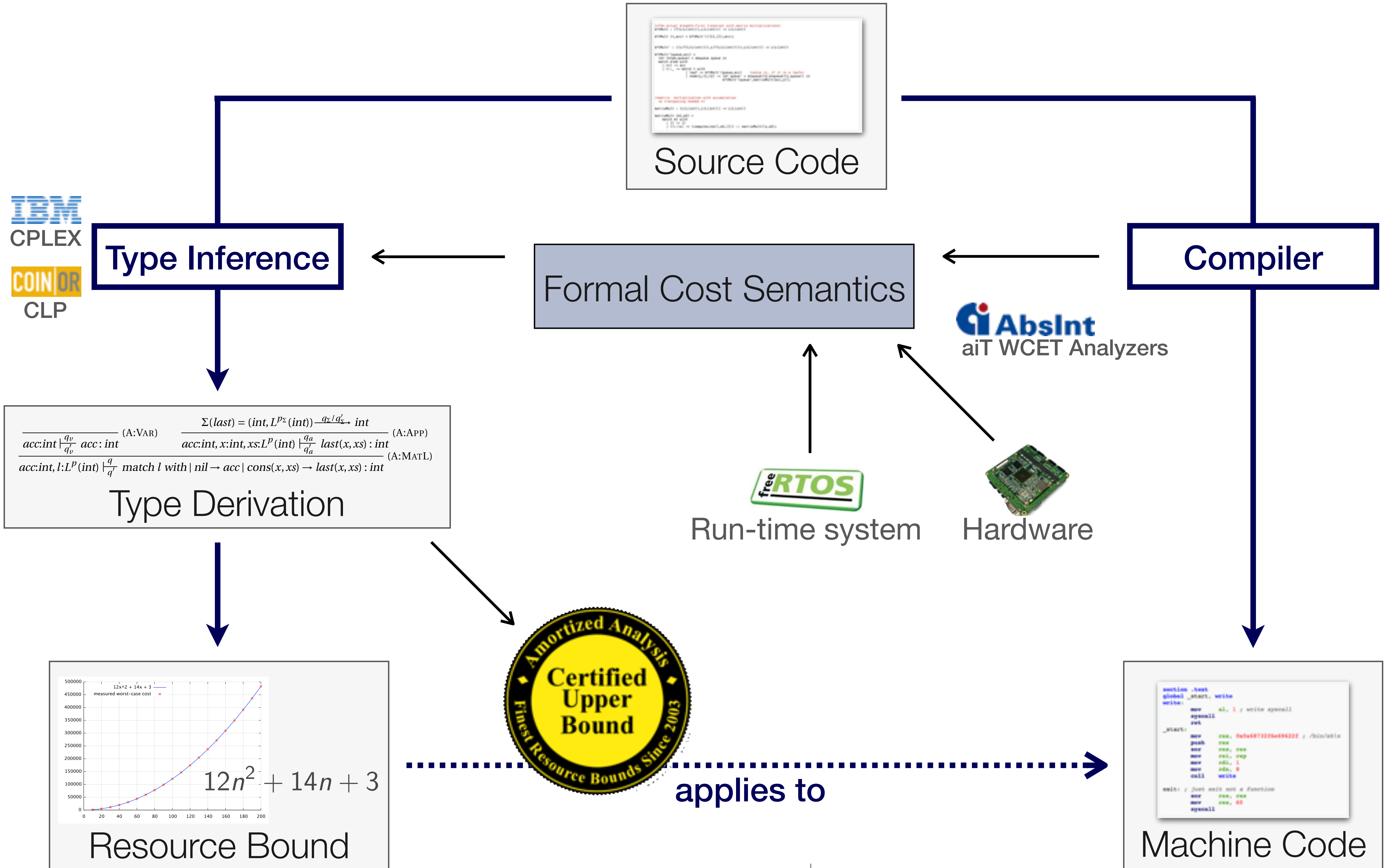
Bird's Eye View

Type-Based Resource Analysis



Bird's Eye View

Type-Based Resource Analysis



Bird's Eye View

Type-Based Resource Analysis

Source Code

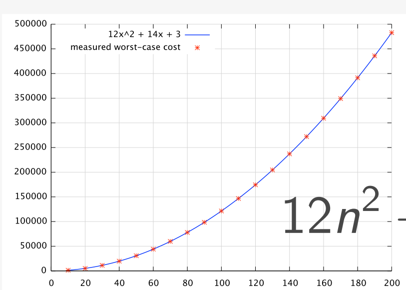
Type Inference

CPLEX

CLP

$$\dfrac{\dfrac{}{acc{:}int \vdash^{q_v}_{q'_v} acc : int}\,(\text{A:Var}) \qquad \dfrac{\Sigma(last) = (int, L^{p_\Sigma}(int)) \xrightarrow{q_\Sigma / q'_\Sigma} int}{acc{:}int, x{:}int, xs{:}L^{p}(int) \vdash^{q_a}_{q'_a} last(x,xs) : int}\,(\text{A:App})}{acc{:}int, l{:}L^{p}(int) \vdash^{q}_{q'} match\ l\ with \mid nil \to acc \mid cons(x,xs) \to last(x,xs) : int}\,(\text{A:MatL})$$

Type Derivation



$12n^2 + 14n + 3$

Resource Bound

Bird's Eye View

Type-Based Resource Analysis

Source Code

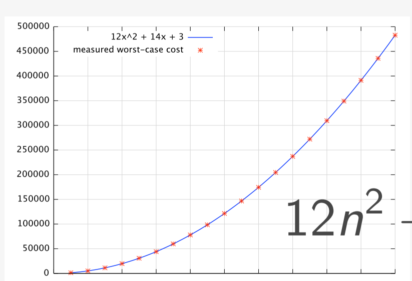
IBM CPLEX

COIN|OR CLP

Type Inference

$$\dfrac{\dfrac{}{acc{:}int \vdash^{q_v}_{q_v'} acc : int}\ (\text{A:Var}) \qquad \dfrac{\Sigma(last) = (int, L^{p_\Sigma}(int)) \xrightarrow{q_\Sigma / q_\Sigma'} int}{acc{:}int, x{:}int, xs{:}L^{p}(int) \vdash^{q_a}_{q_a'} last(x,xs) : int}\ (\text{A:App})}{acc{:}int, l{:}L^{p}(int) \vdash^{q}_{q'} \text{match } l \text{ with} \mid nil \to acc \mid cons(x,xs) \to last(x,xs) : int}\ (\text{A:MatL})$$

Type Derivation



$12n^2 + 14n + 3$

Resource Bound

Clear soundness theorem.

Naturally compositional.

Efficient inference via LP solving.

Bird’s Eye View

Idea: Automate Amortized Analysis

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

$$\Phi(\mathit{state}) \geq 0$$

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

$$\Phi(\mathit{before}) \geq \Phi(\mathit{after}) + \mathit{cost}$$

$\Downarrow$ telescoping $\Downarrow$

- Initial potential is an upper bound

$$\Phi(\mathit{initial\ state}) \geq \sum \mathit{cost}$$

Idea: Automate Amortized Analysis

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

$$\Phi(\mathit{state}) \geq 0$$

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

$$\Phi(\mathit{before}) \geq \Phi(\mathit{after}) + \mathit{cost}$$

$\downarrow$ telescoping $\downarrow$

- Initial potential is an upper bound

$$\Phi(\mathit{initial\ state}) \geq \sum \mathit{cost}$$

Type systems for automatic analysis

- Fix a format of potential functions
- Develop type rules that manipulate potential functions

Idea: Automate Amortized Analysis

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

$$\Phi(state) \geq 0$$

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

$$\Phi(before) \geq \Phi(after) + cost$$

telescoping

- Initial potential is an upper bound

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

Type systems for automatic analysis

Potential is given by types.

- Fix a format of potential functions
- Develop type rules that manipulate potential functions

Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:

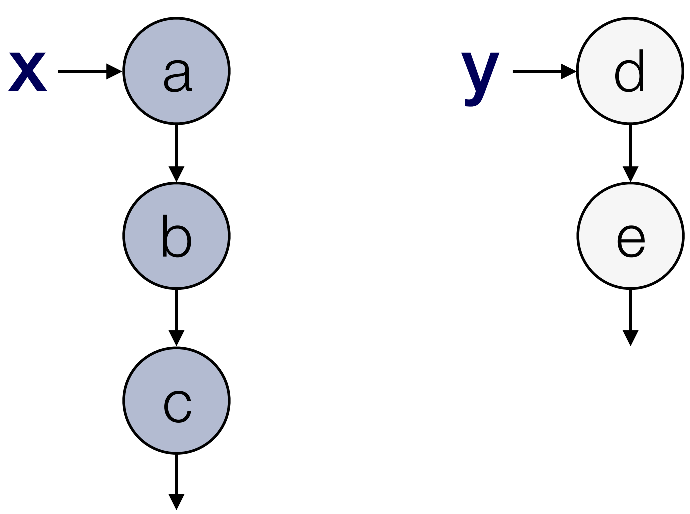
Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:



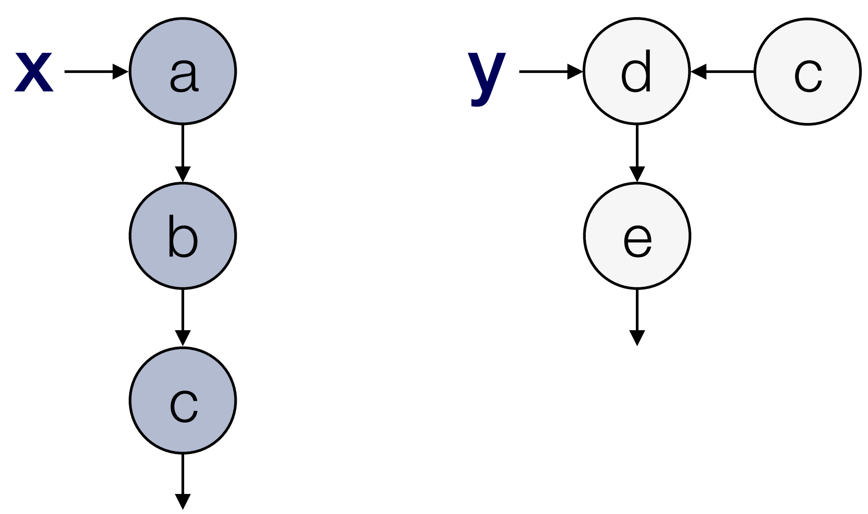
Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:



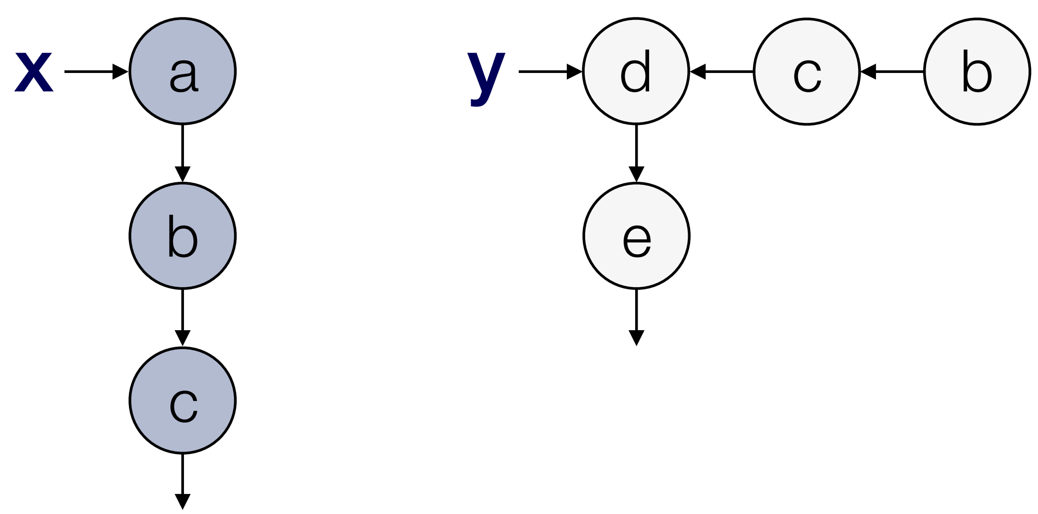
Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:



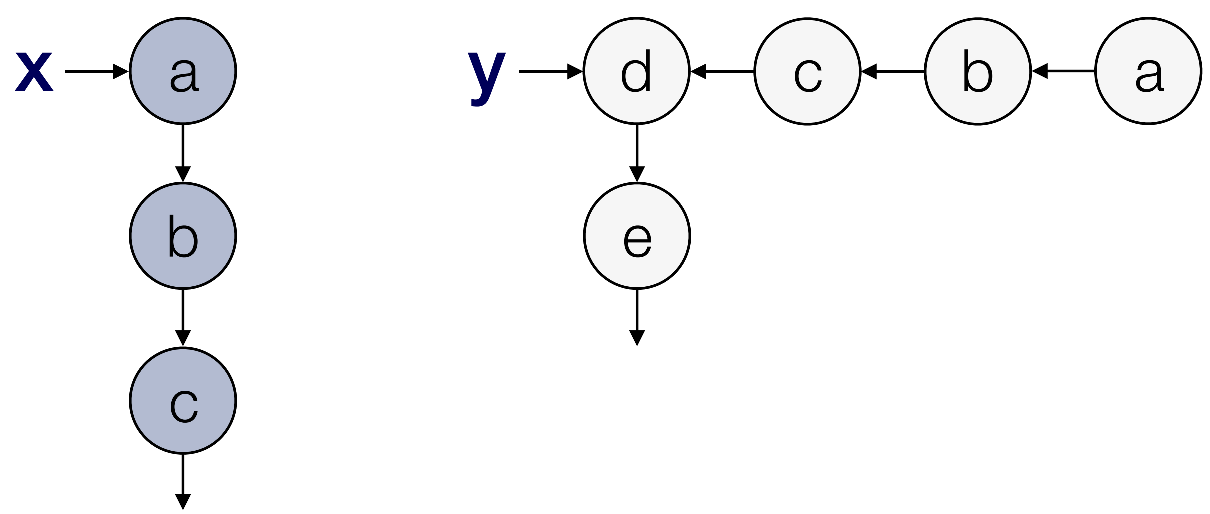
Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:



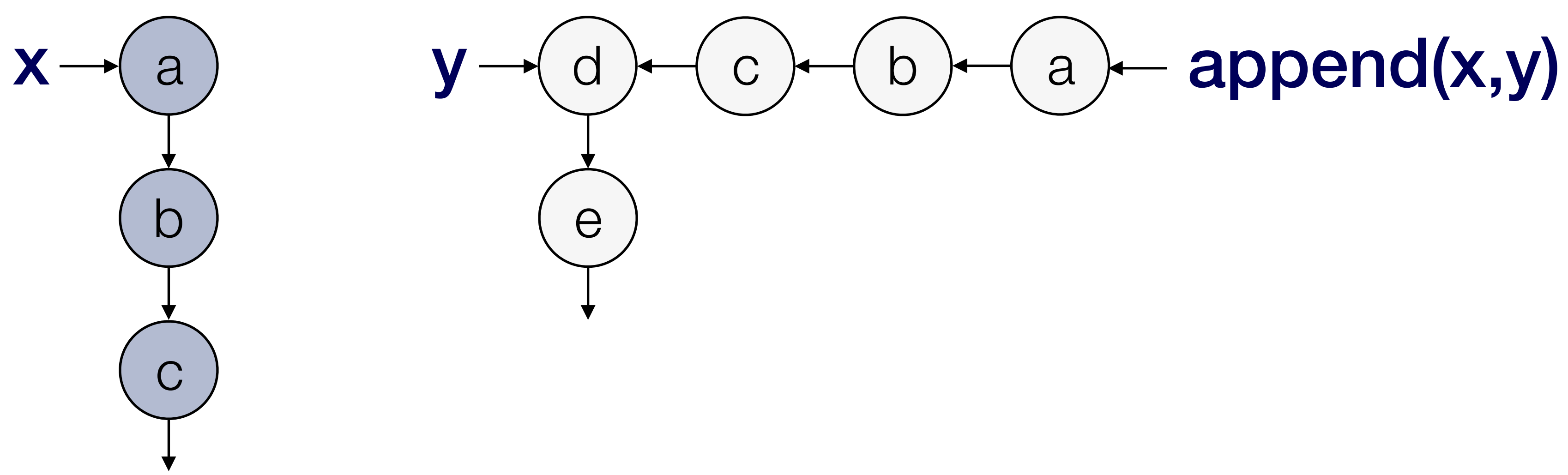
Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:



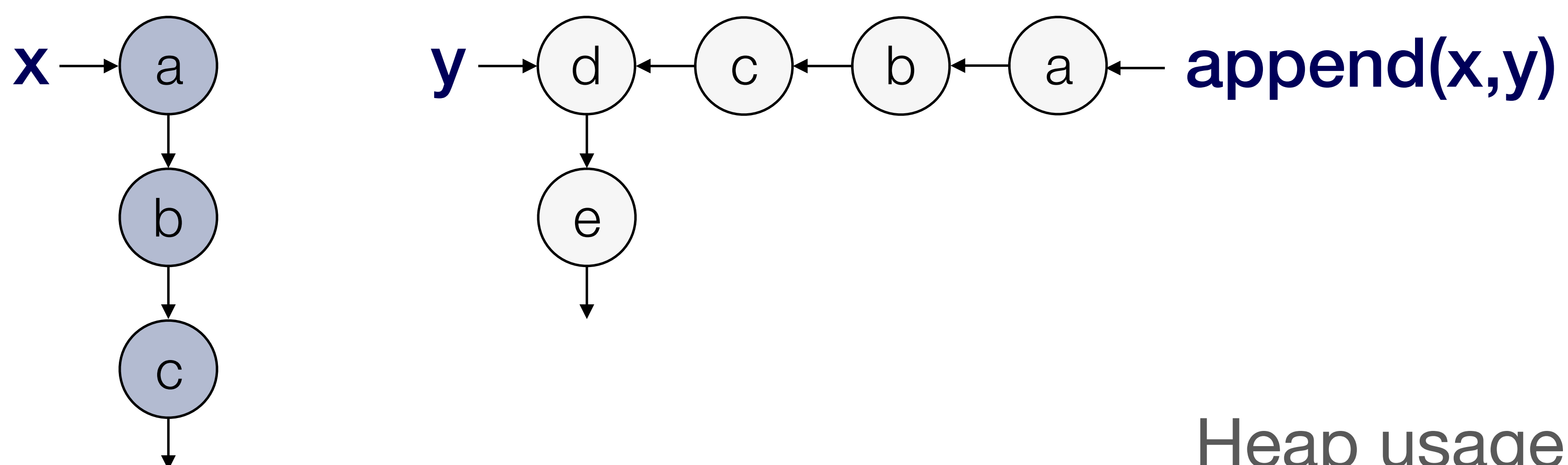
Example: Append for Persistent Lists

```
append(x,y)
```

Heap-space usage is 2n if

- n is the length of list x
- One list element requires two heap cells (data and pointer)

Example evaluation:



Heap usage: 2*n = 2*3 = 6

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is $2n + 2(n+m)$ if

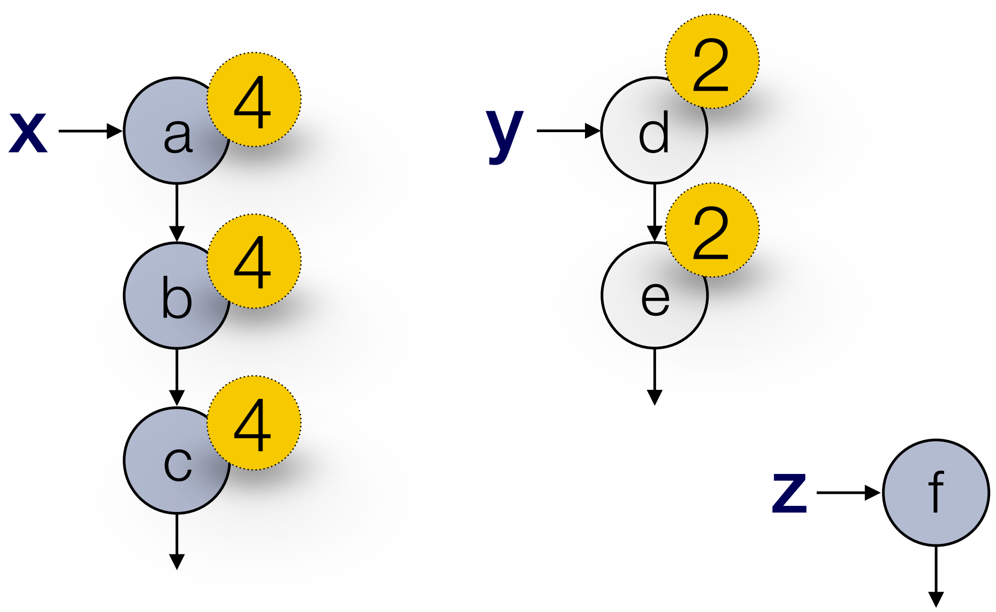
- n is the length of list x
- m is the length of list y

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



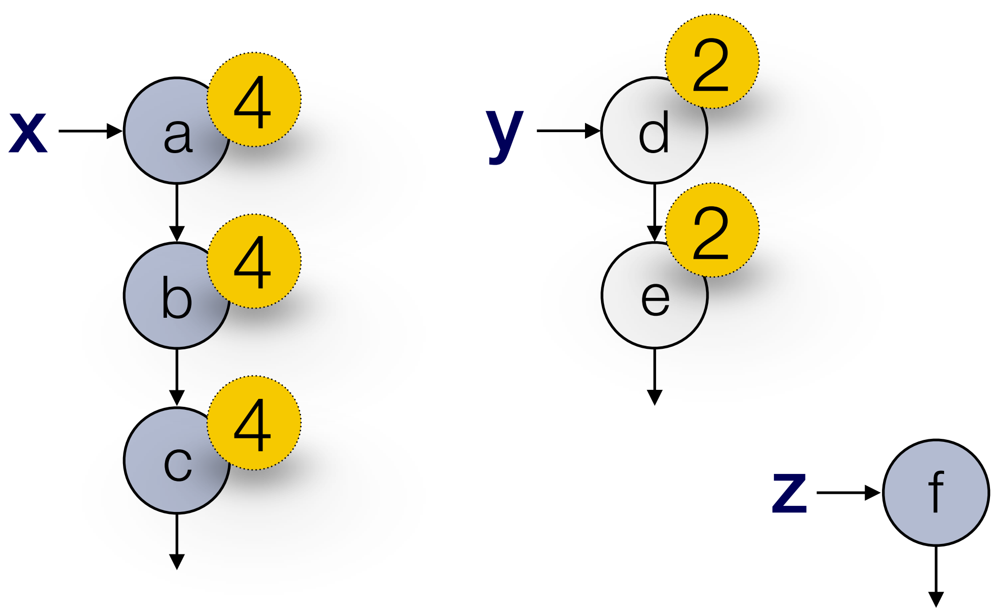
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



append(x,y)

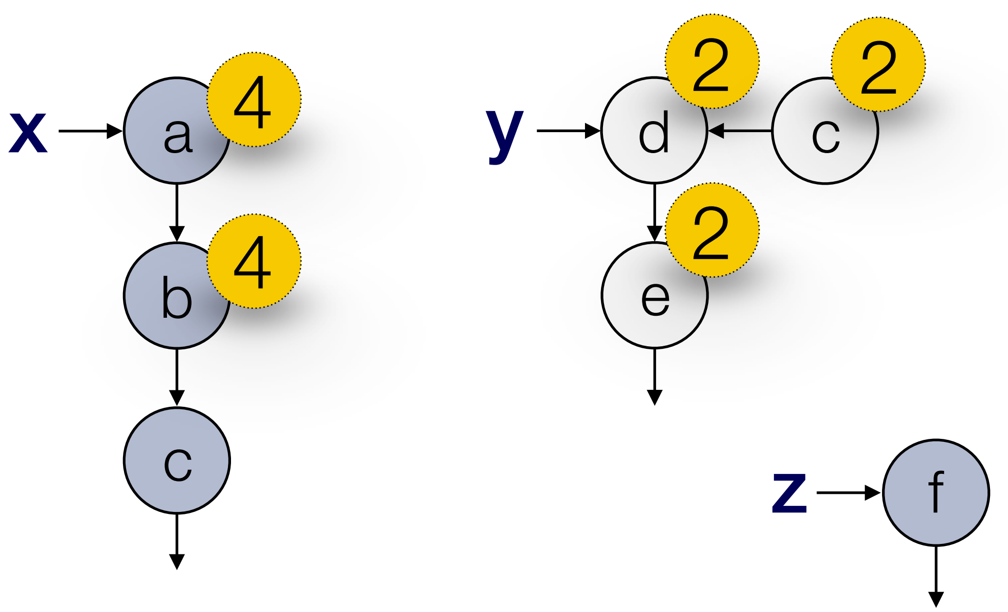
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



append(x,y)

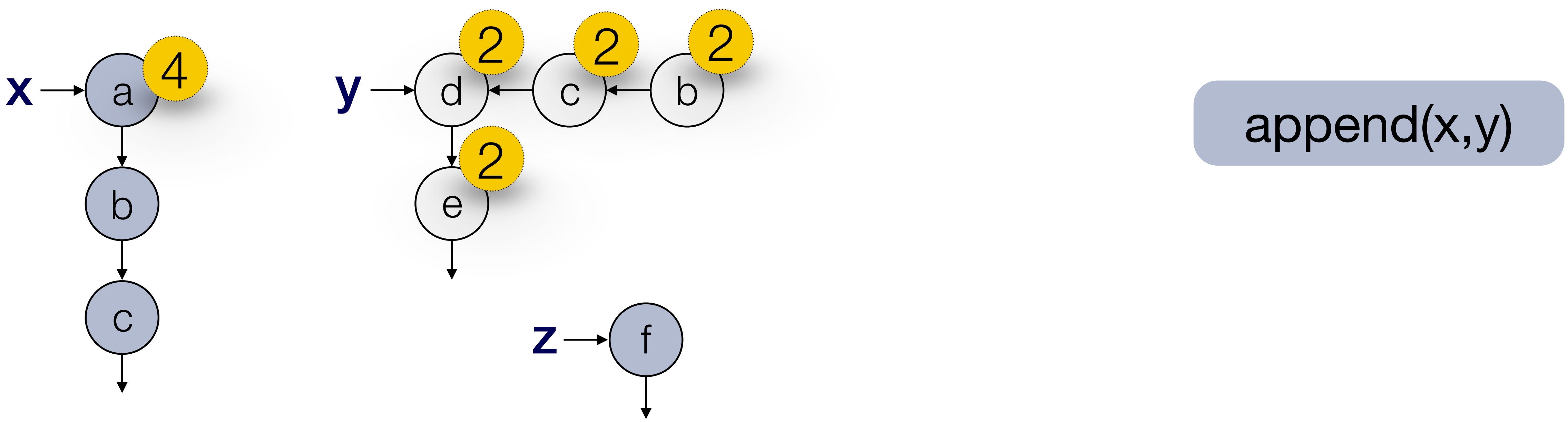
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



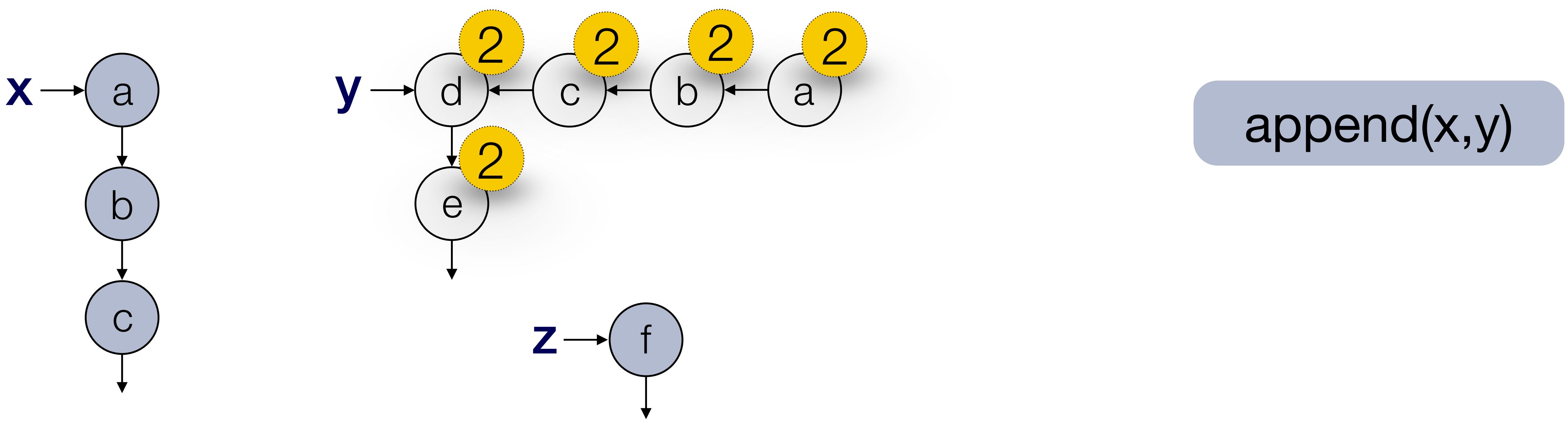
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



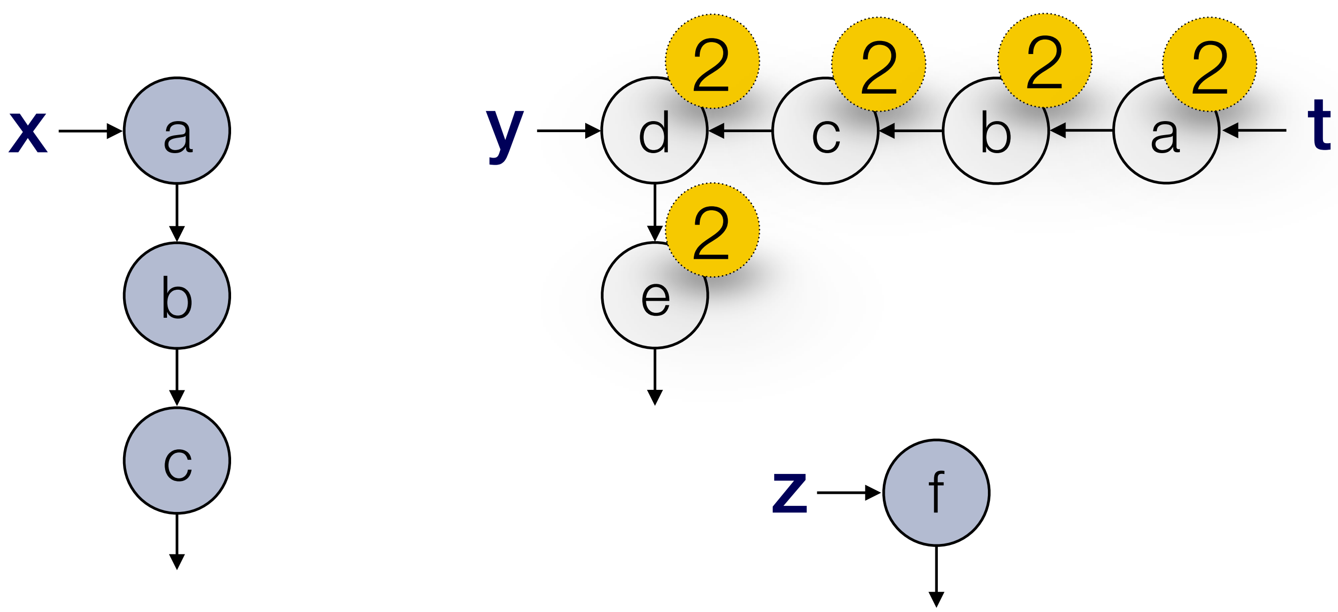
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



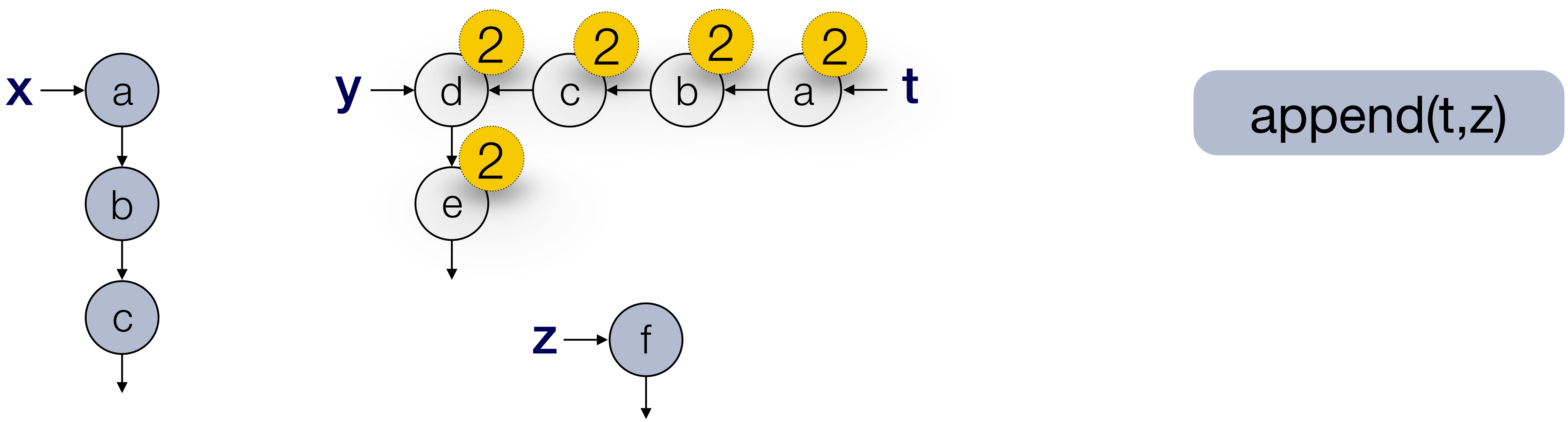
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



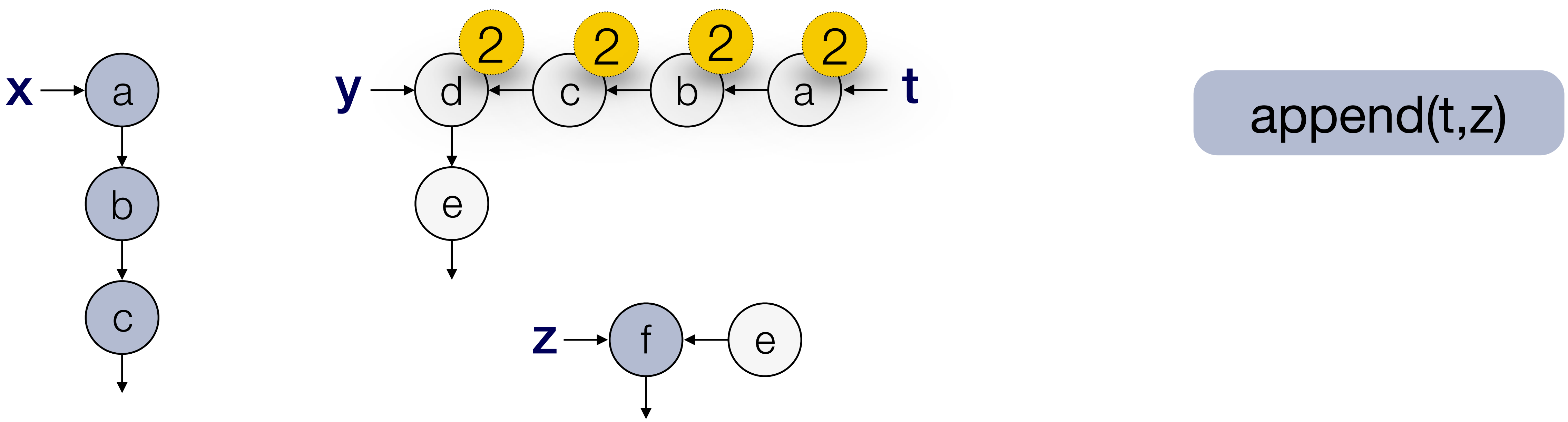
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



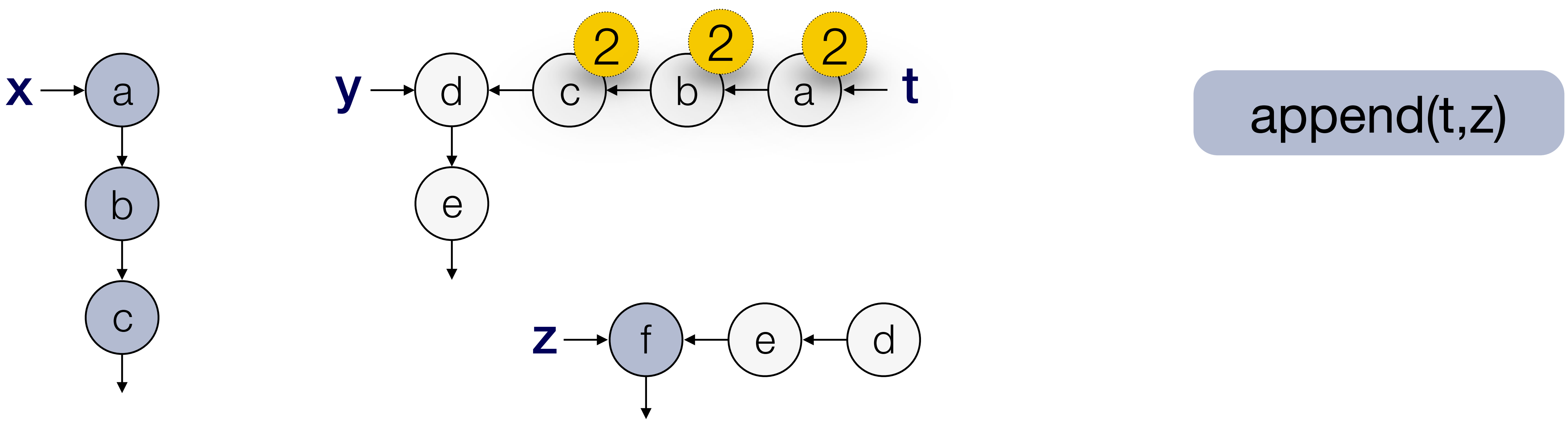
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



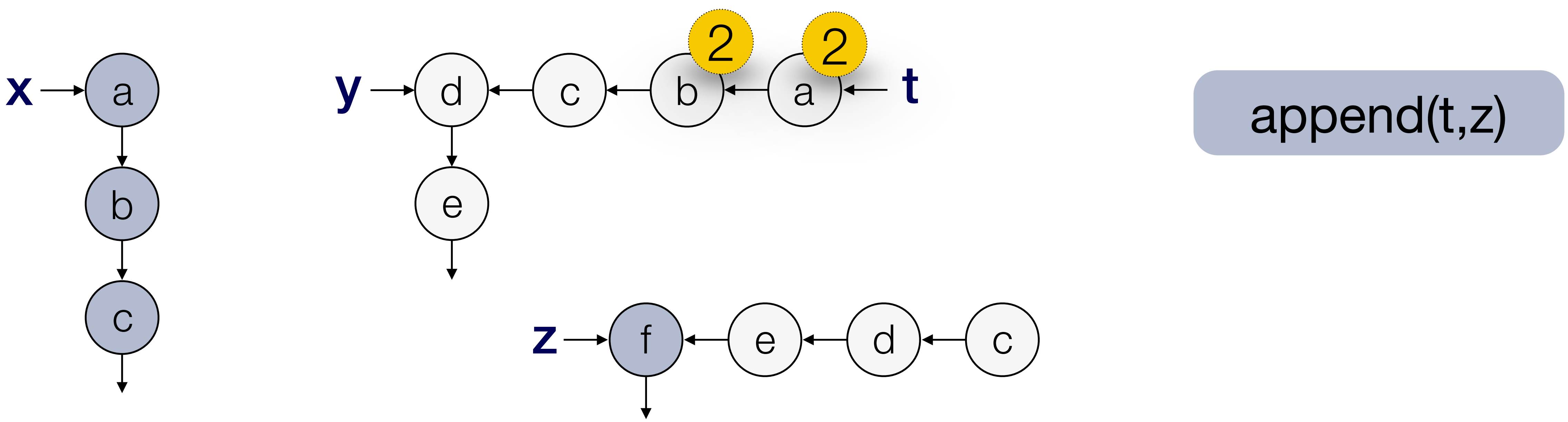
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



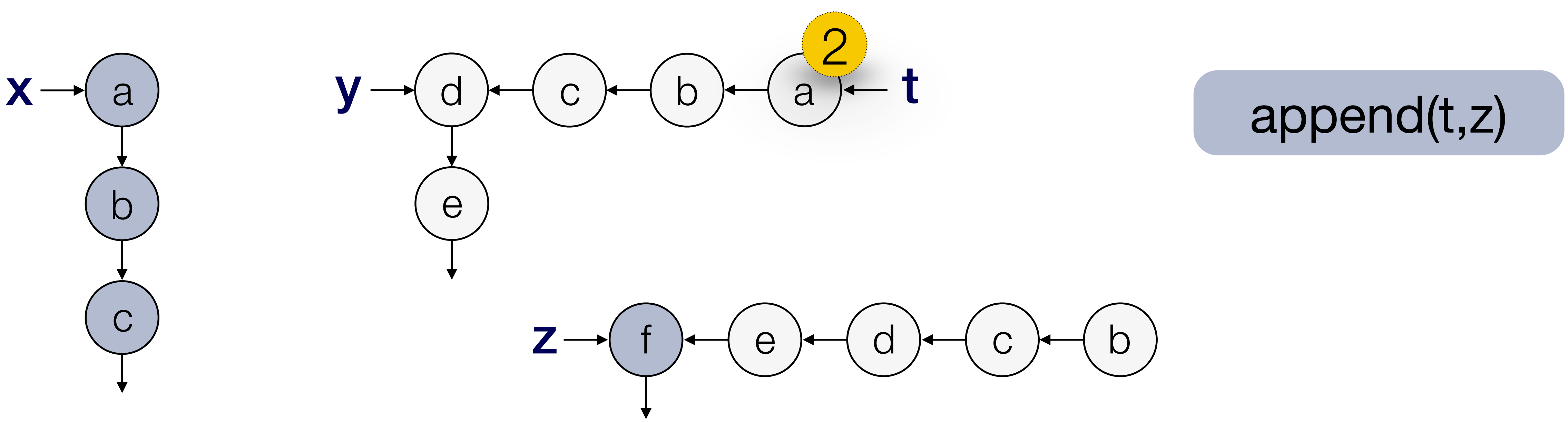
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



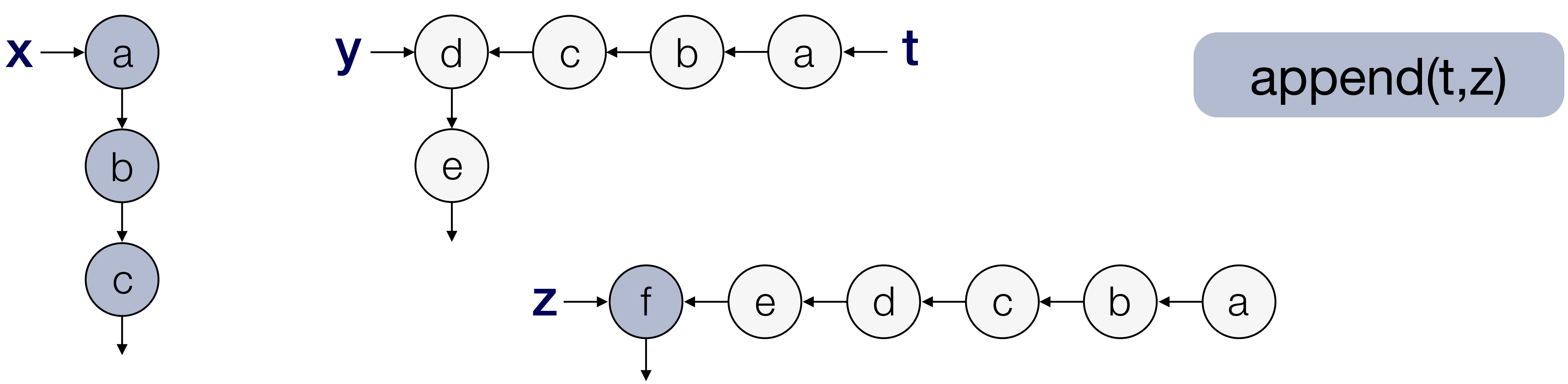
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



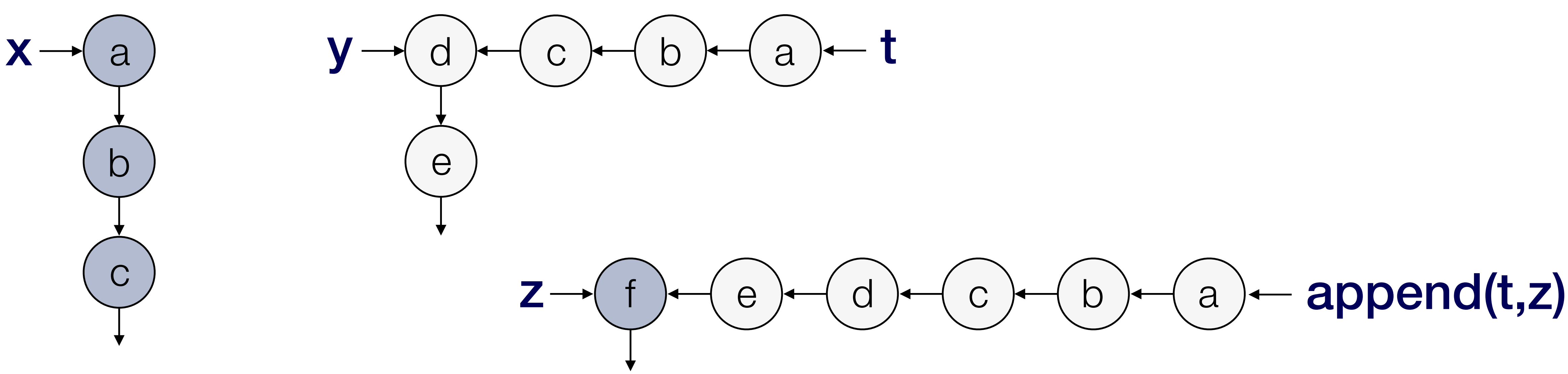
Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y



Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

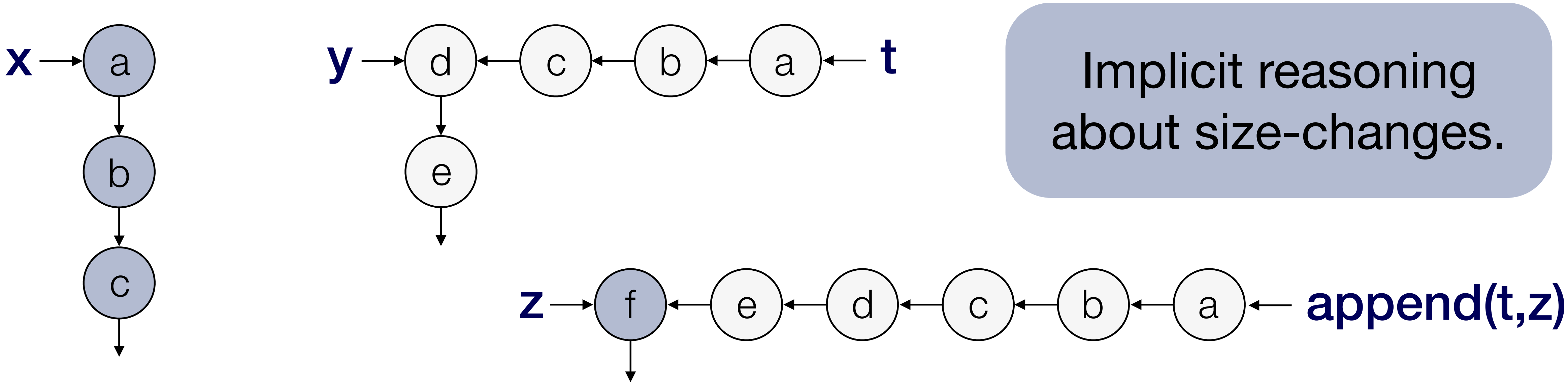
Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

Heap usage of f(x,y,z) is 2n + 2(n+m) if

- n is the length of list x
- m is the length of list y

Implicit reasoning about size-changes.



Initial potential: 4*n + 2*m = 4*3 + 2*2 = 16

Example: Composing Calls of Append

```
f(x,y,z) = {
  t = append(x,y);
  append(t,z)
}
```

append: $(L^{4}(\mathtt{int}), L^{2}(\mathtt{int})) \xrightarrow{0/0} L^{2}(\mathtt{int})$

append: $(L^{2}(\mathtt{int}), L^{0}(\mathtt{int})) \xrightarrow{0/0} L^{0}(\mathtt{int})$

The most general type of append is specialized at call-sites:

append: $(L^{q}(\mathtt{int}), L^{p}(\mathtt{int})) \xrightarrow{s/t} L^{r}(\mathtt{int}) \mid \Phi$

Linear constraints.

Polynomial Potential Functions

	Linear Potential Functions
User-defined **resource metrics** (i.e., by tick(q) in the code)	✓
Naturally **compositional**: tracks size changes, types are specifications	✓
Bound inference by reduction to efficient **LP solving**	✓
Type derivations prove bounds with respect to the cost semantics	✓

Phd Thesis: Polynomial Potential Functions

	Linear Potential Functions	Multivariate Polynomial Potential Functions
User-defined **resource metrics** (i.e., by tick(q) in the code)	✓	✓
Naturally **compositional**: tracks size changes, types are specifications	✓	✓
Bound inference by reduction to efficient **LP solving**	✓	✓
Type derivations prove bounds with respect to the cost semantics	✓	✓

Phd Thesis: Polynomial Potential Functions

For example $m*n^2$.

	Linear Potential Functions	Multivariate Polynomial Potential Functions
User-defined **resource metrics** (i.e., by tick(q) in the code)	✓	✓
Naturally **compositional**: tracks size changes, types are specifications	✓	✓
Bound inference by reduction to efficient **LP solving**	✓	✓
Type derivations prove bounds with respect to the cost semantics	✓	✓

Phd Thesis: Polynomial Potential Functions

For example $m*n^2$.

	Linear Potential Functions	Multivariate Polynomial Potential Functions
User-defined **resource metrics** (i.e., by tick(q) in the code)	✓	✓
Naturally **compositional**: tracks size changes, types are specifications	✓	✓
Bound inference by reduction to efficient **LP solving**	✓	✓
Type derivations prove bounds with respect to the cost semantics	✓	✓

Phd Thesis: Polynomial Potential Functions

First automatic, type-based resource analysis for polynomial bounds.

Example: Polynomial Potential Functions

```
t = append(xs,ys);
quicksort(t)
```

Computed time bound:

$$12m^2 + 24mn + 12n^2 + 14m + 22n + 11$$

Example: Polynomial Potential Functions

```
t = append(xs,ys);
quicksort(t)
```

Computed time bound:

$$12m^2 + 24mn + 12n^2 + 14m + 22n + 11$$

$$\mathit{append} : ((L(\mathit{int}), L(\mathit{int})), \begin{pmatrix} 6 & 26 & 24 \\ 34 & 24 & \\ 24 & & \end{pmatrix}) \to (L(\mathit{int}), (3, 26, 24))$$

$$\mathit{quicksort} : (L(\mathit{int}), (3, 26, 24)) \to (L(\mathit{int}), (0, 0, 0))$$

Example: Polynomial Potential Functions

```
t = append(xs,ys);
quicksort(t)
```

Computed time bound:

$$12m^2 + 24mn + 12n^2 + 14m + 22n + 11$$

$$append : ((L(int), L(int)), \begin{pmatrix} 6 & 26 & 24 \\ 34 & 24 & \\ 24 & & \end{pmatrix}) \to (L(int), (3, 26, 24))$$

$$6 + 34n + 26m + 24\binom{n}{2} + 24nm + 24\binom{m}{2}$$

$$quicksort : (L(int), (3, 26, 24)) \to (L(int), (0, 0, 0))$$

Example: Polynomial Potential Functions

```
t = append(xs,ys);
quicksort(t)
```

Computed time bound:

$$12m^2 + 24mn + 12n^2 + 14m + 22n + 11$$

Cost of *append*

$8n + 3$

$$\mathit{append} : ((L(\mathit{int}), L(\mathit{int})), \begin{pmatrix} 6 & 26 & 24 \\ 34 & 24 & \\ 24 & & \end{pmatrix}) \to (L(\mathit{int}), (3, 26, 24))$$

$$6 + 34n + 26m + 24\binom{n}{2} + 24nm + 24\binom{m}{2}$$

$$\mathit{quicksort} : (L(\mathit{int}), (3, 26, 24)) \to (L(\mathit{int}), (0, 0, 0))$$

Example: Polynomial Potential Functions

```
t = append(xs,ys);
quicksort(t)
```

Computed time bound:

$$12m^2 + 24mn + 12n^2 + 14m + 22n + 11$$

Cost of *append*

$$8n + 3$$

$$3 + 26(n{+}m) + 24\binom{n{+}m}{2}$$

$$\mathit{append} : ((L(\mathit{int}), L(\mathit{int})), \begin{pmatrix} 6 & 26 & 24 \\ 34 & 24 & \\ 24 & & \end{pmatrix}) \to (L(\mathit{int}), (3, 26, 24))$$

$$6 + 34n + 26m + 24\binom{n}{2} + 24nm + 24\binom{m}{2}$$

$$\mathit{quicksort} : (L(\mathit{int}), (3, 26, 24)) \to (L(\mathit{int}), (0, 0, 0))$$

Multivariate Resource Polynomials [POPL'11]

Map data structures to non-negative rational numbers

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

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

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

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

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

Multivariate Resource Polynomials [POPL'11]

Map data structures to non-negative rational numbers

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

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

Important innovation: sigma-pi formula for data structures

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

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

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

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

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

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

Resource Polynomials: Examples

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

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

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

Resource Polynomials: Examples

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

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

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

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

Resource Polynomials: Examples

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

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

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

$$[[a^1_1, \ldots, a^1_{m_1}], \ldots, [a^n_1, \ldots a^n_{m_n}]] \mapsto 18\binom{n}{2} + 12n + 3 + \sum_{1 \le i < j \le n} 12m_i$$

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

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

   $$q_{0,0} + q_{1,1}nm + q_{1,0}n + q_{2,0}\binom{n}{2} + q_{0,1}m + q_{0,2}\binom{m}{2}$$

   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

Outline

- Motivation ✓
- How does automatic resource bound analysis work? ✓
- **How well does automatic resource bound analysis work? (Implementation and experiments)**
- What are the properties of the LP instances that we get?

Resource Aware ML

- Based on Inria's **OCaml** compiler
- ~**12,000 lines of code** (+ ~29,000 loc form the OCaml compiler)
- Currently we use Coin-Or's CLP C interface
- F**eatures:**
  - Higher-order functions and polymorphism
  - User defined inductive types
  - Parallel evaluation
  - Side effects
  - User defined (non-monotone) resource metrics
  - Upper and lower bounds

Resource Aware ML

Web interface at
http://raml.co

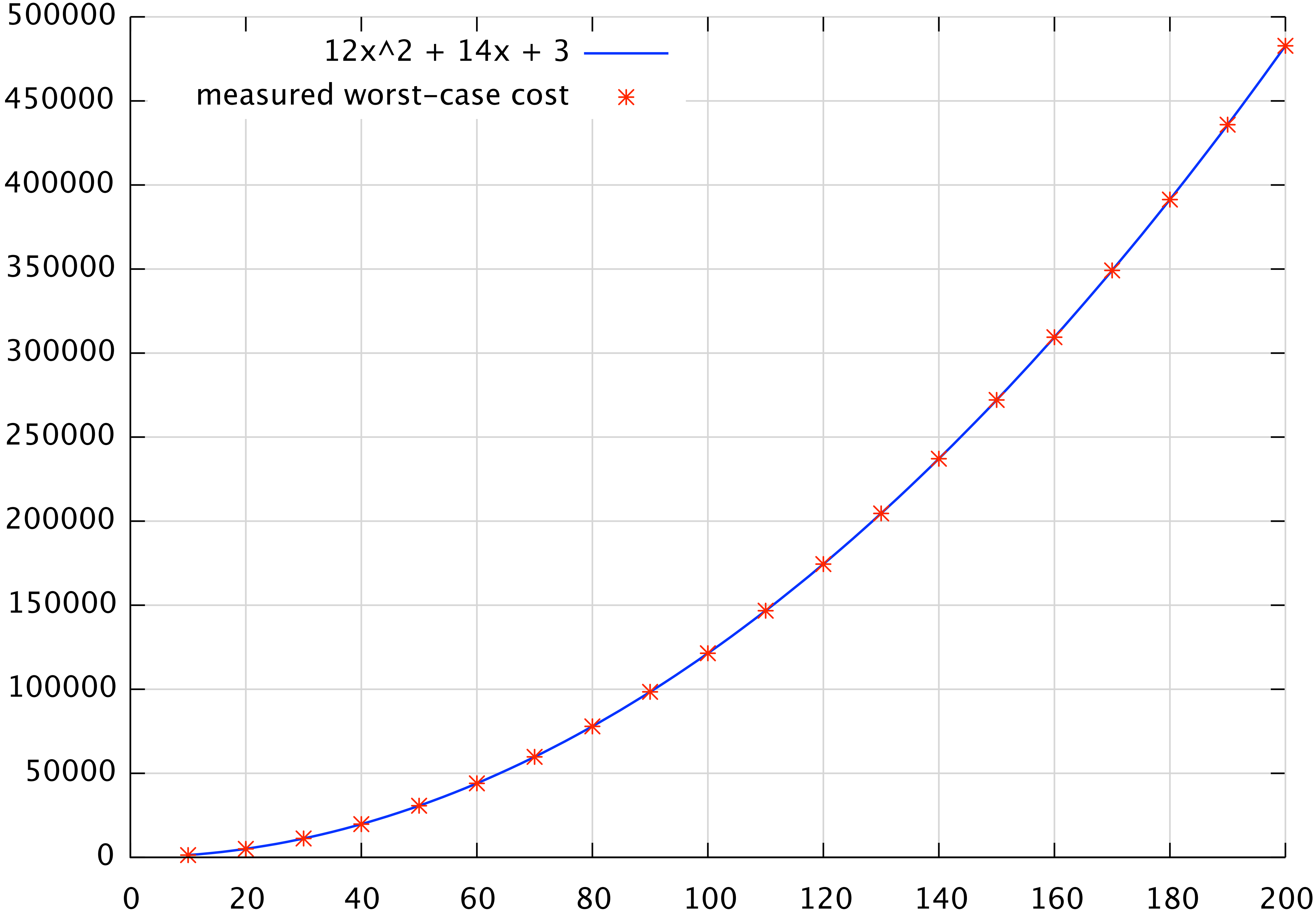
- Based on Inria's **OCaml** compiler
- ~**12,000 lines of code** (+ ~29,000 loc form the OCaml compiler)
- Currently we use Coin-Or's CLP C interface
- F**eatures:**
  - Higher-order functions and polymorphism
  - User defined inductive types
  - Parallel evaluation
  - Side effects
  - User defined (non-monotone) resource metrics
  - Upper and lower bounds

Experimental Evaluation

	Computed Bound	Actual Behavior	Performance
Quick Sort (Integers)	$12n^2 + 14n + 3$	$O(n^2)$	0.1 s
Split and Sort	$16n^2 + 46n + 9$	$O(n^2)$	2.1 s
Insertion Sort (Strings)	$8n^2m + 8n^2 - 8nm + 4n + 3$	$O(n^2m)$	0.91 s
Duplicate Elimination	$6n^2m + 9n^2 - 6nm + 3n + 3$	$O(n^2m)$	0.97 s
Longest Common Subsequence	$39nm + 6m + 21n + 19$	$O(nm)$	0.36 s
Matrix Multiplication	$28xmn + 32xm + 2x + 14n + 21$	$O(xmn)$	1.96 s
Breadth-First Matrix Multiplication	$2yz + 15ynmx + 14ynm + 15yn + 104n + 51$	$O(ynmx)$	4.98 s
Dijkstra's Shortest-Path Algorithm	$79.5n^2 + 31.5n + 38$	$O(n^2)$	2.50 s
In-Place Quick Sort for Arrays	$12.25x^2 + 52.75x + 3$	$O(x^2)$	0.64 s

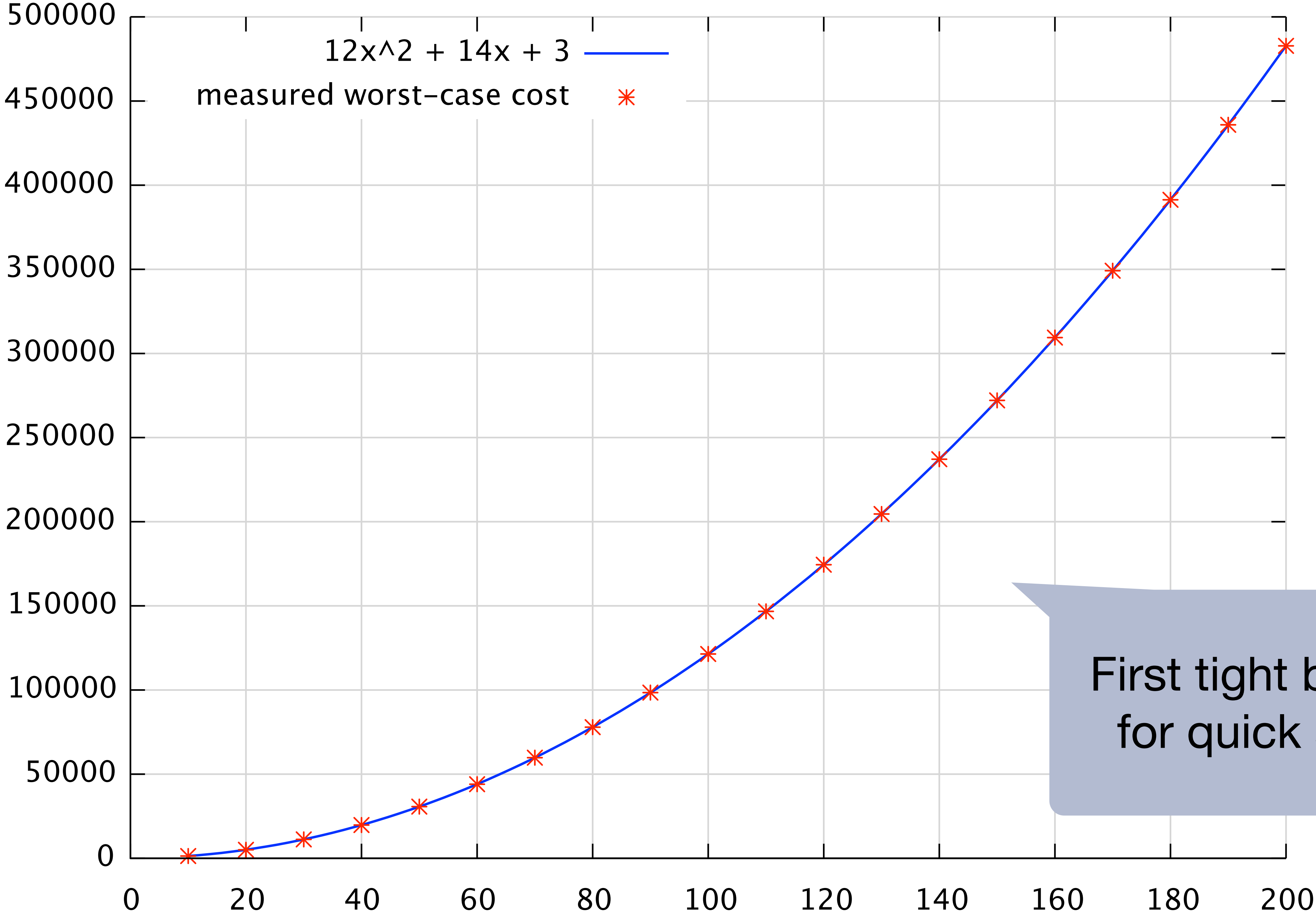
Micro Benchmarks

Evaluation-Step Bounds



Quick Sort for Integers

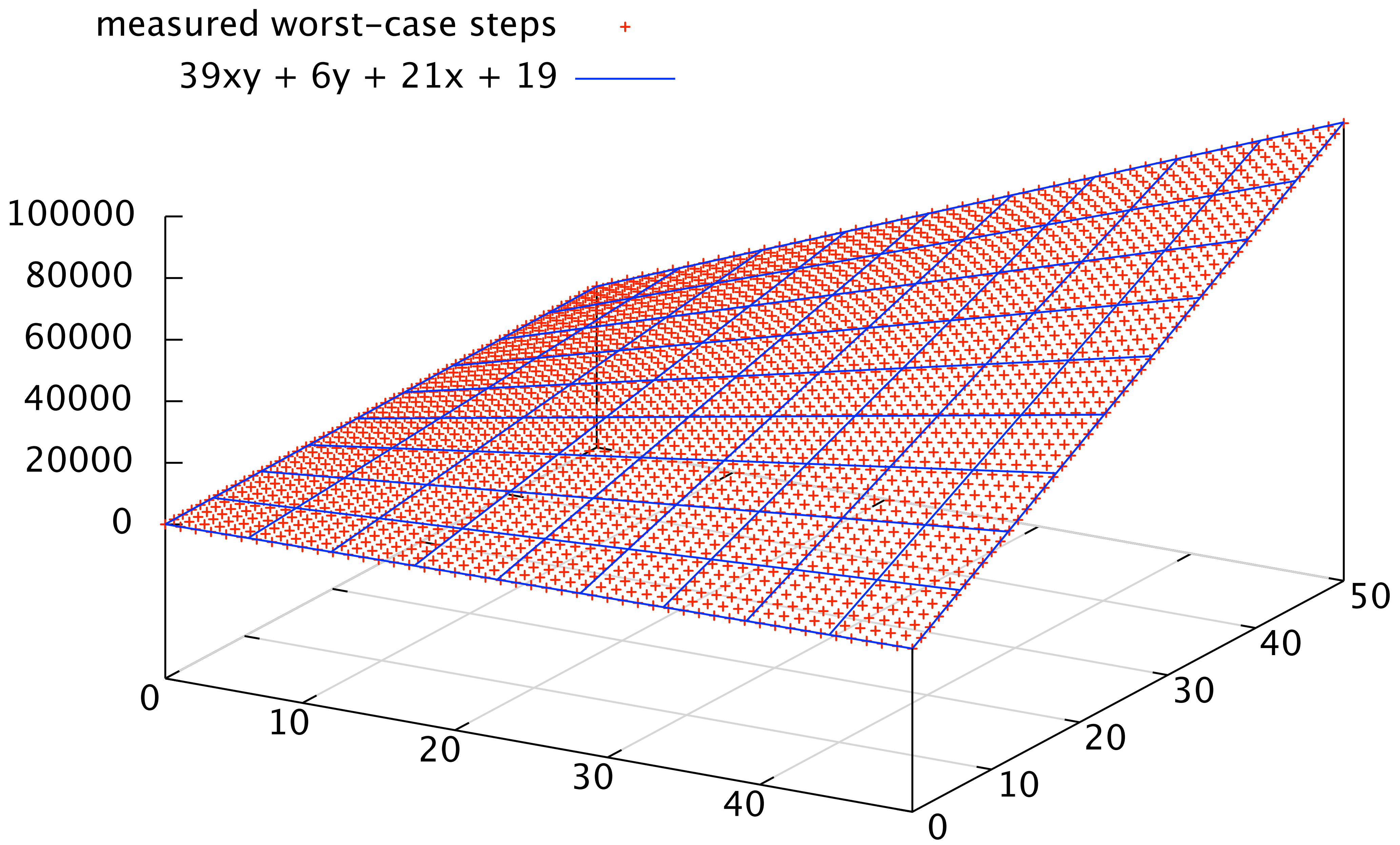
Evaluation-step bound vs. measured behavior



First tight bound for quick sort.

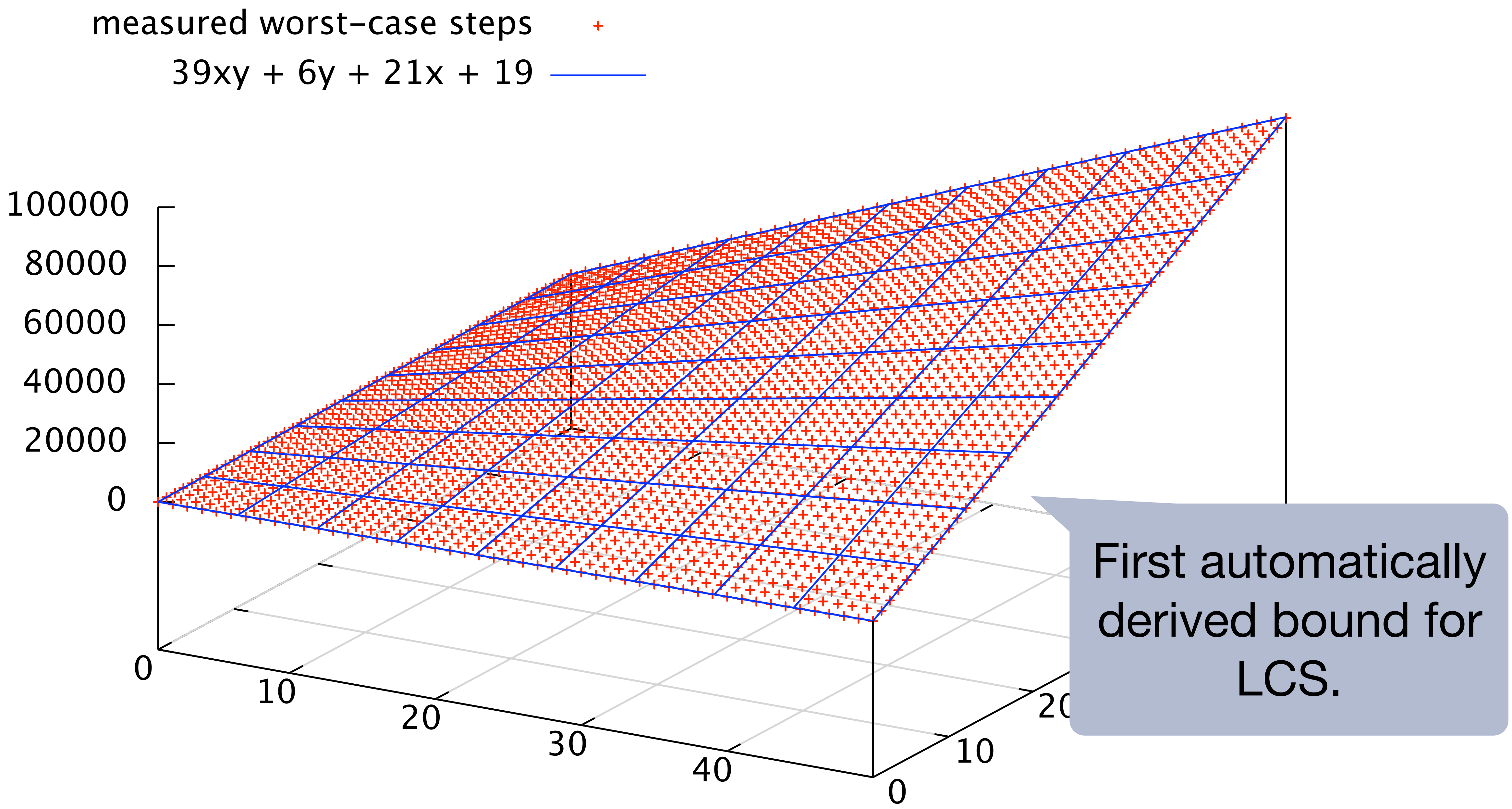
Quick Sort for Integers

Evaluation-step bound vs. measured behavior



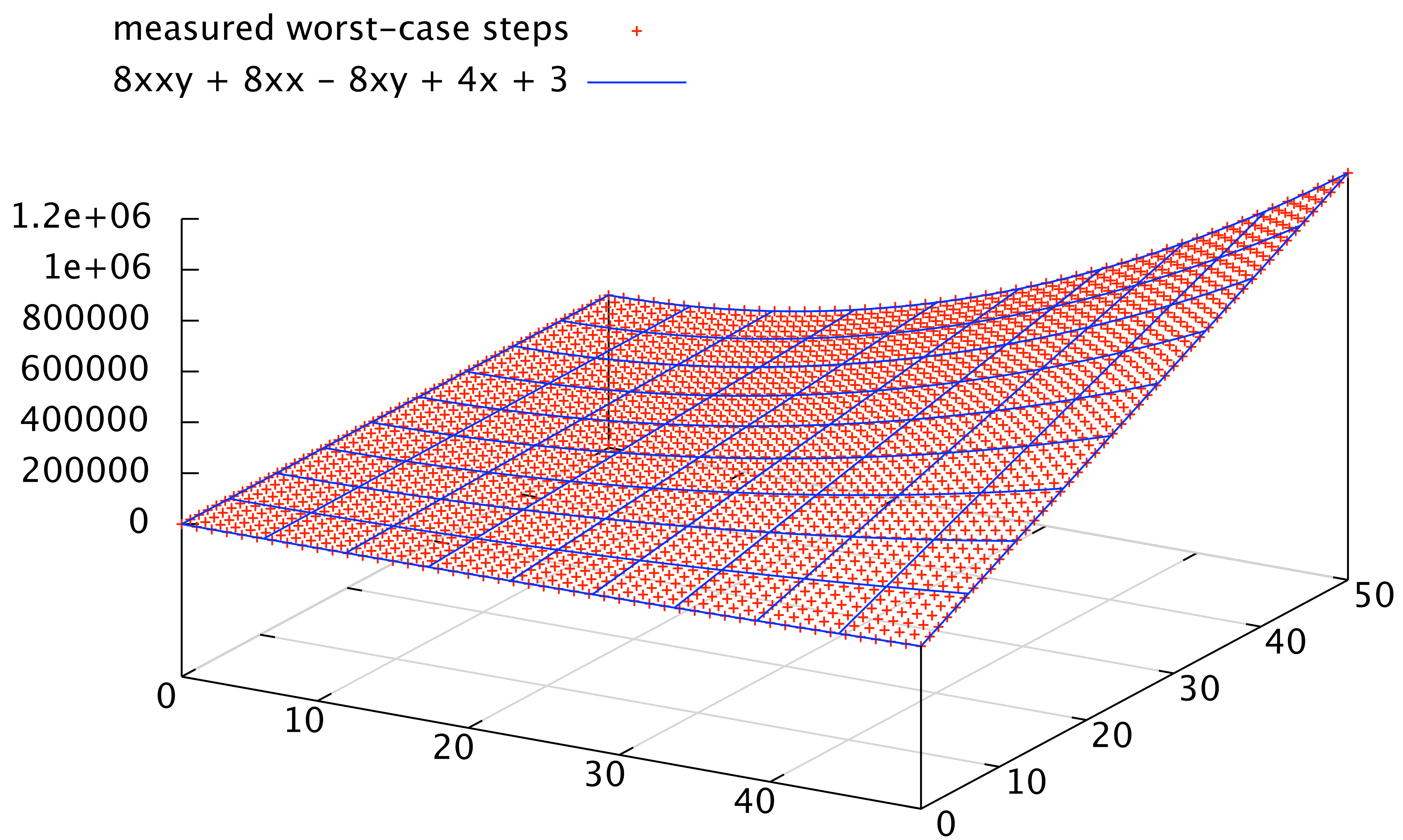
Longest Common Subsequence

Evaluation-step bound vs. measured behavior



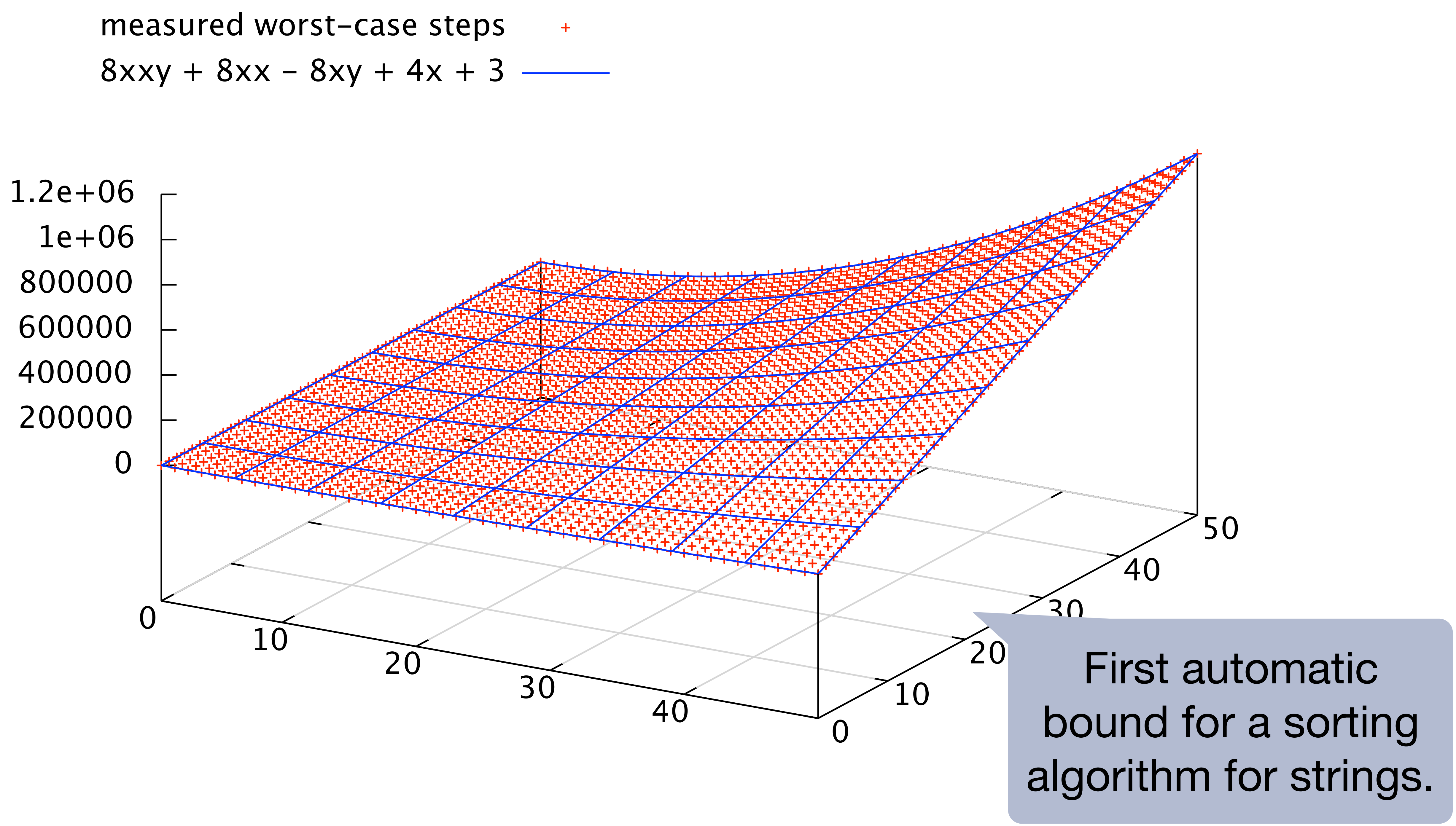
Longest Common Subsequence

Evaluation-step bound vs. measured behavior



Insertion Sort for Strings

Evaluation-step bound vs. measured behavior



Insertion Sort for Strings

Evaluation-step bound vs. measured behavior

Macro Benchmarks

1) OCaml's standard list library list.ml

- Evaluation-step bounds for 47 of 51 top-level functions
- 428 lines of code; 3.2 seconds on a Macbook Pro

2) CompCert C Compiler

- OCaml code extracted from the Coq specification
- Evaluation-step bounds for 13 topmost modules in the dependency graph
- 138 of 164 functions bounded; 2740 lines of code; 21min

Macro Benchmarks

1) OCaml's standard list library list.ml

- Evaluation-step bounds for 47 of 51 top-level functions
- 428 lines of code; 3.2 seconds on a Macbook Pro

2) CompCert C Compiler

- OCaml code extracted from the Coq specification
- Evaluation-step bounds for 13 topmost modules in the dependency graph
- 138 of 164 functions bounded; 2740 lines of code; 21min

Problems: Modules and untyped code.

Metric	#Funs	LOC	Time	#Const	#Lin	#Quad	#Cubic	#Poly	#Fail	Asym. Tight
steps	243	3218	72.10s	16	130	60	28	239	4	225
heap	243	3218	70.36s	41	112	60	22	239	4	225
tick	174	2144	64.68s	19	79	53	19	174	0	160

CompCert:

steps	164	2740	1300.91s	32	99	7	0	138	26	137

Macro Benchmarks

How can we make predictions about compiled code?

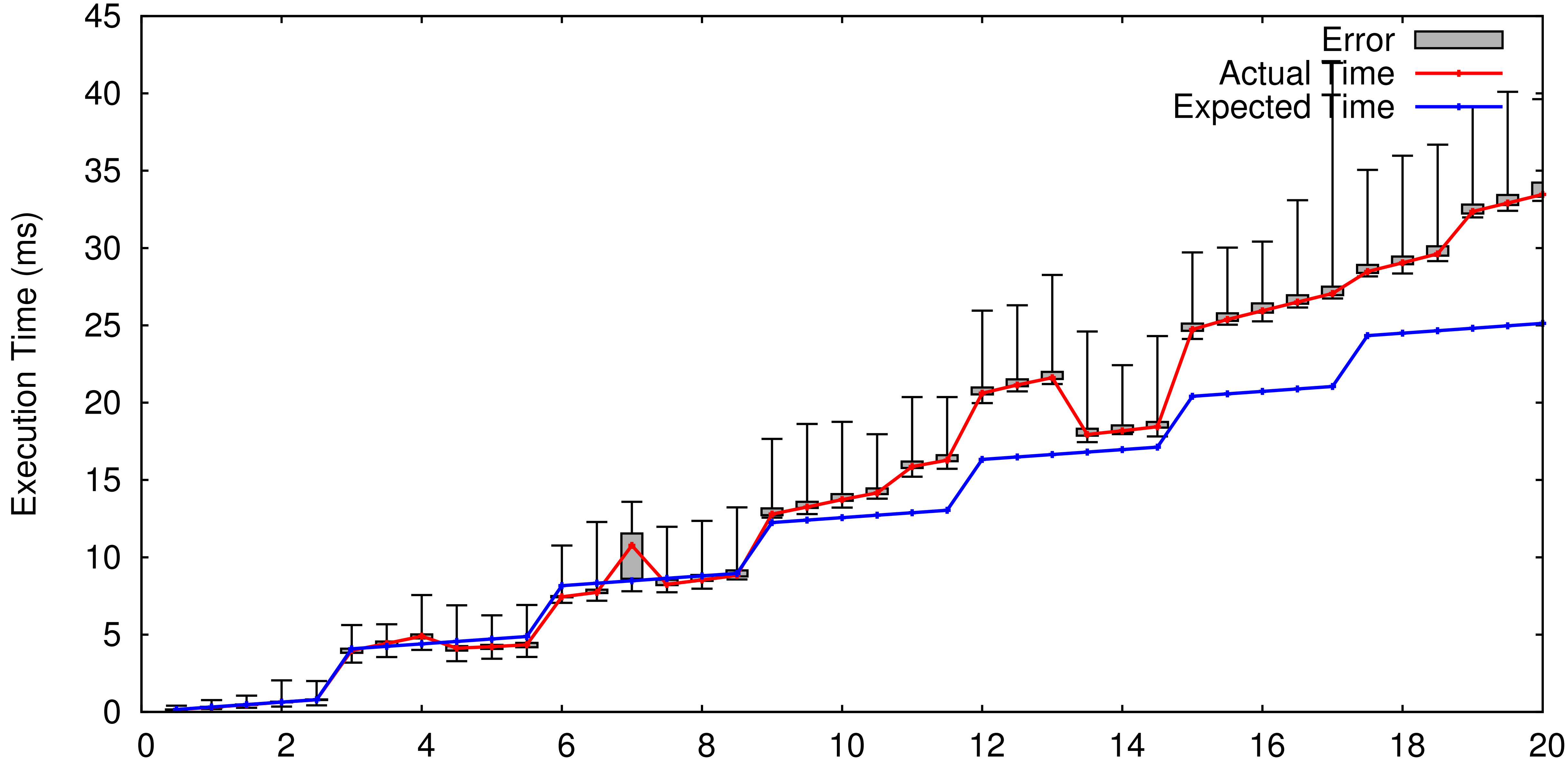
Machine Learning Cost Models

How to obtain realistic cost metrics for high-level analysis?

- Treat hardware, compiler, and runtime systems as black box
- Select training programs that cover relevant operations
- Use linear regression to obtain average time and memory costs of operations
- Combine time and memory predictions to get a time model for execution with garbage collection

Bound for List Append on x86

1.6 GHz Intel Core i5-5250U processor



Outline

- Motivation ✓
- How does automatic resource bound analysis work? ✓
- How well does automatic resource bound analysis work? ✓
  (Implementation and experiments)
- **What are the properties of the LP instances that we get?**

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

$$\sum_i x_i - \sum_j x_j = b$$

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

$$\sum_i x_i - \sum_j x_j = b$$

Inflow.

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

$$\sum_i x_i - \sum_j x_j = b$$

Inflow.

Outflow.

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

$$\sum_i x_i - \sum_j x_j = b$$

Constant sink/ source.

Inflow.

Outflow.

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

Constant sink/ source.

$$\sum_i x_i - \sum_j x_j = b$$

Inflow.

Outflow.

$$l_i \leq x_i \leq u_i$$

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

Constant sink/ source.

$$\sum_i x_i - \sum_j x_j = b$$

Inflow.

Outflow.

$$l_i \leq x_i \leq u_i$$

Flow capacity.

Linear Constraints have a Simple Form

- Pay for constant cost

$$q = q' + c$$

- Account for size changes

$$q'_i = q_i + q_{i+1}$$

- Recursive call

$$q = p$$

- Conditional branches

$$q \geq p_1 \qquad q \geq p_2$$

Linear Constraints have a Simple Form

- Pay for constant cost

$$q = q' + c$$

Network constraints.

- Account for size changes

$$q'_i = q_i + q_{i+1}$$

- Recursive call

$$q = p$$

- Conditional branches

$$q \geq p_1 \qquad q \geq p_2$$

Linear Constraints have a Simple Form

- Pay for constant cost

$$q = q' + c$$

Network constraints.

- Account for size changes

$$q'_i = q_i + q_{i+1}$$

Flow through an edge is used twice.

- Recursive call

$$q = p$$

- Conditional branches

$$q \geq p_1 \qquad q \geq p_2$$

Linear Constraints have a Simple Form

- Pay for constant cost

$$q = q' + c$$

Network constraints.

- Account for size changes

$$q'_i = q_i + q_{i+1}$$

Flow through an edge is used twice.

- Recursive call

$$q = p$$

Account for cost recursively.

- Conditional branches

$$q \geq p_1 \qquad q \geq p_2$$

Linear Constraints have a Simple Form

- Pay for constant cost

$$q = q' + c$$

Network constraints.

- Account for size changes

$$q'_i = q_i + q_{i+1}$$

Flow through an edge is used twice.

- Recursive call

$$q = p$$

Account for cost recursively.

- Conditional branches

$$q \geq p_1 \qquad q \geq p_2$$

Cover cost in both branches (possible waste).

Accounting for Size Change

List types:	$L^{(q_1,\ldots,q_k)}(A)$
Potential:	$\Phi(\ell : L^{(q_1,\ldots,q_k)}) = \sum_{i=1,\ldots,k} q_i \binom{\lvert\ell\rvert}{i}$
Additive shift:	$\lhd(q_1, \ldots, q_k) = (q_1 + q_2, \ldots, q_{k-1} + q_k, q_k)$

$$\boxed{\sum_{i=1,\ldots,k} q_i \binom{n+1}{i} = q_1 + \sum_{i=1,\ldots,k-1} q_{i+1} \binom{n}{i} + \sum_{i=1,\ldots,k} q_i \binom{n}{i}}$$

Accounting for Size Change

List types:	$L^{(q_1,\dots,q_k)}(A)$
Potential:	$\Phi(\ell : L^{(q_1,\dots,q_k)}) = \sum_{i=1,\dots,k} q_i \binom{\lvert\ell\rvert}{i}$
Additive shift:	$\lhd(q_1,\dots,q_k) = (q_1+q_2,\dots,q_{k-1}+q_k,q_k)$

potential of a list

$$\boxed{\sum_{i=1,\dots,k} q_i \binom{n+1}{i} = q_1 + \sum_{i=1,\dots,k-1} q_{i+1}\binom{n}{i} + \sum_{i=1,\dots,k} q_i \binom{n}{i}}$$

Accounting for Size Change

List types:	$L^{(q_1,\ldots,q_k)}(A)$
Potential:	$\Phi(\ell : L^{(q_1,\ldots,q_k)}) = \sum_{i=1,\ldots,k} q_i \binom{\lvert\ell\rvert}{i}$
Additive shift:	$\lhd(q_1, \ldots, q_k) = (q_1 + q_2, \ldots, q_{k-1} + q_k, q_k)$

$$\underbrace{\sum_{i=1,\ldots,k} q_i \binom{n+1}{i}}_{\text{potential of a list}} = \underbrace{q_1}_{\text{constant}} + \underbrace{\sum_{i=1,\ldots,k-1} q_{i+1} \binom{n}{i} + \sum_{i=1,\ldots,k} q_i \binom{n}{i}}_{\text{potential of the tail using the shift}}$$

Generation of Linear Constraints

1. It is easy to pass potential to list tails without loss

$$\boxed{\Phi((x{::}xs) : L^{\vec{q}}) + c = \Phi(xs : L^{\lhd(\vec{q})}) + (q_1 + c)}$$

2. Pattern: one recursive call and polynomial spill

$$\boxed{\Phi((x{::}xs){:}L^{\vec{q}}) - \Phi(xs{:}L^{\vec{q}}) = \Phi(xs{:}L^{(q_2,\dots,q_k,0)}) + q_1}$$

3. It is easy to share potential when aliasing data

$$\boxed{\Phi(\ell : L^{\vec{q}+\vec{p}}) = \Phi(\ell : L^{\vec{q}}) + \Phi(\ell : L^{\vec{p}})}$$

Generation of Linear Constraints

$$\triangleleft(q_1, \ldots, q_k) = (q_1 + q_2, \ldots, q_{k-1} + q_k, q_k)$$

1. It is easy to pass potential to list tails without loss

$$\boxed{\Phi((x{::}xs) : L^{\vec{q}}) + c = \Phi(xs : L^{\triangleleft(\vec{q})}) + (q_1 + c)}$$

2. Pattern: one recursive call and polynomial spill

$$\boxed{\Phi((x{::}xs){:}L^{\vec{q}}) - \Phi(xs{:}L^{\vec{q}}) = \Phi(xs{:}L^{(q_2,\ldots,q_k,0)}) + q_1}$$

3. It is easy to share potential when aliasing data

$$\boxed{\Phi(\ell : L^{\vec{q}+\vec{p}}) = \Phi(\ell : L^{\vec{q}}) + \Phi(\ell : L^{\vec{p}})}$$

Most Complex Constraints: Sharing

```
f(x) = g(x,x)
```

Assume cost of **g(x,y)** is $10|x| \cdot |y|$

Bound for **f** is given as $\sum_i q_i \binom{x}{i}$

Most Complex Constraints: Sharing

```
f(x) = g(x,x)
```

Assume cost of **g(x,y)** is $10|x|\cdot|y|$

Bound for **f** is given as $\displaystyle\sum_i q_i \binom{x}{i}$

Need to covert $q|x|^2$ to $\displaystyle\sum_i q_i \binom{x}{i}$

Most Complex Constraints: Sharing

```
f(x) = g(x,x)
```

Assume cost of **g(x,y)** is $10|x|\cdot|y|$

Bound for **f** is given as $\sum_i q_i \binom{x}{i}$

Need to covert $q|x|^2$ to $\sum_i q_i \binom{x}{i}$

Constraints:

$q_1 = 1 \cdot q$ $q_2 = 2 \cdot q$ $q_k = 0 \cdot q \text{ for } k > 2$

Most Complex Constraints: Sharing

```
f(x) = g(x,x)
```

Assume cost of **g(x,y)** is $10|x|\cdot|y|$

Bound for **f** is given as $\sum_i q_i \binom{x}{i}$

Need to covert $q|x|^2$ to $\sum_i q_i \binom{x}{i}$

Coefficients for change of basis.

Constraints:

$q_1 = 1 \cdot q$ $q_2 = 2 \cdot q$ $q_k = 0 \cdot q \text{ for } k > 2$

Constraint Solving in Practice

- LP solving of our constraints is linear in practice
- CLP and CPLEX are similar; lp_solve is slow (non-linear)
- Large programs (with high degree search space) have around 1 million constraints
- Solving 1 million constraints takes about 1 minute with CLP
- Generating the constraints takes about as much time as solving them

Automatic Amortized Resource Analysis

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

Current and future research:

- Non-polynomial bounds
- Garbage collection
- Concurrency
- Better hardware models

Automatic Amortized Resource Analysis

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

Current and future research:

- Non-polynomial bounds
- Garbage collection
- Concurrency
- Better hardware models

Web interface at
http://raml.co