

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

Timing



The image is a screenshot of a web article from EDN Network. The header includes the EDN Network logo and navigation links for Design Centers, Tools & Learning, Community, and ED. The article title is "Toyota's killer firmware: Bad design and its consequences" by Michael Dunn, dated October 28, 2013, with 109 comments. Below the title are social media sharing buttons for LinkedIn (277 shares), Google+ (932), Twitter (724), and Facebook (3.8k likes). There are also icons for print, PDF, and email. The article text begins with: "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.

Resource Usage in Safety-Critical Systems

Memory Usage



The image shows a screenshot of a web article from EDN Network. The article title is "Toyota's killer firmware: Bad design and its consequences" by Michael Dunn, dated October 28, 2013. It has 109 comments. The article text states: "On Thursday October 24, 2013, an Oklahoma court ruled against Toyota in a case of unintended acceleration that led to the death of one of the occupants. Central to the trial was the Engine Control Module's (ECM) firmware." The article includes social media sharing buttons for LinkedIn (277 shares), Google+ (932), Twitter (724), and Facebook (3.8k likes). There are also icons for printing, PDF, and email.

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

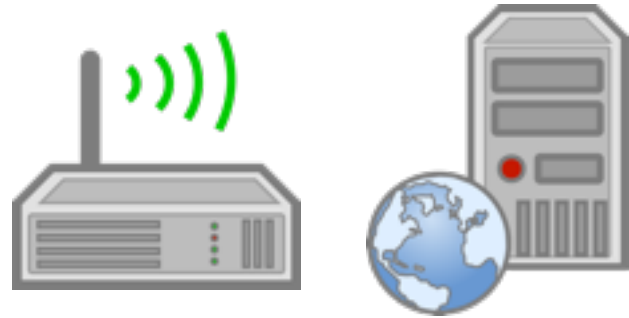


HealthCare.gov debacle has been mainly caused by performance issues.



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

Software Security

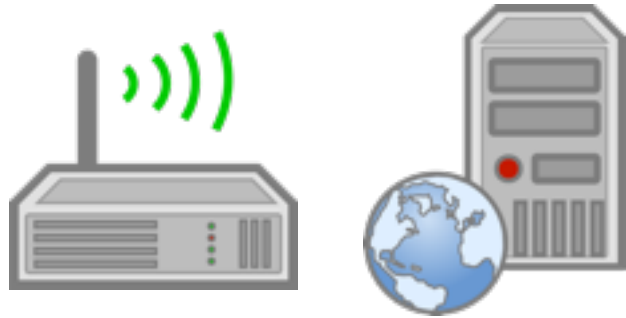


Algorithmic Complexity
Attacks



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

Given: A program P

Clock cycles, heap
space, power, ...

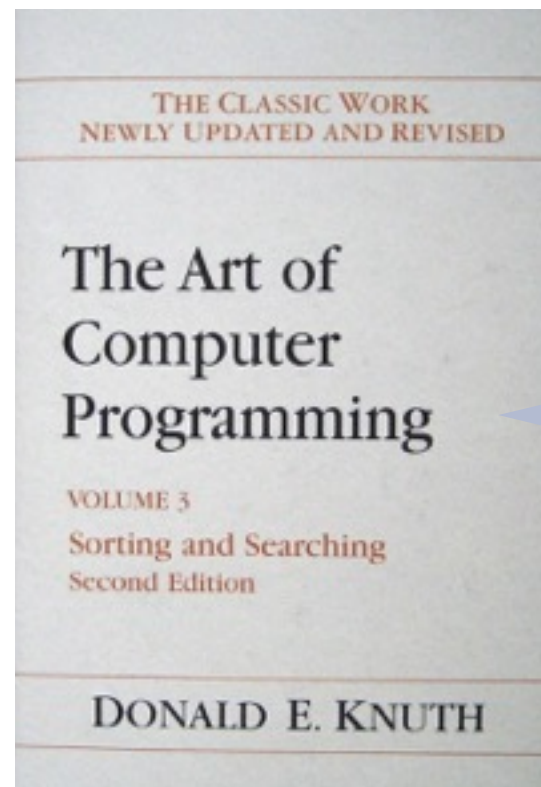
Question: What is the worst-case resource consumption of P as a function of the size of its inputs?

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?



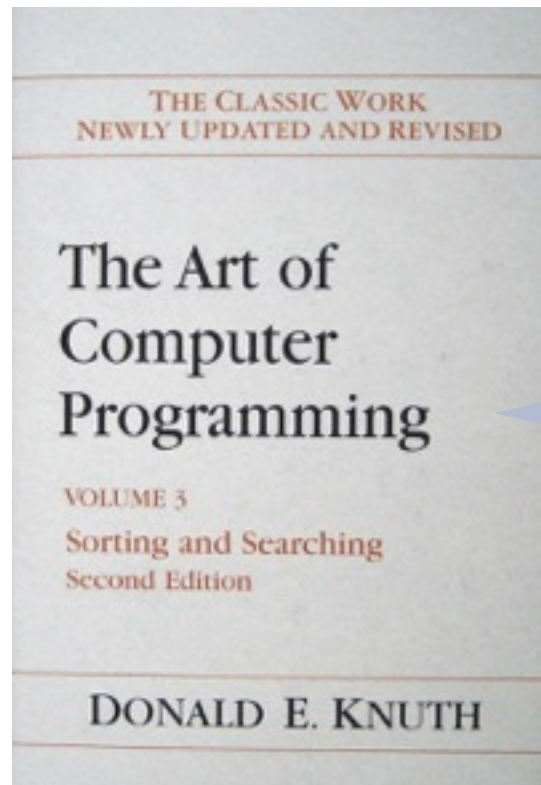
Not only asymptotic bounds but concrete constant factors.

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.

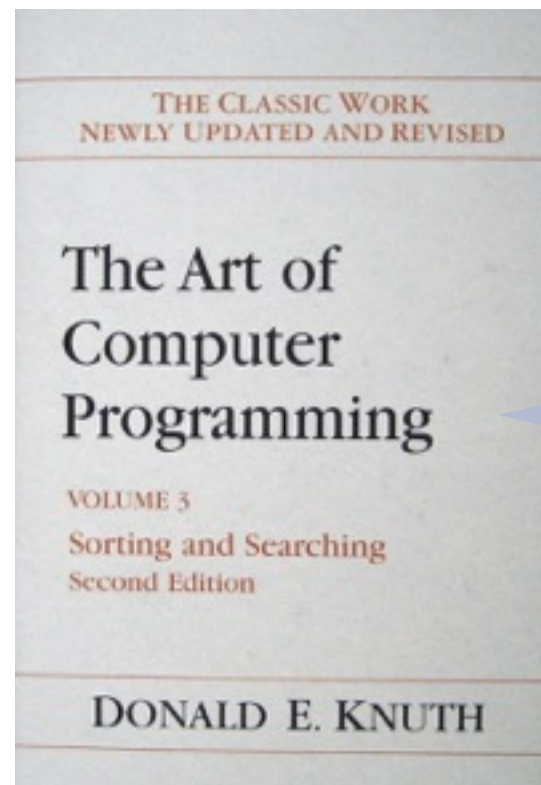
Automatic Static Resource Bound Analysis

Undecidable problem

Given: A program P

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.

Research Challenges

Model and predict resource usage at development time.



Compiler



Source code



Libraries



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?

```
def main(argv):
    """Main function"""
    # Parse command-line arguments
    parser = argparse.ArgumentParser()
    parser.add_argument('n', type=int, help='Number of elements')
    parser.add_argument('--verbose', action='store_true', help='Verbose output')
    args = parser.parse_args()

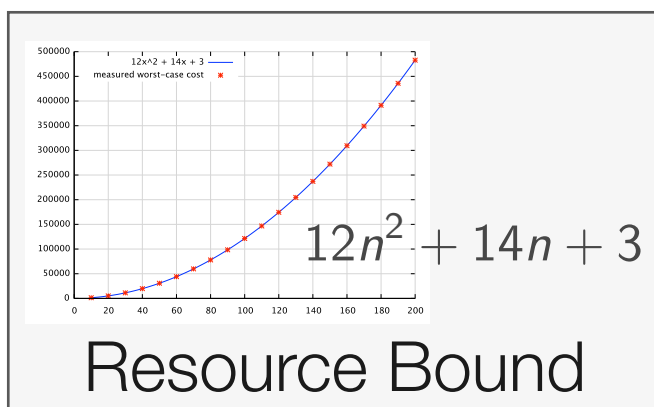
    # Calculate the sum of squares
    total = 0
    for i in range(1, args.n + 1):
        total += i * i

    # Print the result
    print('Sum of squares: %d' % total)

    # Exit
    sys.exit(0)

if __name__ == '__main__':
    main(sys.argv)
```

Source Code

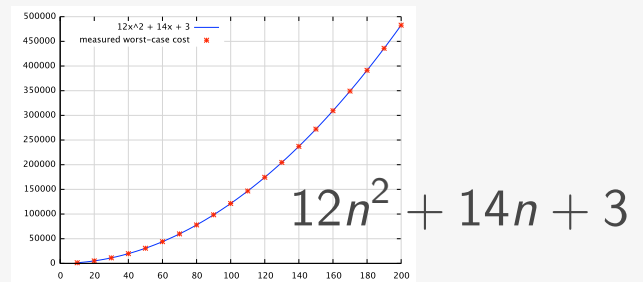


Bird's Eye View

Type-Based Resource Analysis

```
/* ... */
int main() {
    int n = 100;
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += i;
    }
    return sum;
}
/* ... */
```

Source Code



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

Resource Bound



applies to

```
section .text
global _start, write
write:
    mov     edi, 1 ; write syscall
    syscall

_start:
    mov     eax, 0x0407322669622F ; /bin/sh
    push   eax
    mov     ebx, eax
    mov     edi, 0
    mov     esi, 1
    mov     edx, 0
    call   write

exit: ; just exit not a function
    mov     eax, 0
    mov     ebx, 0
    syscall
```

Machine Code

Bird's Eye View

Type-Based Resource Analysis


```
/*...*/
int main() {
    int i;
    for (i = 0; i < 100; i++) {
        // ...
    }
    return 0;
}
```

Source Code

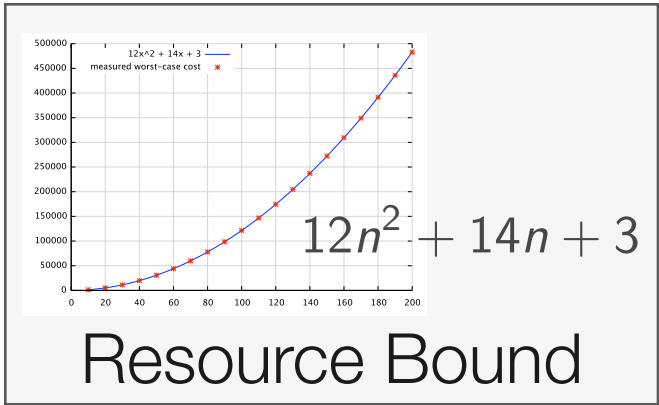
Compiler



Run-time system



Hardware



Bird's Eye View

applies to

```

.section .text
global _start, write
write:
    mov     edi, 1 ; write syscall
    syscall

_start:
    mov     eax, 0x04073226649622F ; /bin/sh
    push   eax
    mov     ebx, eax
    mov     edi, 1
    mov     ebx, 0
    call   write

exit: ; just exit not a function
    mov     eax, 0
    mov     ebx, 0
    syscall

```

Machine Code

Type-Based Resource Analysis

```

int main() {
    int x = 1;
    while (x < 100000000) {
        x = x + 1;
    }
}

```

Source Code

Formal Cost Semantics

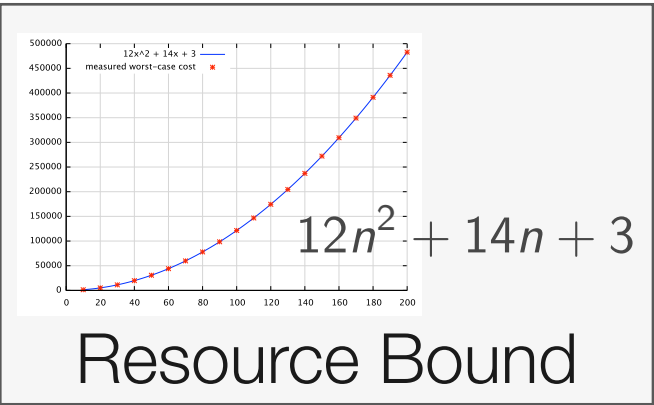
Compiler



Run-time system



Hardware



applies to

```

.section .text
global _start, write
write:
    mov     edi, 1 ; write syscall
    syscall

_start:
    mov     eax, 0x407322669622f ; /bin/sh
    push   eax
    mov     ebx, eax
    mov     edi, 1
    mov     edx, 8
    call   write

exit: ; just exit not a function
    mov     eax, 0
    syscall

```

Machine Code

Bird's Eye View

Type-Based Resource Analysis

```

// ...
void foo(int x) {
    int y = x * x;
    int z = y * y;
    int w = z * z;
    int v = w * w;
    int u = v * v;
    int t = u * u;
    int s = t * t;
    int r = s * s;
    int q = r * r;
    int p = q * q;
    int o = p * p;
    int n = o * o;
    int m = n * n;
    int l = m * m;
    int k = l * l;
    int j = k * k;
    int i = j * j;
    int h = i * i;
    int g = h * h;
    int f = g * g;
    int e = f * f;
    int d = e * e;
    int c = d * d;
    int b = c * c;
    int a = b * b;
}

```

Source Code

Formal Cost Semantics

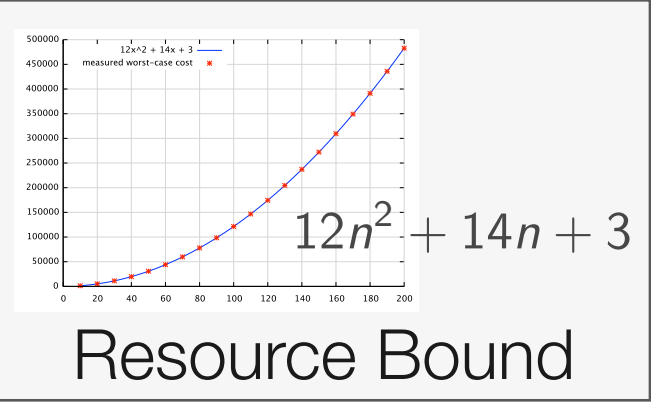
Compiler



Run-time system



Hardware



applies to

```

.section .text
global _start, write
write:
    mov     edi, 1 ; write syscall
    ret

_start:
    mov     eax, 0x407322669622f ; /bin/sh
    push   eax
    mov     ebx, eax
    mov     esi, esp
    mov     edi, 1
    mov     edx, 8
    call   write

    exit ; just exit not a function
    mov     eax, 0
    mov     ebx, 0
    syscall

```

Machine Code

Bird's Eye View

Type-Based Resource Analysis

```
let rec loop (n: int) (acc: int) : int =
  if n < 0 then acc
  else loop (n - 1) (acc + n)
let rec fact (n: int) : int =
  if n <= 1 then 1
  else n * fact (n - 1)
let rec fib (n: int) : int =
  if n <= 1 then n
  else fib (n - 1) + fib (n - 2)
let rec loop2 (n: int) (acc: int) : int =
  if n < 0 then acc
  else loop2 (n - 1) (acc + n * 2)
let rec fact2 (n: int) : int =
  if n <= 1 then 1
  else n * fact2 (n - 1)
let rec fib2 (n: int) : int =
  if n <= 1 then n
  else fib2 (n - 1) + fib2 (n - 2)
```

Source Code

IBM
CPLEX
COIN OR
CLP

Type Inference

Formal Cost Semantics

Compiler

AbsInt
aiT WCET Analyzers

$$\frac{}{acc:int \mid \frac{q_v}{q_v} \mid acc:int} \text{ (A:VAR)} \quad \frac{\Sigma(last) = (int, L^P \circ (int)) \xrightarrow{q_v \cdot L^P} int}{acc:int, x:int, xs:L^P(int) \mid \frac{q_a}{q_a} \mid last(x, xs) : int} \text{ (A:APP)}$$

$$\frac{}{acc:int, l:L^P(int) \mid \frac{q}{q} \mid match\ l\ with\ |nil \rightarrow acc \mid cons(x, xs) \rightarrow last(x, xs) : int} \text{ (A:MATL)}$$

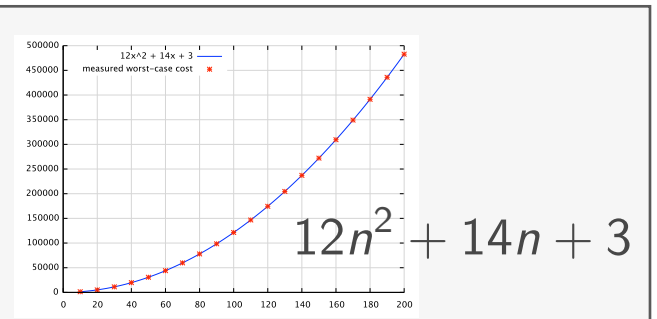
Type Derivation



Run-time system



Hardware



Resource Bound

applies to

```
section .text
global _start, write
write:
mov     edi, 1 ; write syscall
ret

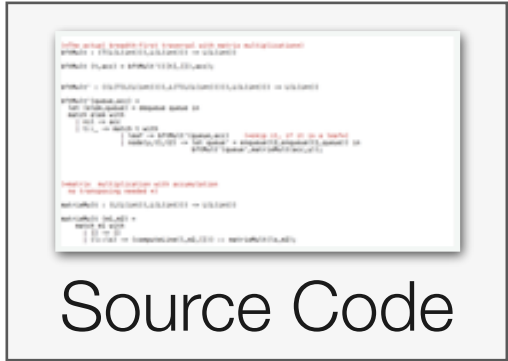
_start:
mov     eax, 0x004073220x69620f ; /bin/sh
push   esi
mov     esi, esp
mov     edi, 1
mov     ebx, 0
call   write

exit: ; Just exit not a function
mov     eax, 0
mov     ebx, 0
syscall
```

Machine Code

Bird's Eye View

Type-Based Resource Analysis



Source Code



Type Inference

Formal Cost Semantics

Compiler



Type Derivation

$$\frac{}{acc:int \mid \frac{q_v}{q_v} acc:int} \text{ (A:VAR)} \quad \frac{\Sigma(last) = (int, L^{P^2}(int)) \xrightarrow{q_v L^{q_v} int}}{acc:int, x:int, xs:L^P(int) \mid \frac{q_a}{q_a} last(x, xs) : int} \text{ (A:APP)}$$

$$\frac{}{acc:int, l:L^P(int) \mid \frac{q}{q} match\ l\ with\ nil \rightarrow acc \mid cons(x, xs) \rightarrow last(x, xs) : int} \text{ (A:MATL)}$$

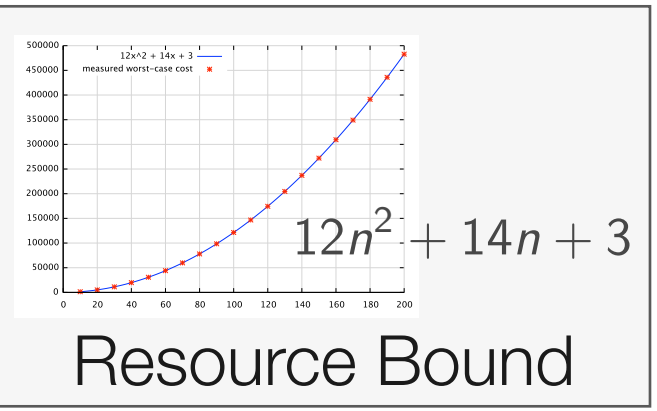

Run-time system



Hardware



applies to



```

section .text
global _start, write
write:
    mov     edi, 1 ; write syscall
    mov     eax, 0x407322669622f ; /bin/sh
    push   esi
    mov     esi, esp
    mov     edi, 1
    mov     ebx, 0
    call   write

exit: ; just exit not a function
    mov     eax, 0
    mov     ebx, 0
    syscall
    
```

Machine Code

Bird's Eye View

Type-Based Resource Analysis

```

def last(xs: List[A]): A = xs match {
  case Nil => throw new NoSuchElementException
  case _::xs => last(xs)
}

def cons(x: A, xs: List[A]): List[A] = x::xs

def matchWith[A, B](xs: List[A], f: A => B): List[B] = xs match {
  case Nil => Nil
  case x::xs => f(x)::matchWith(xs, f)
}

// Example: multiplication with recursion
// in following code:
// cons(1, cons(2, cons(3, cons(4, cons(5, Nil))))
// matchWith(cons(1, cons(2, cons(3, cons(4, cons(5, Nil)))))

// Example: multiplication with recursion
// in following code:
// cons(1, cons(2, cons(3, cons(4, cons(5, Nil))))
// matchWith(cons(1, cons(2, cons(3, cons(4, cons(5, Nil)))))

```

Source Code

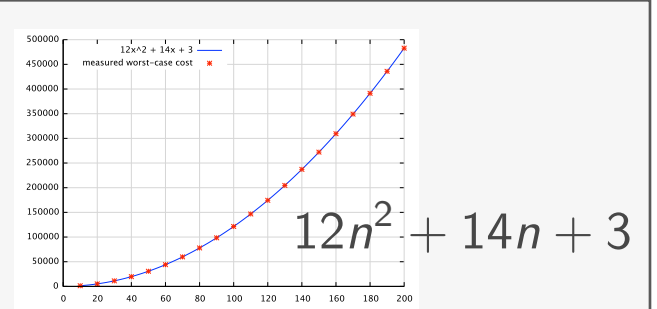


Type Inference

$$\frac{}{acc:int \mid \frac{q_v}{q_v} acc:int} \text{ (A:VAR)} \quad \frac{\Sigma(last) = (int, L^P(int)) \xrightarrow{q_x L^q_x} int}{acc:int, x:int, xs:L^P(int) \mid \frac{q_a}{q_a} last(x, xs) : int} \text{ (A:APP)}$$

$$\frac{}{acc:int, l:L^P(int) \mid \frac{q}{q} match\ l\ with\ nil \rightarrow acc \mid cons(x, xs) \rightarrow last(x, xs) : int} \text{ (A:MATL)}$$

Type Derivation



Resource Bound

Bird's Eye View

Type-Based Resource Analysis

```

def last(xs: List[A]): A = xs match {
  case Nil => ???
  case _::xs => last(xs)
}

```

Source Code



Type Inference

Clear soundness theorem.

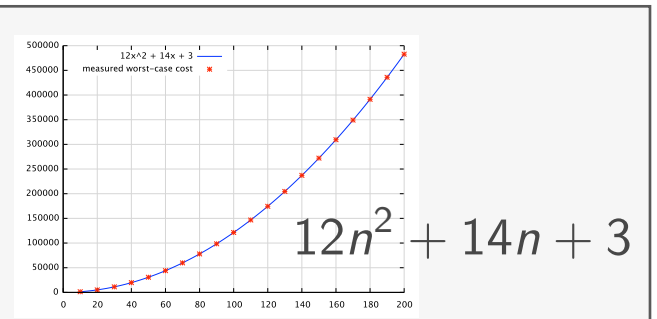
Naturally compositional.

Efficient inference via LP solving.

$$\frac{}{acc: int \mid \frac{q_v}{q_v} acc: int} \text{ (A:VAR)} \quad \frac{\Sigma(last) = (int, LP^z(int) \xrightarrow{q_v} int)}{acc: int, x: int, xs: LP^P(int) \mid \frac{q_a}{q_a} last(x, xs) : int} \text{ (A:APP)}$$

$$\frac{}{acc: int, l: LP^P(int) \mid \frac{q}{q} match\ l\ with\ nil \rightarrow acc \mid cons(x, xs) \rightarrow last(x, xs) : int} \text{ (A:MATL)}$$

Type Derivation



Resource Bound

Bird's Eye View

Type-Based Resource Analysis

Idea: Automate Amortized Analysis

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

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

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

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

↓ telescoping ↓

- Initial potential is an upper bound

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

Idea: Automate Amortized Analysis

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

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

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

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

↓ telescoping ↓

- Initial potential is an upper bound

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

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

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

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

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

↓ telescoping ↓

- Initial potential is an upper bound

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

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:

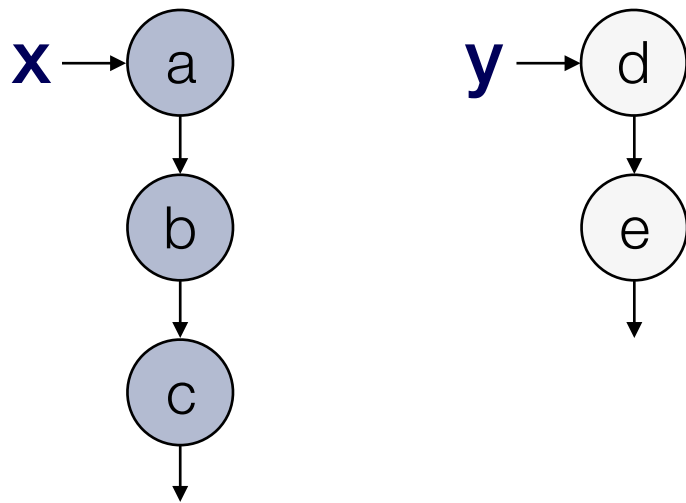
Example: Append for Persistent Lists

`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:



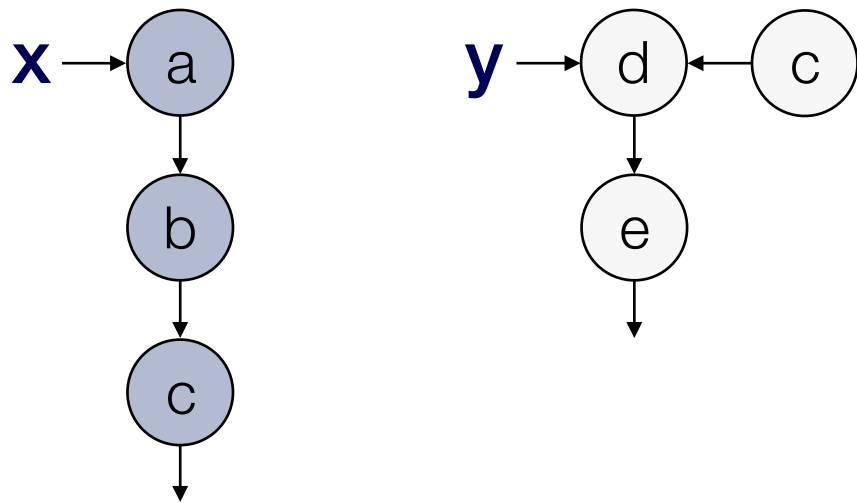
Example: Append for Persistent Lists

`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:



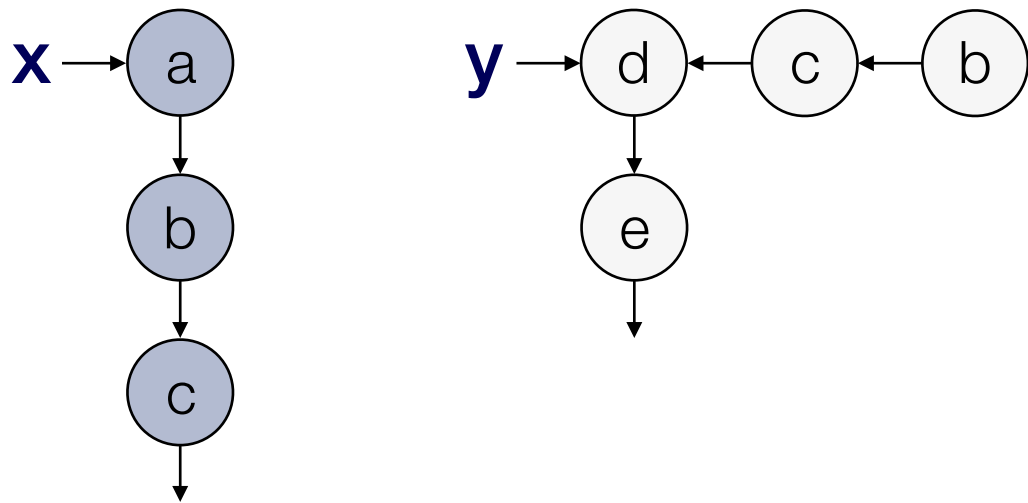
Example: Append for Persistent Lists

`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:



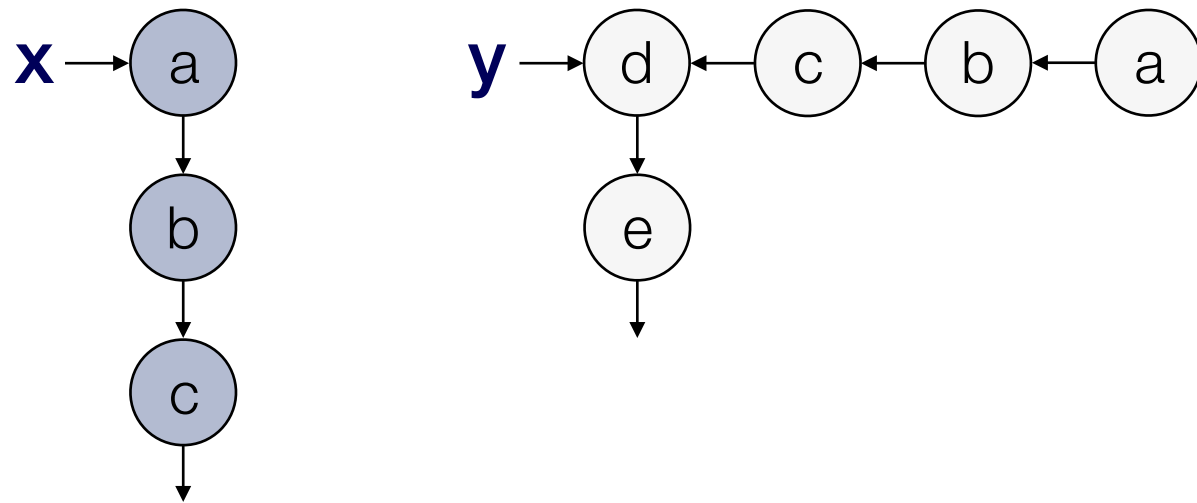
Example: Append for Persistent Lists

`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:



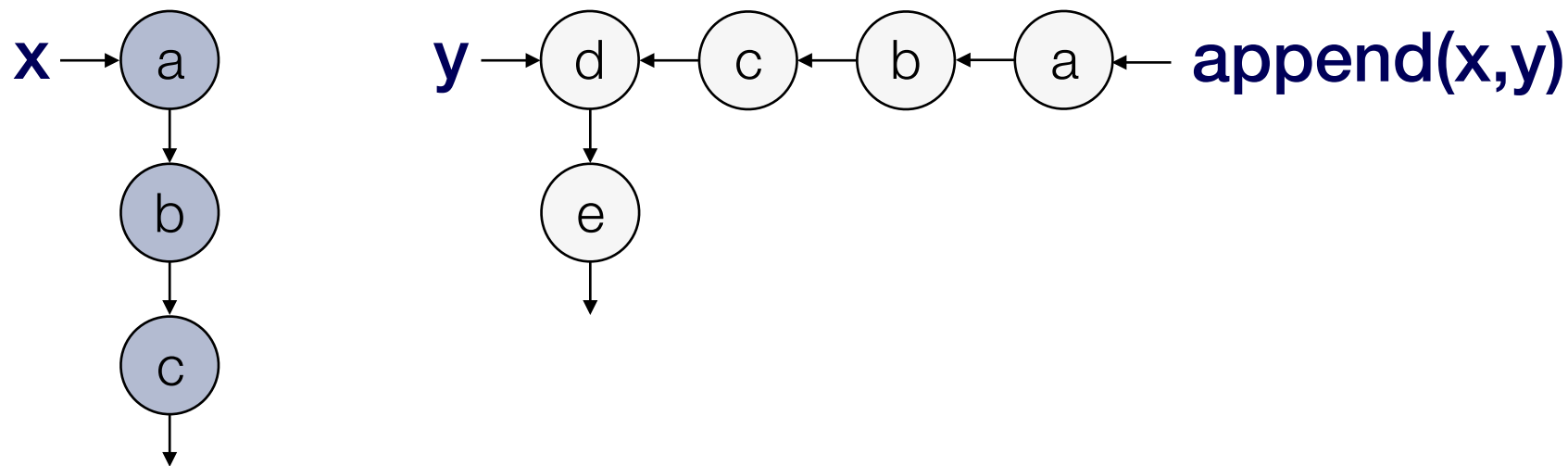
Example: Append for Persistent Lists

`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:



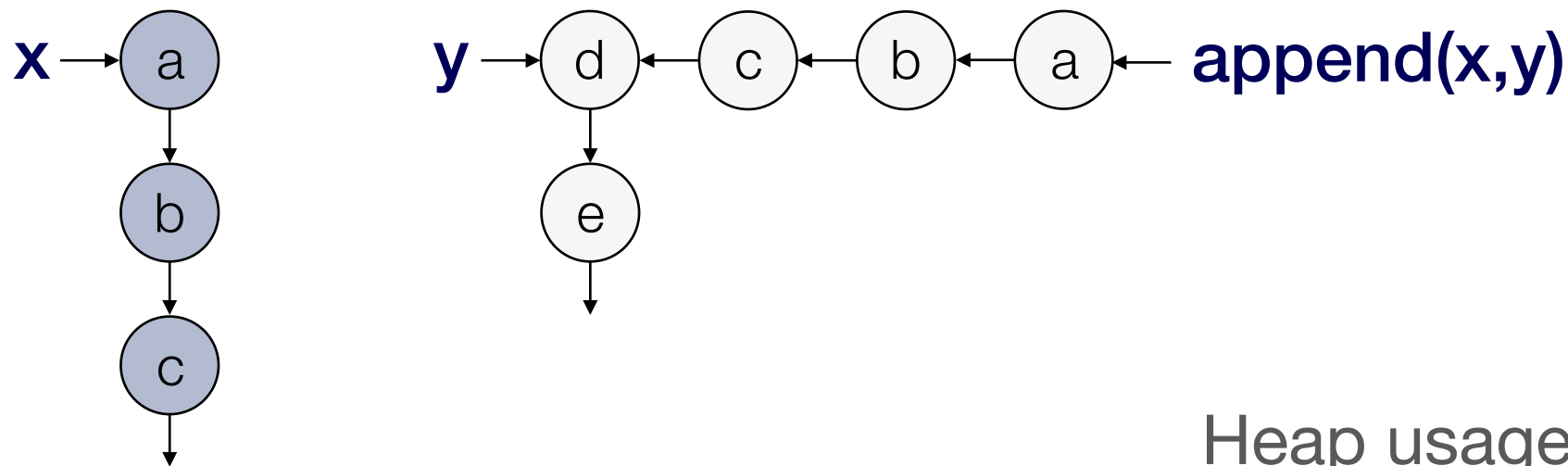
Example: Append for Persistent Lists

`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

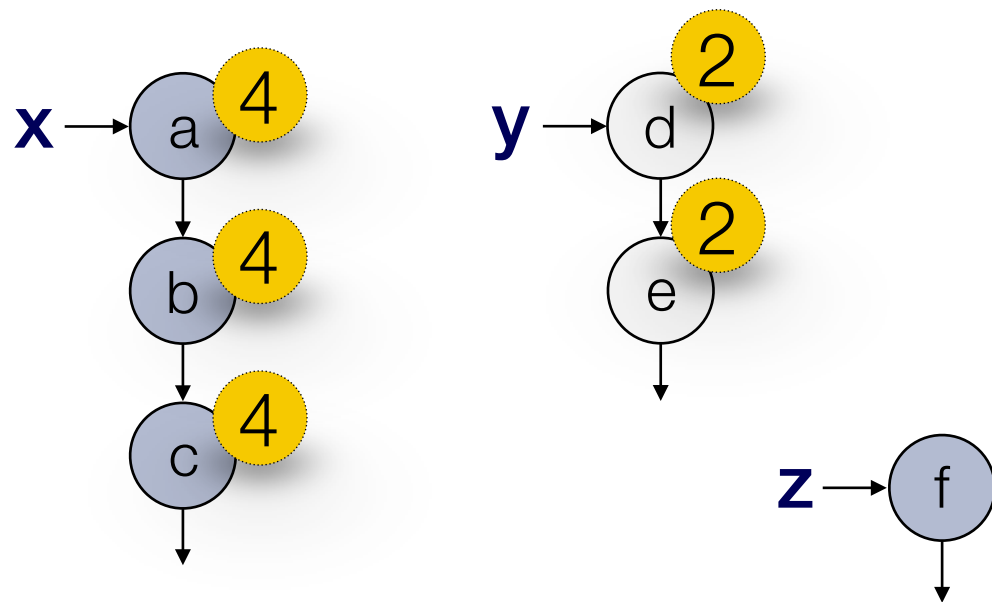
- n is the length of list x
- m is the length of list y

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



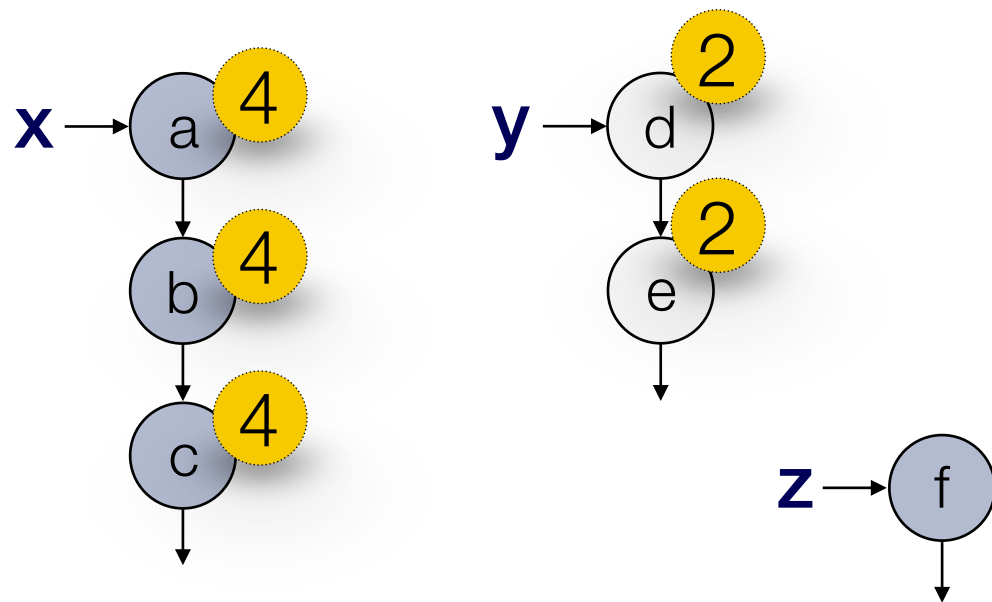
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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)

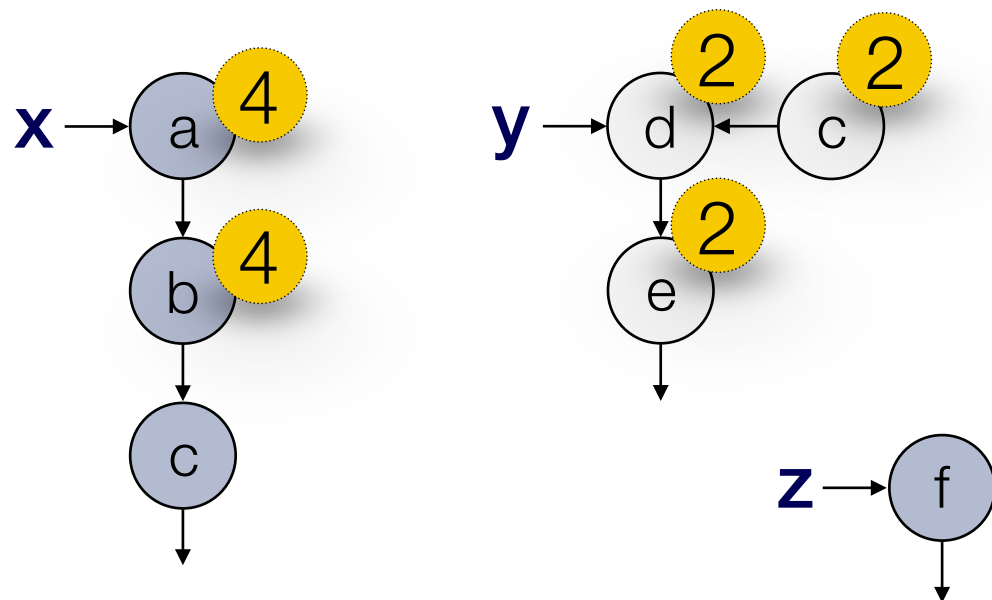
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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)

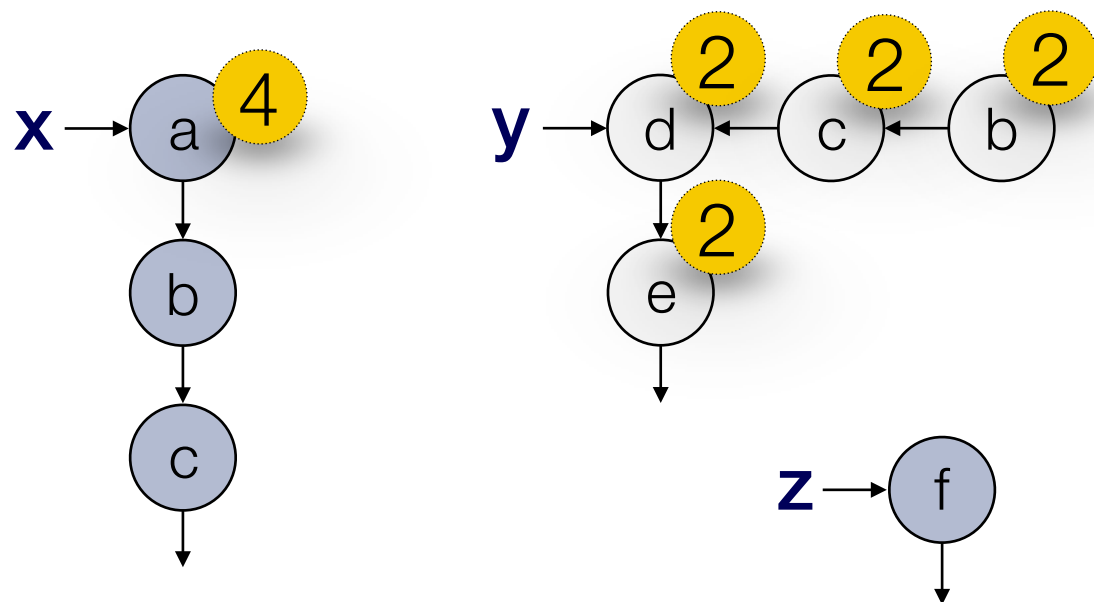
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



append(x,y)

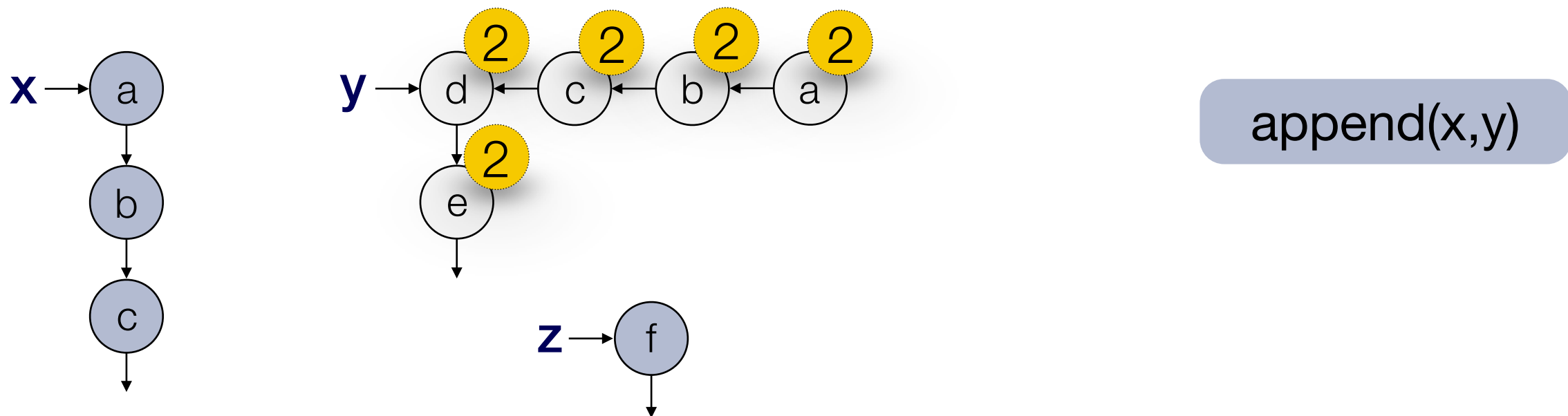
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



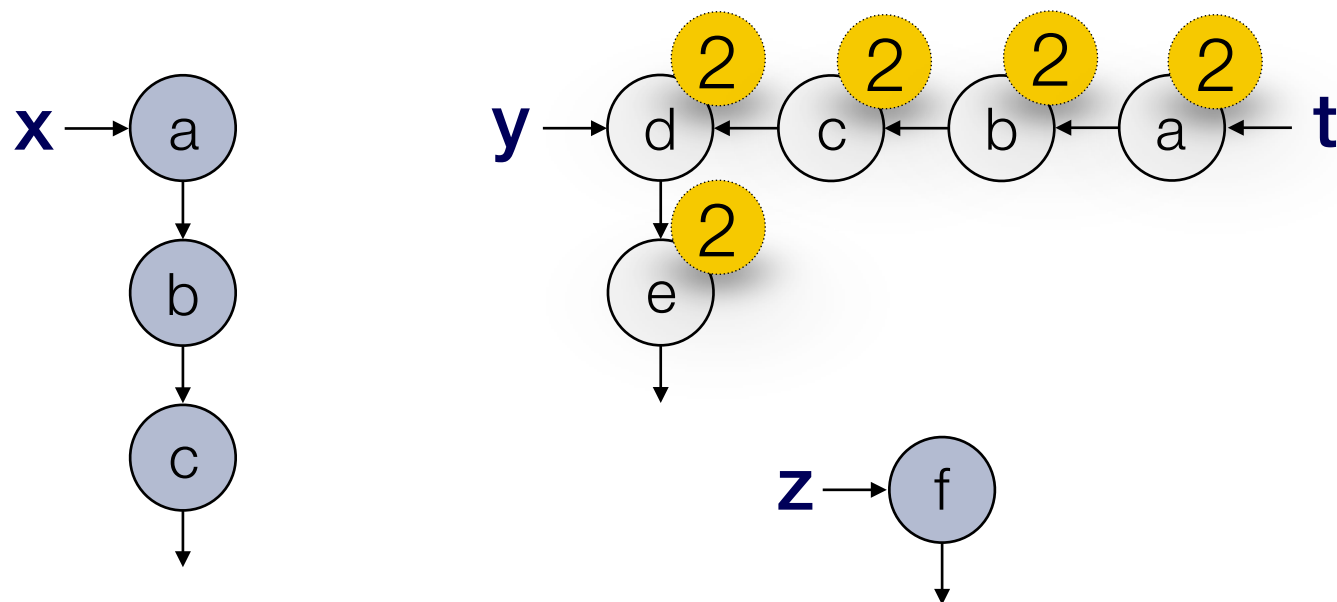
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



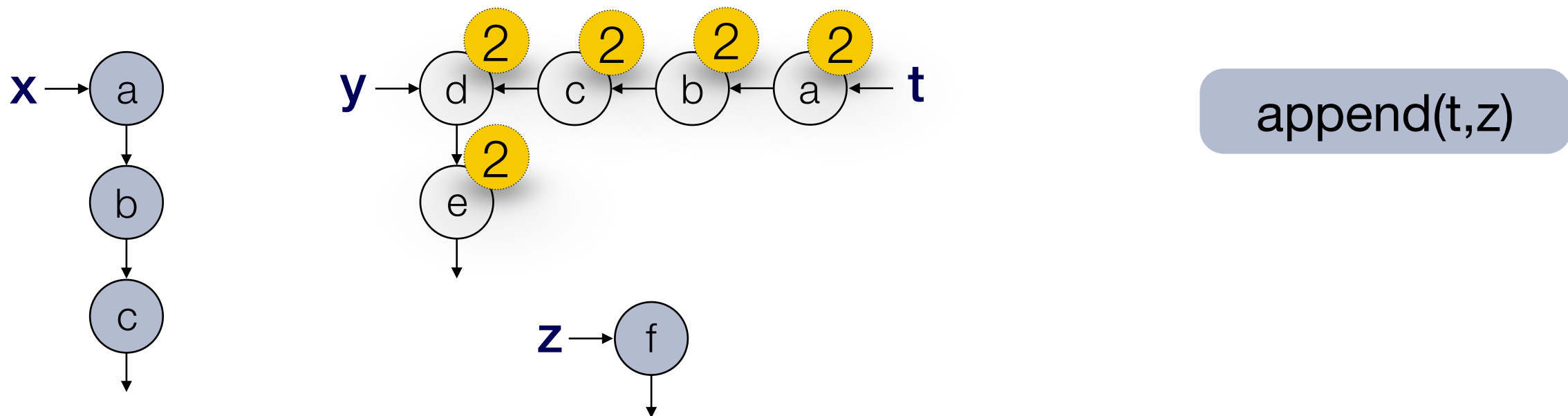
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



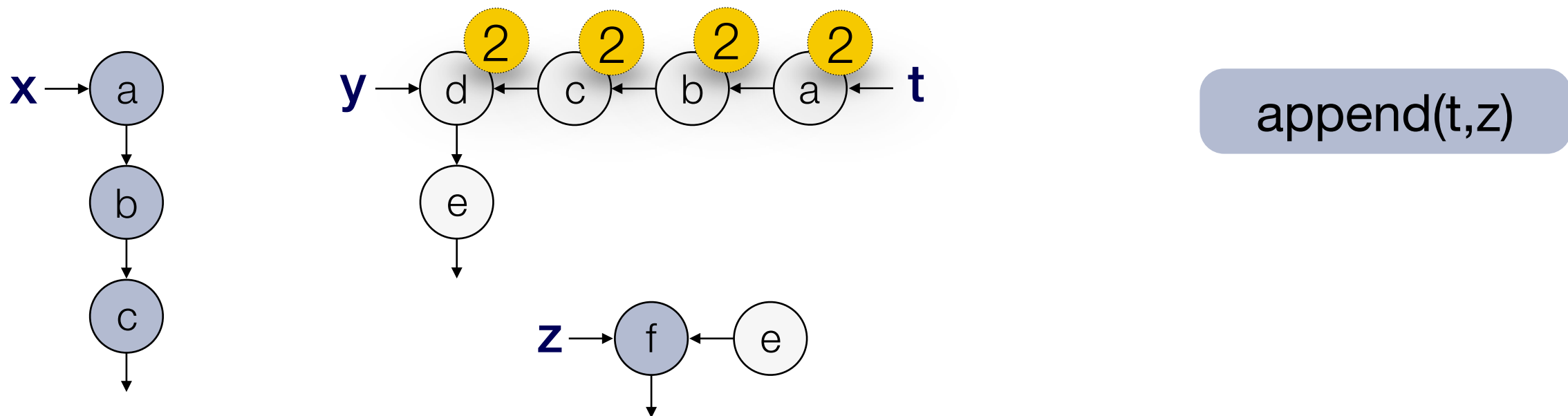
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



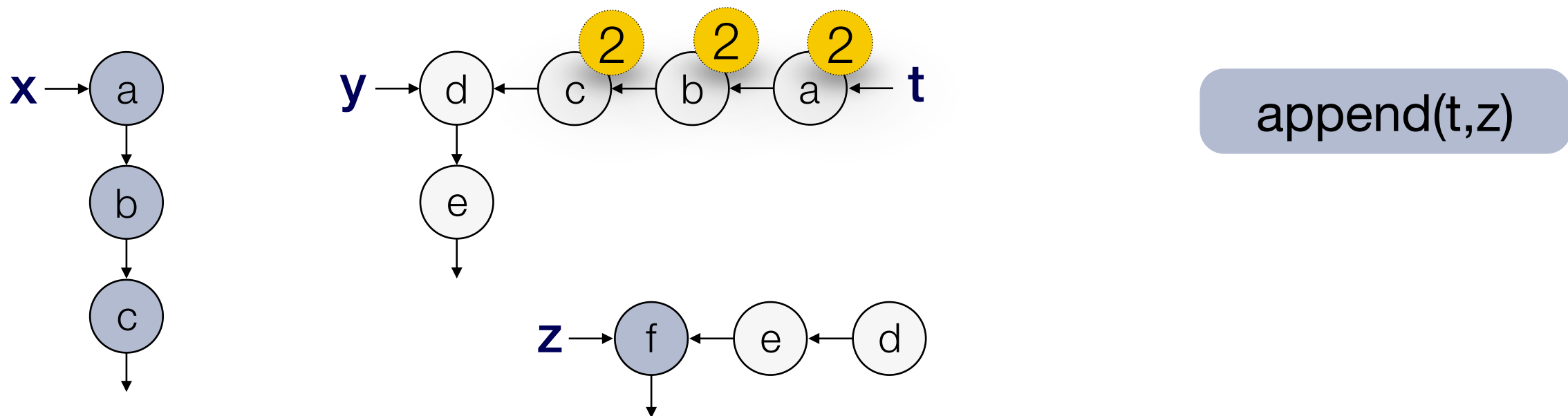
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



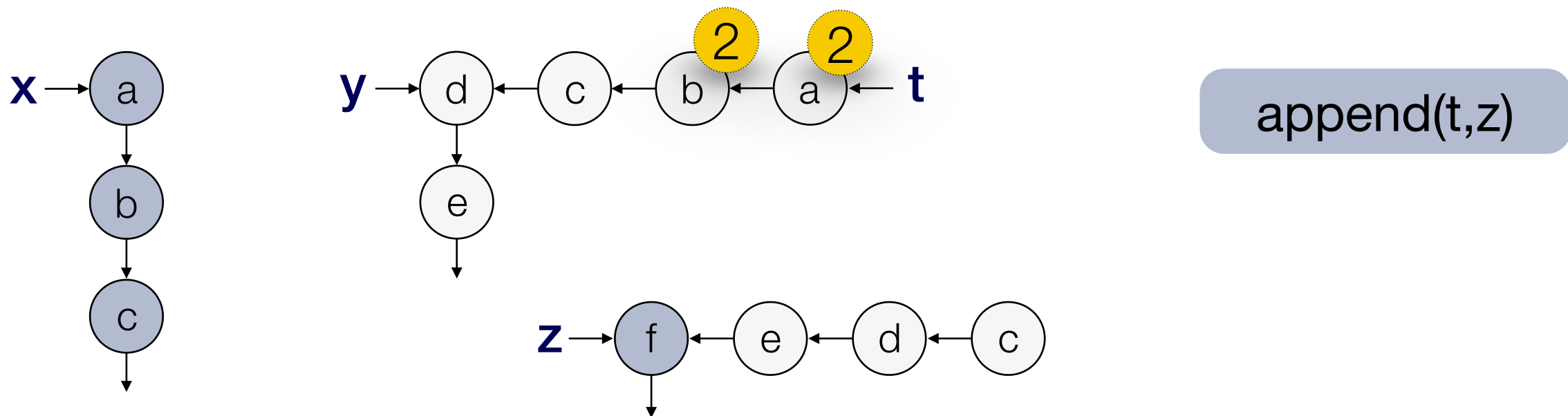
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



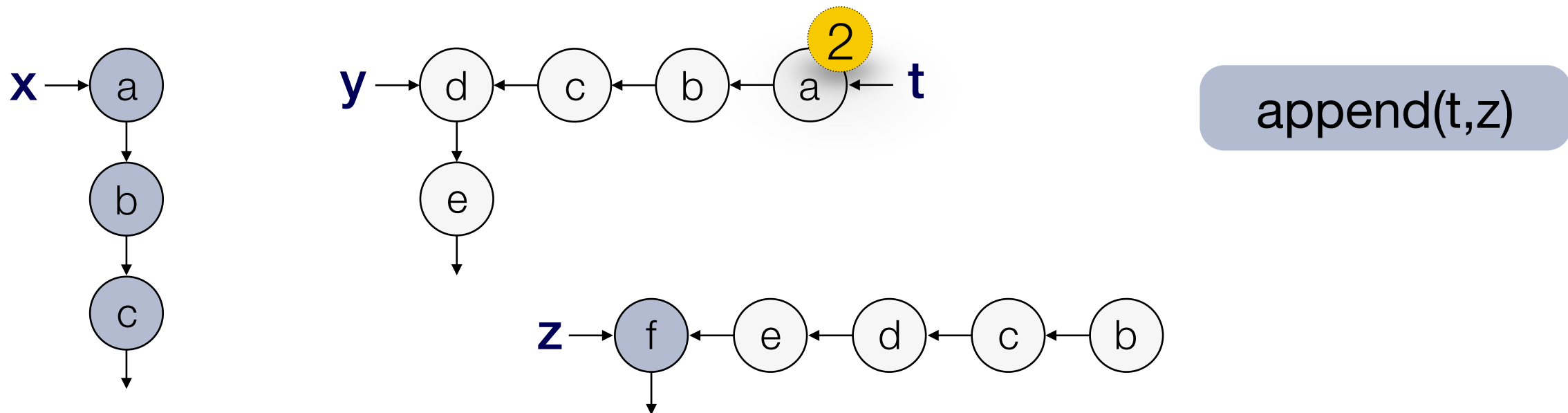
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



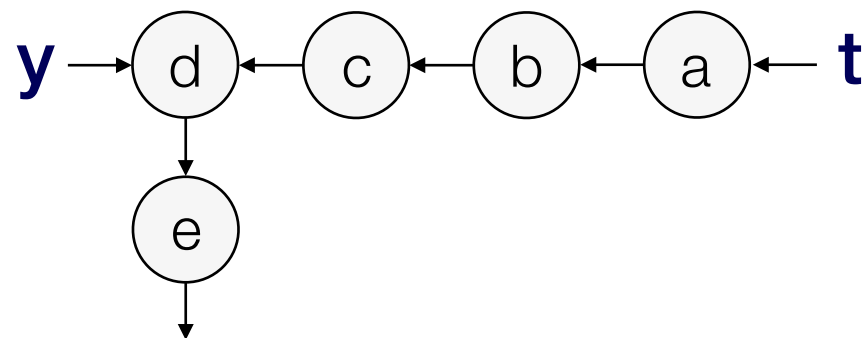
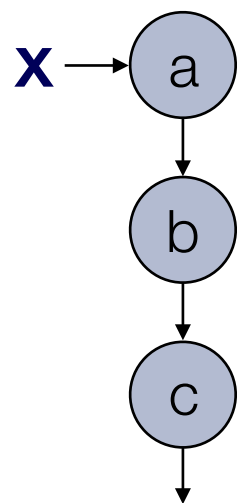
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

Example: Composing Calls of Append

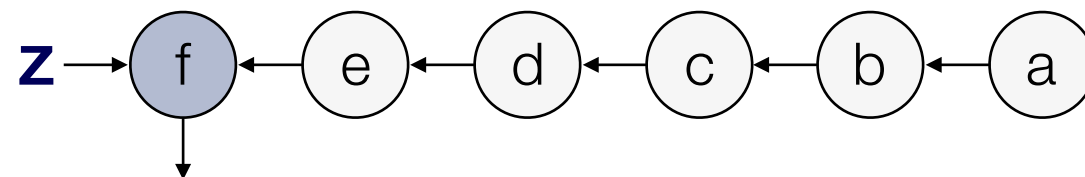
```
f(x,y,z) = {  
  t = append(x,y);  
  append(t,z)  
}
```

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



append(t,z)



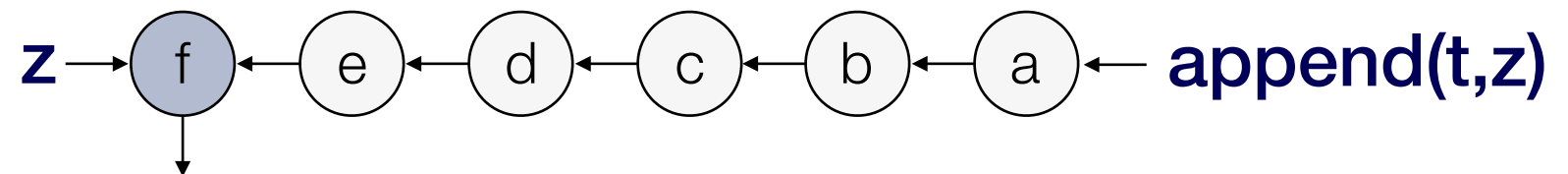
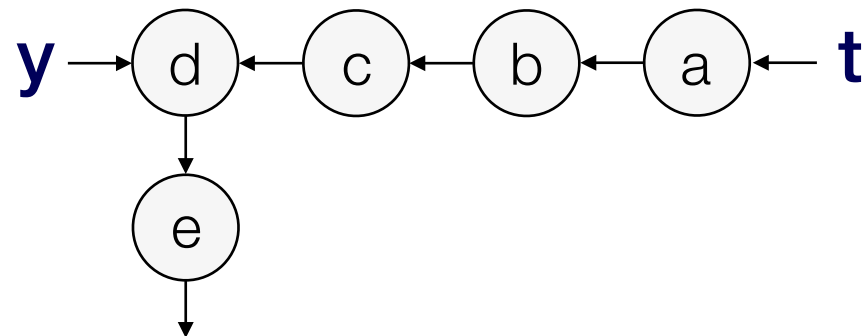
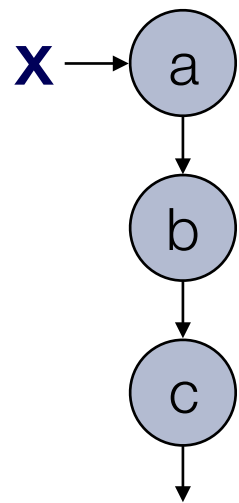
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

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



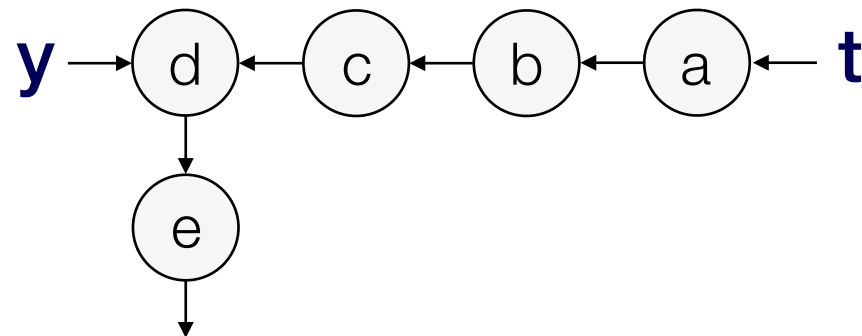
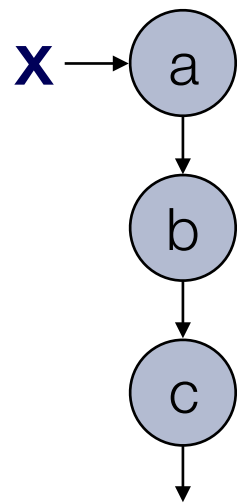
Initial potential: $4*n + 2*m = 4*3 + 2*2 = 16$

Example: Composing Calls of Append

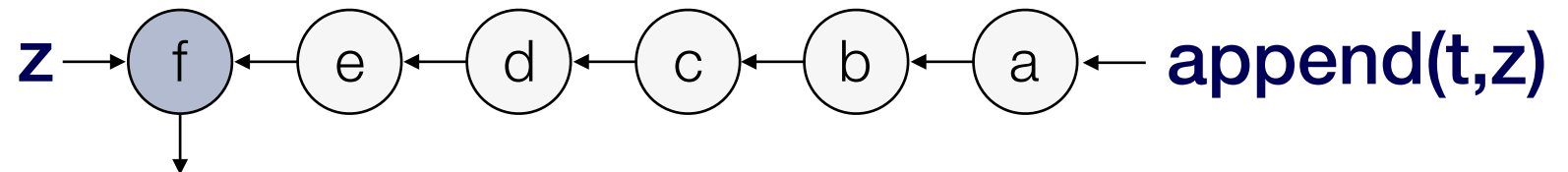
```
f(x,y,z) = {  
  t = append(x,y);  
  append(t,z)  
}
```

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

$f(x, y, z) = \{$ $\text{append}: (L^4(\text{int}), L^2(\text{int})) \xrightarrow{0/0} L^2(\text{int})$

$t = \text{append}(x, y);$

$\text{append}(t, z)$

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

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

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

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 $m \cdot n^2$.

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 $m \cdot n^2$.

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:

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

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

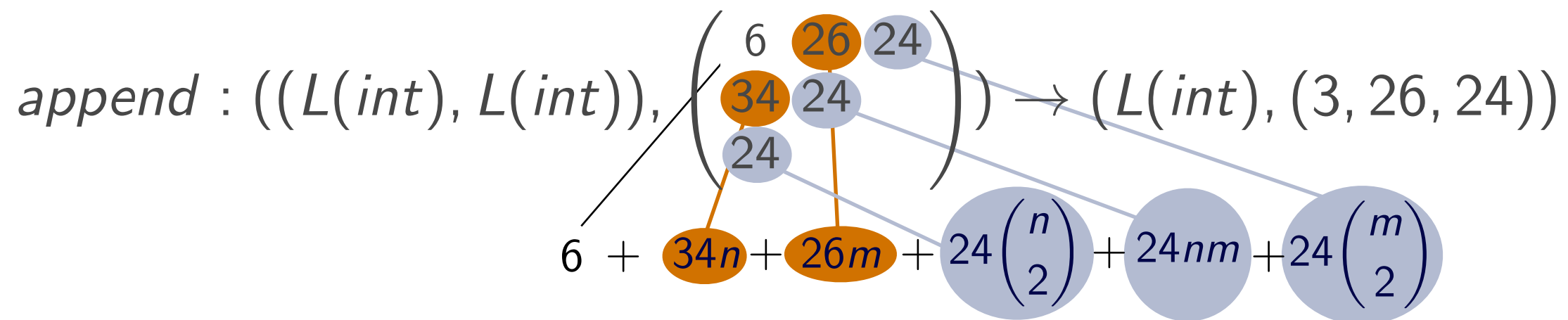
$$\textit{quicksort} : (L(\textit{int}), (3, 26, 24)) \rightarrow (L(\textit{int}), (0, 0, 0))$$

Example: Polynomial Potential Functions

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

Computed time bound:

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



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

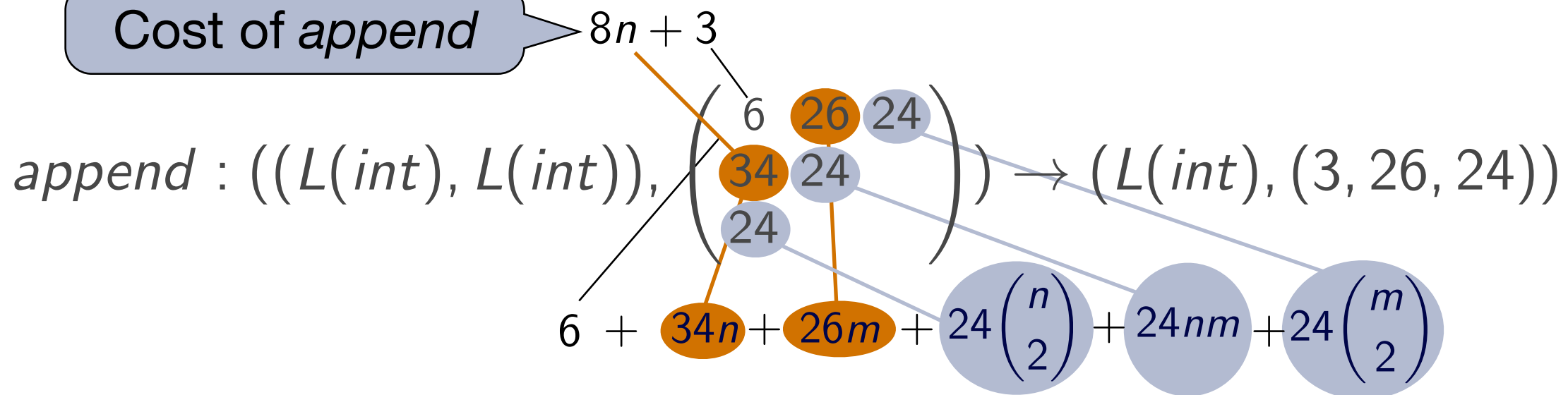
Example: Polynomial Potential Functions

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

Computed time bound:

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

Cost of *append*



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

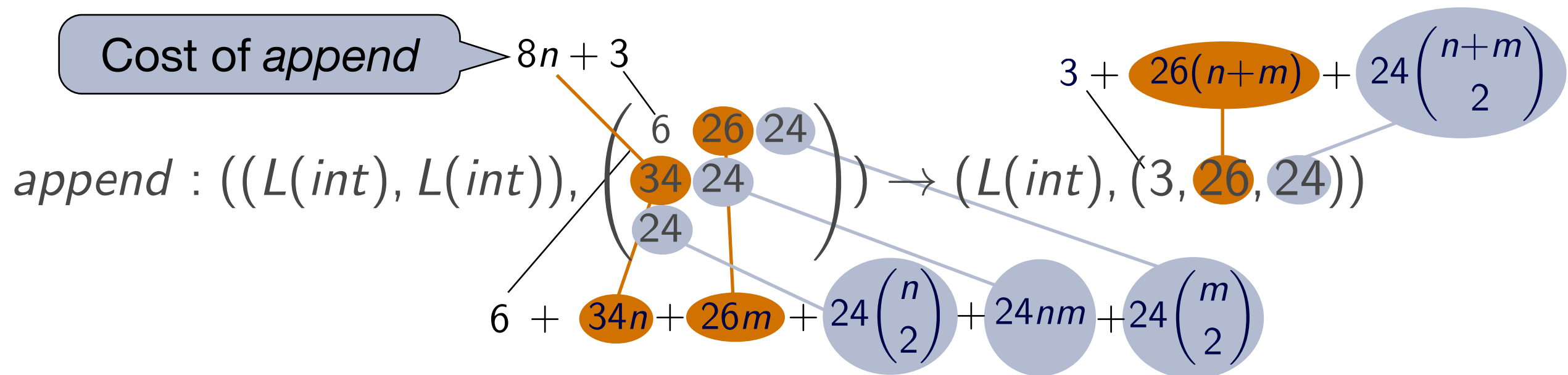
Example: Polynomial Potential Functions

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

Computed time bound:

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

Cost of *append*



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

Multivariate Resource Polynomials [POPL'11]

Map data structures to non-negative rational numbers

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

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

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

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

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

Multivariate Resource Polynomials [POPL'11]

Map data structures to non-negative rational numbers

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

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

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

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

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

Important innovation:
sigma-pi formula for data
structures

Resource Polynomials: Examples

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

Resource Polynomials: Examples

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

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

Resource Polynomials: Examples

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

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

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

Resource Polynomials: Examples

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

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

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

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

Resource Polynomials: Examples

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

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

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

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

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

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

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

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 from the OCaml compiler)
- Currently we use Coin-Or's CLP C interface
- **Features:**
 - 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>

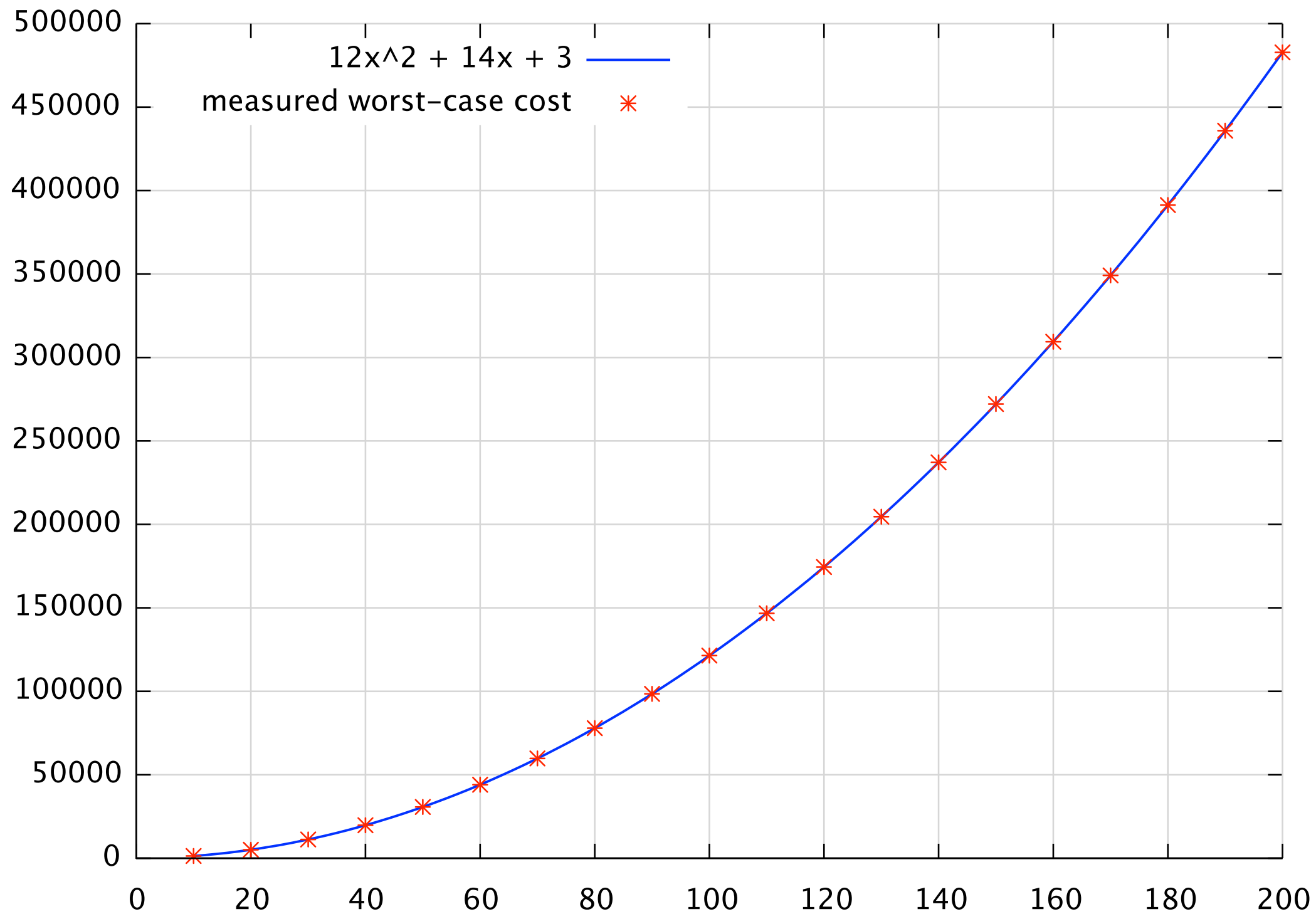
- Based on Inria's **OCaml** compiler
- ~**12,000 lines of code** (+ ~29,000 loc from the OCaml compiler)
- Currently we use Coin-Or's CLP C interface
- **Features:**
 - Higher-order functions and polymorphism
 - User defined inductive types
 - Parallel evaluation
 - Side effects
 - User defined (non-monotone)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

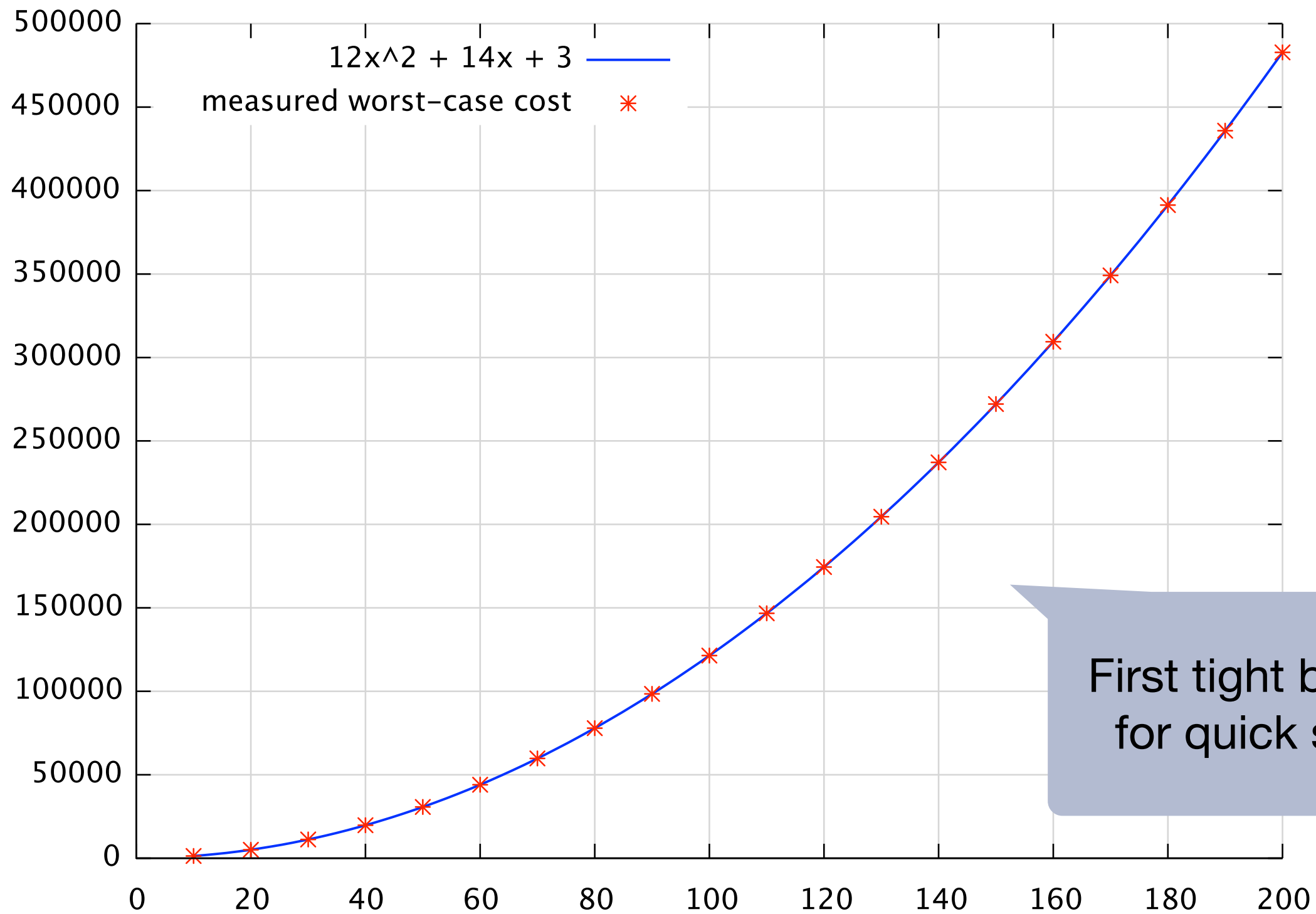
Micro Benchmarks

Evaluation-Step Bounds



Quick Sort for Integers

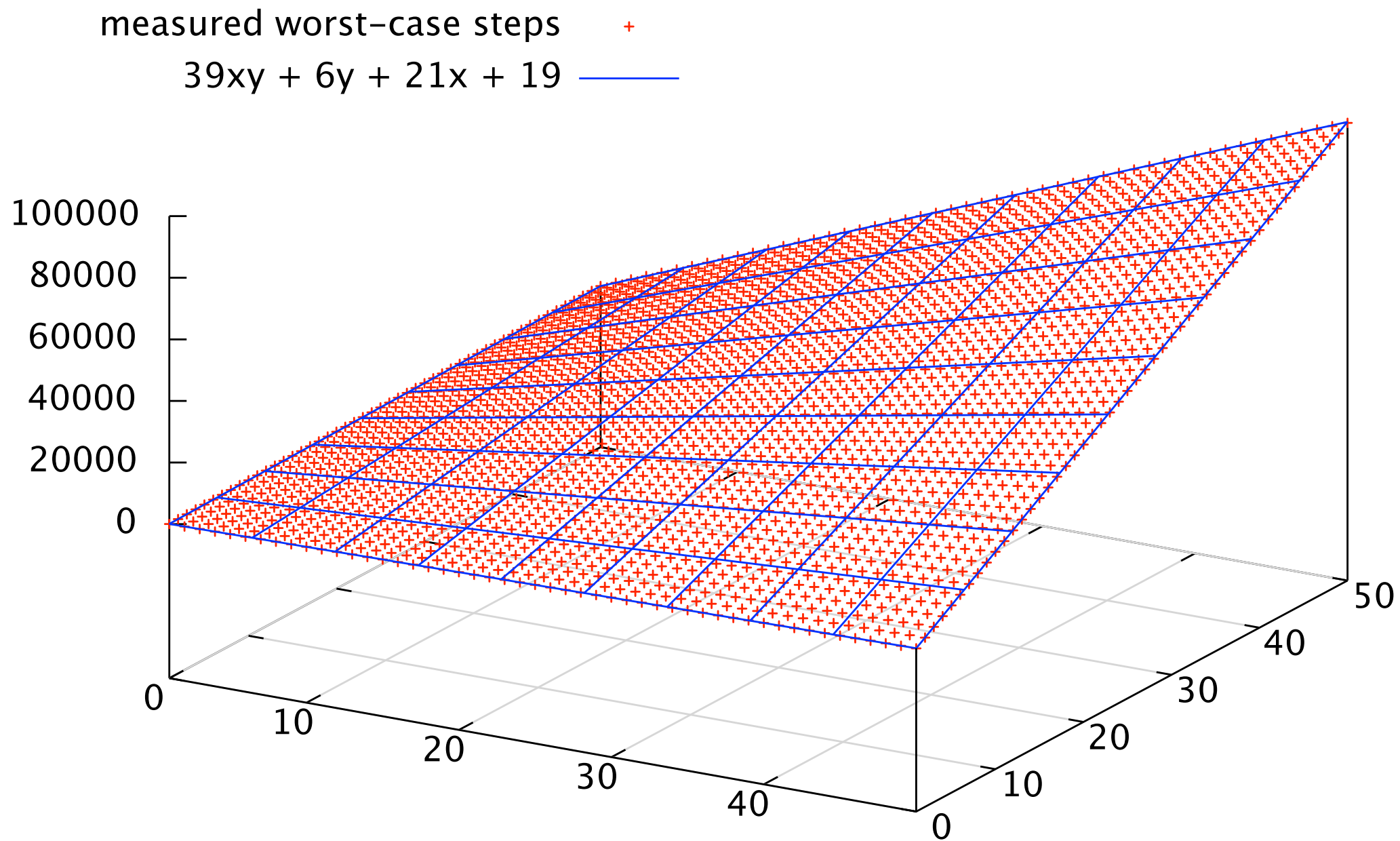
Evaluation-step bound vs.
measured behavior



First tight bound
for quick sort.

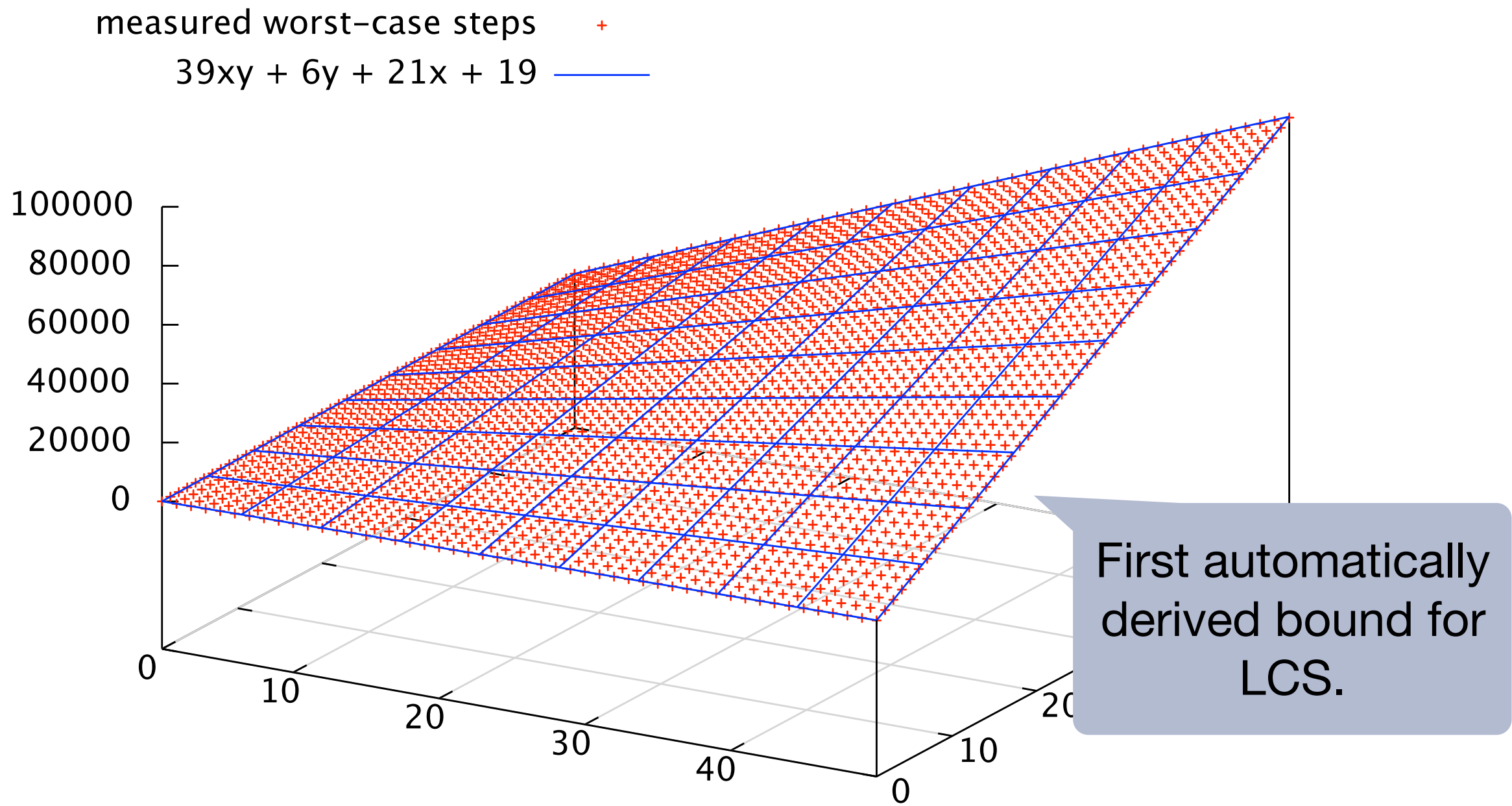
Quick Sort for Integers

Evaluation-step bound vs.
measured behavior



Longest Common
 Subsequence

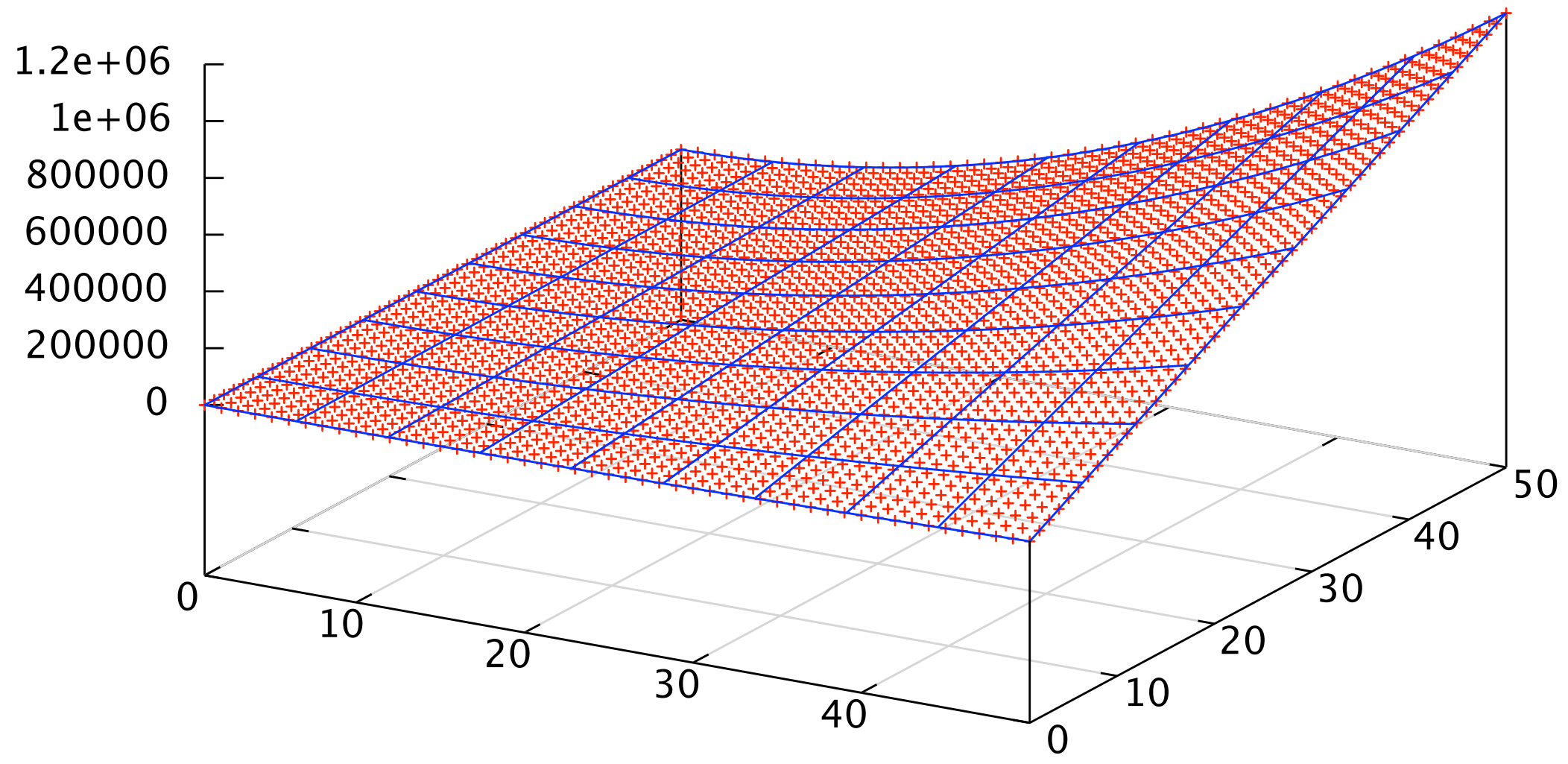
Evaluation-step bound vs.
 measured behavior



Longest Common
Subsequence

Evaluation-step bound vs.
measured behavior

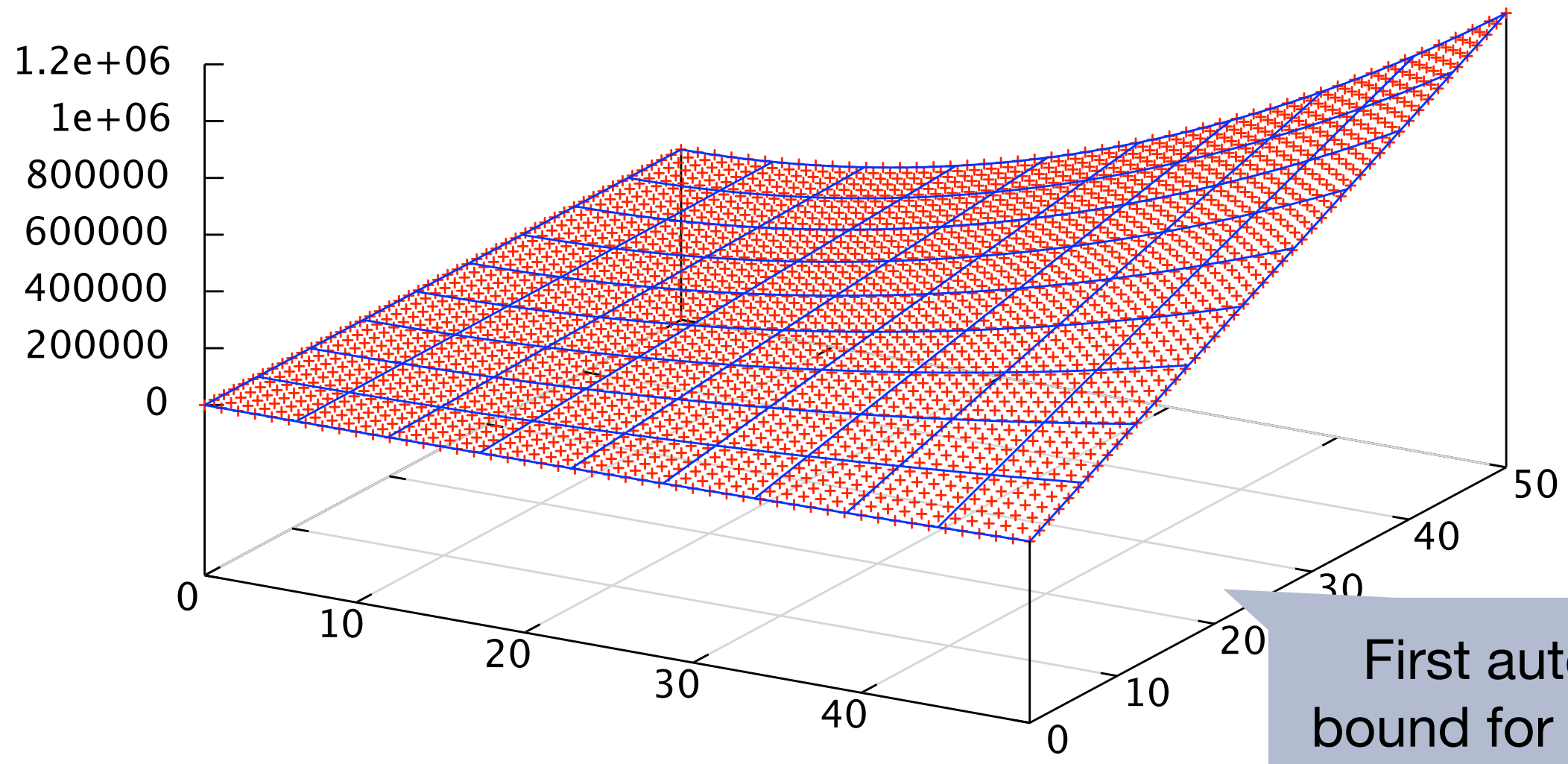
measured worst-case steps +
 $8xy + 8x + 8y + 4x + 3$ —



Insertion Sort for Strings

Evaluation-step bound vs. measured behavior

measured worst-case steps +
 $8xy + 8xx - 8xy + 4x + 3$ —



First automatic bound for a sorting algorithm for strings.

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
<i>steps</i>	243	3218	72.10s	16	130	60	28	239	4	225
<i>heap</i>	243	3218	70.36s	41	112	60	22	239	4	225
<i>tick</i>	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?

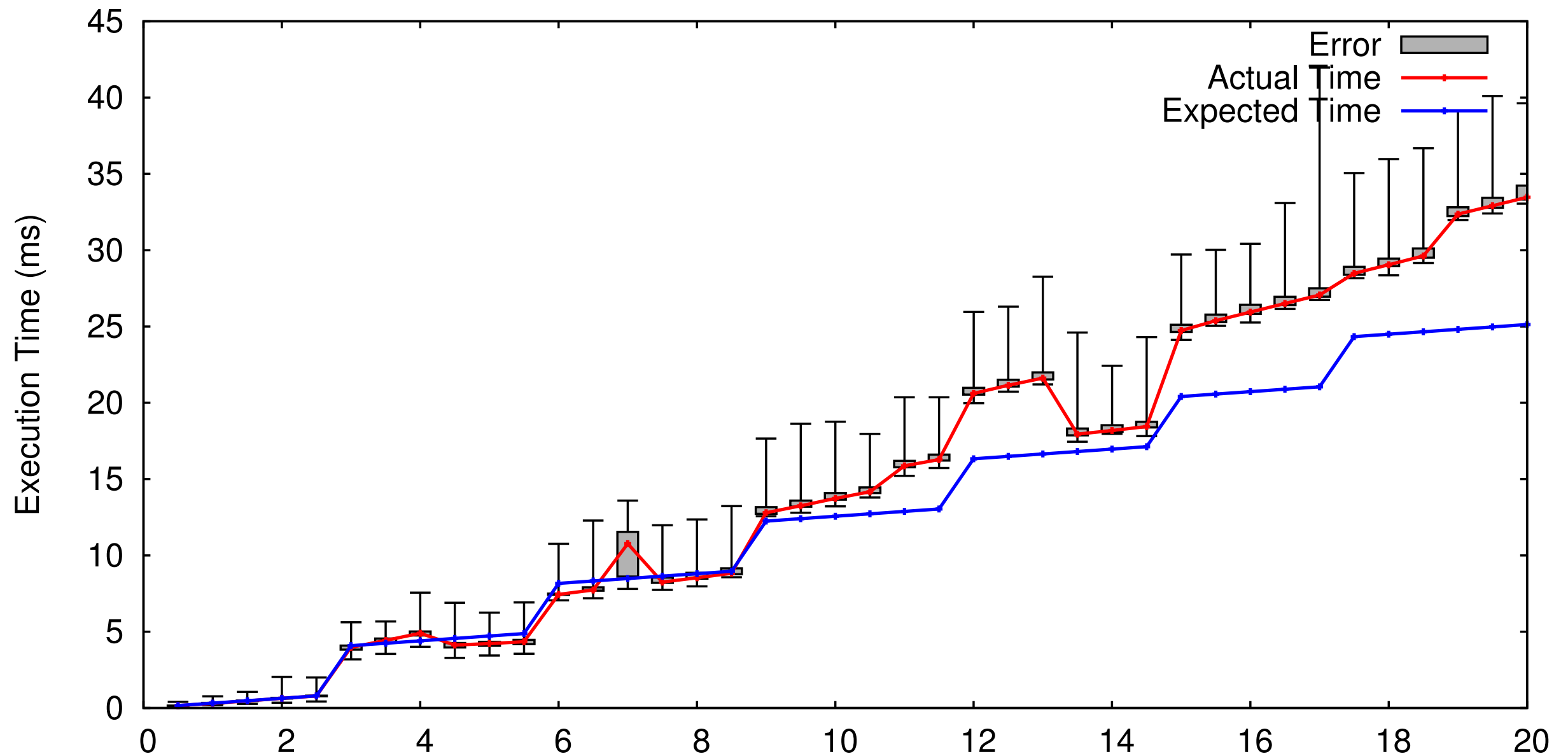
Machine Learning Cost Models

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:

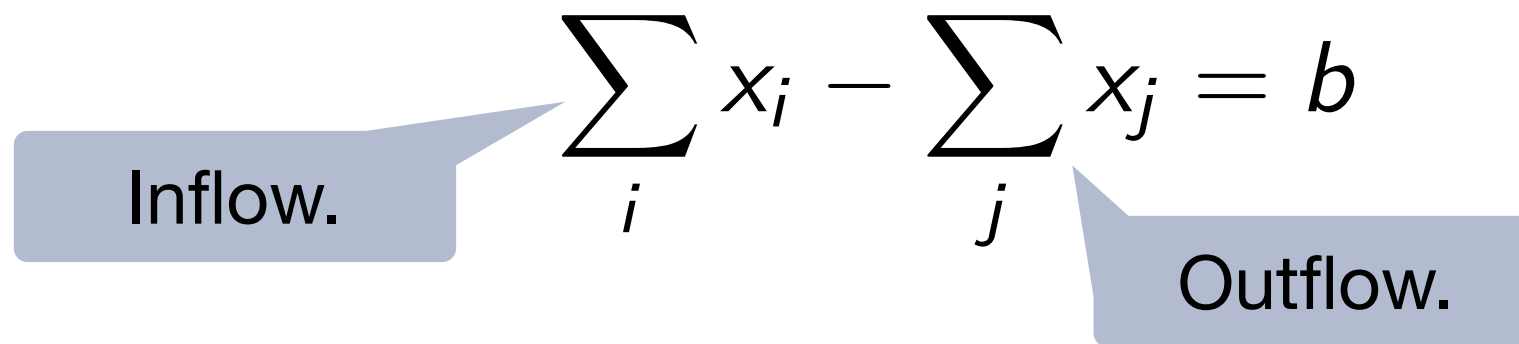
Inflow.

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

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:



The diagram shows the equation $\sum_i x_i - \sum_j x_j = b$. A light blue callout box on the left points to the first summation term $\sum_i x_i$ and contains the text "Inflow.". A light blue callout box on the right points to the second summation term $\sum_j x_j$ and contains the text "Outflow.".

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

Network-Flow Problems

Constraints we derive are *almost* network-flow problems

Network-flow constraints:

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

Inflow.

Outflow.

Constant sink/
source.

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.

Constant sink/
source.

$$l_i \leq x_i \leq u_i$$

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.

Constant sink/
source.

$$l_i \leq x_i \leq u_i$$

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

$$q \geq p_1 \quad q \geq p_2$$

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

$$q \geq p_1 \quad q \geq p_2$$

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

$$q \geq p_1 \quad q \geq p_2$$

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

$$q \geq p_1 \quad q \geq p_2$$

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

$$q \geq p_1$$

$$q \geq p_2$$

Cover cost in both branches (possible waste).

Accounting for Size Change

List types: $L^{(q_1, \dots, q_k)}(A)$

Potential: $\Phi(\ell : L^{(q_1, \dots, q_k)}) = \sum_{i=1, \dots, k} q_i \binom{|\ell|}{i}$

Additive shift: $\triangleleft(q_1, \dots, q_k) = (q_1 + q_2, \dots, q_{k-1} + q_k, q_k)$

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

Accounting for Size Change

List types: $L^{(q_1, \dots, q_k)}(A)$

Potential: $\Phi(\ell : L^{(q_1, \dots, q_k)}) = \sum_{i=1, \dots, k} q_i \binom{|\ell|}{i}$

Additive shift: $\triangleleft(q_1, \dots, q_k) = (q_1 + q_2, \dots, q_{k-1} + q_k, q_k)$

potential of a list

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

Accounting for Size Change

List types: $L^{(q_1, \dots, q_k)}(A)$

Potential: $\Phi(\ell : L^{(q_1, \dots, q_k)}) = \sum_{i=1, \dots, k} q_i \binom{|\ell|}{i}$

Additive shift: $\triangleleft(q_1, \dots, q_k) = (q_1 + q_2, \dots, q_{k-1} + q_k, q_k)$

potential of a list

constant

potential of the tail
using the shift

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

Generation of Linear Constraints

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

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

2. Pattern: one recursive call and polynomial spill

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

3. It is easy to share potential when aliasing data

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

Generation of Linear Constraints

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

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

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

2. Pattern: one recursive call and polynomial spill

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

3. It is easy to share potential when aliasing data

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

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 $\sum_i q_i \binom{x}{i}$

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

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

Constraints:

$$q_1 = 1 \cdot q$$

$$q_2 = 2 \cdot q$$

Coefficients for
change of basis.

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

Constraint Solving in Practice

- LP solving of our constraints is linear in practice
- CLP and CPLEX are similar; Ip_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>