Live Heap Space Analysis for Languages with Garbage Collection

Elvira Albert, Samir Genaim, Miguel Gómez-Zamalloa

Complutense University of Madrid (Spain)

2009 International Symposium on Memory Management (ISMM'09)

Dublin, June 20, 2009

Introduction

- *Cost Analysis* is the automatic study of *program efficiency* (or the resource consumption).
  - Its aim is to statically **estimate the cost of a program execution in terms of the size of its input args.**

Introduction

- *Cost Analysis* is the automatic study of *program efficiency* (or the resource consumption).
  - Its aim is to statically **estimate the cost of a program execution in terms of the size of its input args.**
- The cost can be defined w.r.t. different *cost models*:
  - number of instructions executed
  - **memory allocated**
  - number calls to certain methods: billable events on a mobile

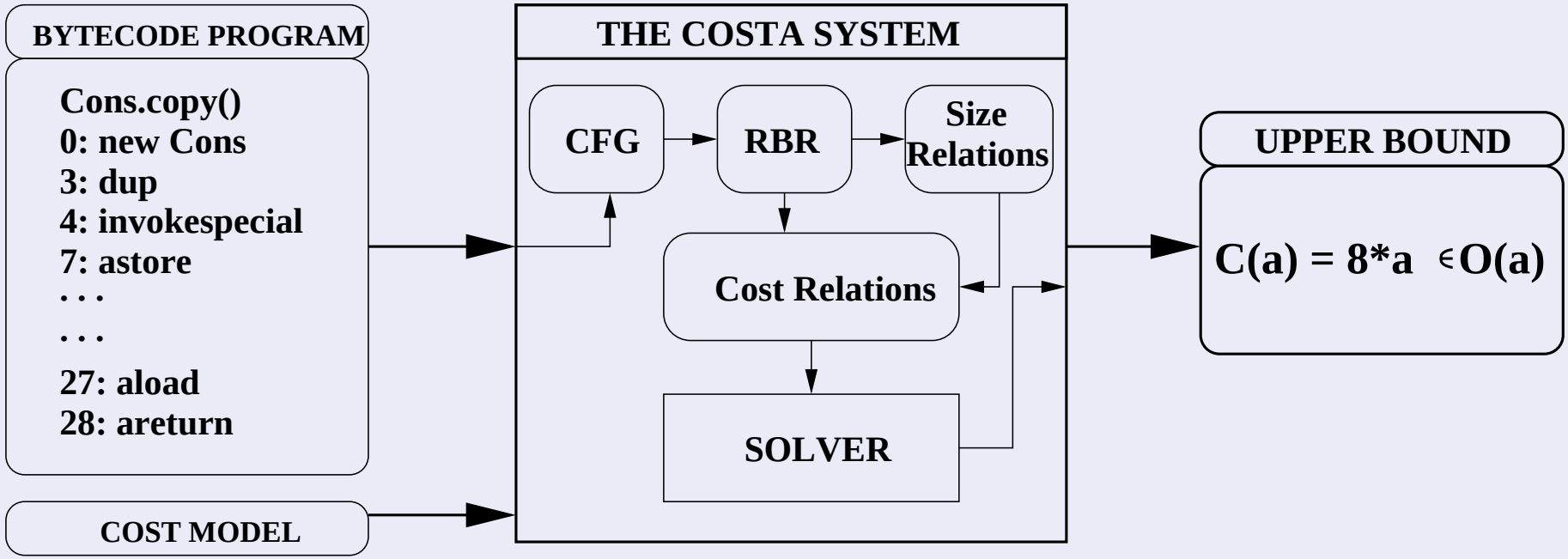
Introduction

- *Cost Analysis* is the automatic study of *program efficiency* (or the resource consumption).
  - Its aim is to statically **estimate the cost of a program execution in terms of the size of its input args.**
- The cost can be defined w.r.t. different *cost models*:
  - number of instructions executed
  - **memory allocated**
  - number calls to certain methods: billable events on a mobile
- Finding the exact cost of programs is undecidable, but it is possible to infer useful information (*bounds*):
  - **Upper bounds**: a program runs within the resources available.
  - *Lower bounds*: useful for scheduling distributed execution.

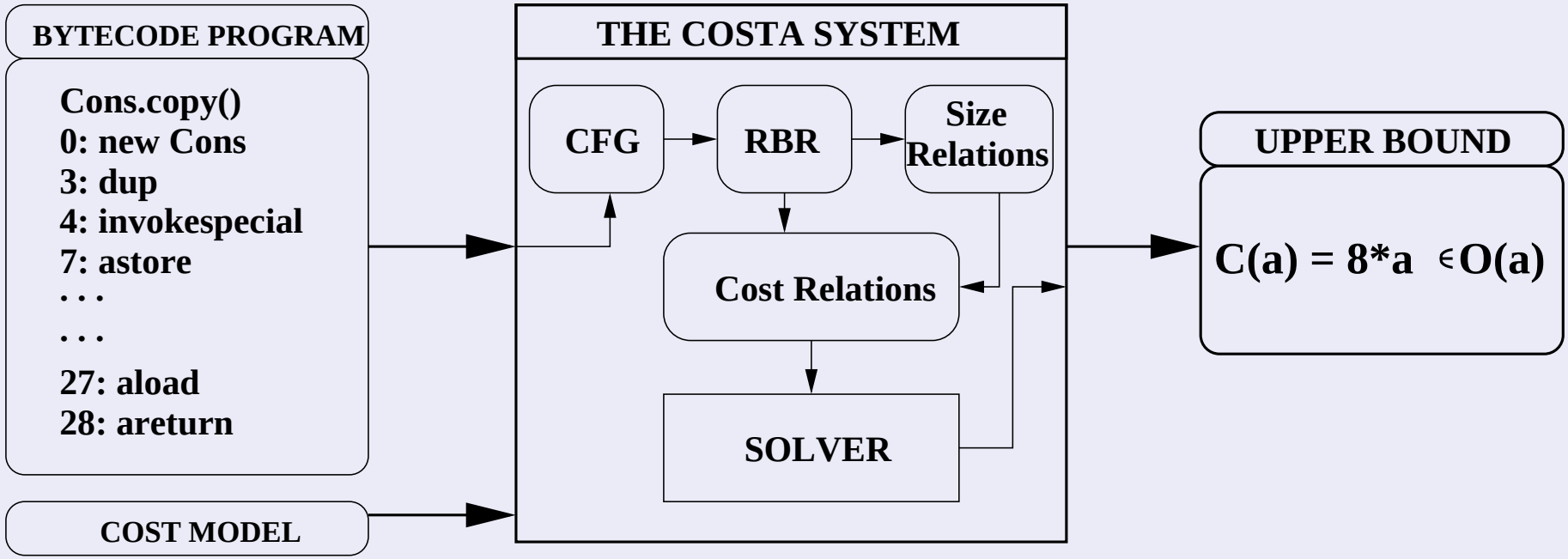
Introduction

- *Cost Analysis* is the automatic study of *program efficiency* (or the resource consumption).
  - Its aim is to statically **estimate the cost of a program execution in terms of the size of its input args.**
- The cost can be defined w.r.t. different *cost models*:
  - number of instructions executed
  - **memory allocated**
  - number calls to certain methods: billable events on a mobile
- Finding the exact cost of programs is undecidable, but it is possible to infer useful information (*bounds*):
  - **Upper bounds**: a program runs within the resources available.
  - *Lower bounds*: useful for scheduling distributed execution.
- Two classes of upper bounds can be considered:
  - **non-asymptotic** (or concrete, or micro-analysis)
  - asymptotic (or macro-analysis)

COSTA: Cost Analyzer for Java Bytecode

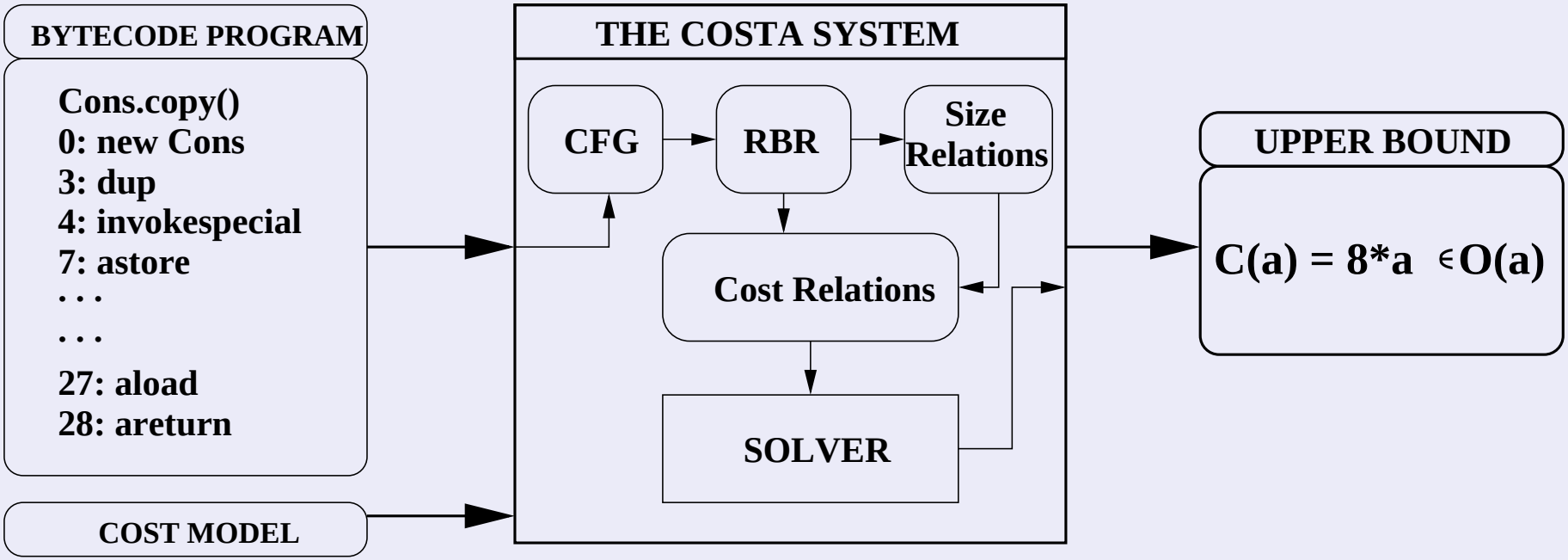


COSTA: Cost Analyzer for Java Bytecode



- For mobile code, we do not have access to source code.
- Java Bytecode: widely used (mobile systems), platform indep., etc

COSTA: Cost Analyzer for Java Bytecode



- For mobile code, we do not have access to source code.
- Java Bytecode: widely used (mobile systems), platform indep., etc

I will focus on the main components and on cost models *heap* and *peak*.

HEAP: Total Allocation Analysis

- Symbolic *cost model* for heap consumption, *size*(Tree), *size*(Integer)..
- We get upper bounds of the *total* allocated memory.
- In presence of **garbage collection (GC)**, it is a too pessimistic estimation of the actual memory consumption.

HEAP: Total Allocation Analysis

- Symbolic *cost model* for heap consumption, *size*(Tree), *size*(Integer)..
- We get upper bounds of the *total* allocated memory.
- In presence of **garbage collection (GC)**, it is a too pessimistic estimation of the actual memory consumption.

PEAK: Live Heap Space Analysis

- Aims at approximating the maximum or **peak** heap usage.
- Scope-based GC model:
  1. Reclaims unreachable memory when methods return. This assumption can be refined up to an ideal GC.
  2. Collects unreachable objects which have been created during the execution of the corresponding method call and not before.
- Much tighter estimation in presence of GC.

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \text{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \text{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0. \\
\\
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \text{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a. \\
\\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \text{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle). \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \text{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \text{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0. \\
\\
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \text{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a. \\
\\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \text{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle). \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \text{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \text{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0. \\
\\
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \text{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a. \\
\\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \text{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle). \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \text{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \text{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0.
\end{array}
$$

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \text{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a.
\end{array}
$$

$$
\begin{array}{l}
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \text{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle).
\end{array}
$$

$$
\begin{array}{l}
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \mathrm{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \mathrm{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0. \\
\\
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \mathrm{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a. \\
\\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \mathrm{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle). \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \text{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \text{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0. \\
\\
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \text{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a. \\
\\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \text{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle). \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Java to Intermediate representation

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return
      new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (; n>1 ; n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }
  static Long h(int n) {
    return new Long(n-1);
  }
}
```

$$
\begin{array}{l}
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n > 0}, \\
\quad s_0 := \text{new } \text{Tree}^1; \\
\quad s_1 := n-1, \\
\quad m(\langle s_1\rangle,\langle s_1\rangle), \\
\quad s_2 := n-1, \\
\quad m(\langle s_2\rangle,\langle s_2\rangle), \\
\quad f(\langle n\rangle,\langle s_3\rangle), \\
\quad \text{init}(\langle s_0,s_1,s_2,s_3\rangle,\langle\rangle), \\
\quad r = s_0. \\
\\
m(\langle n\rangle,\langle r\rangle)::= \\
\quad \mathbf{n \leq 0}, \\
\quad r := \text{null}.
\end{array}
$$

$$
\begin{array}{l}
f(\langle n\rangle,\langle r\rangle)::= \\
\quad a := 0, \\
\quad i := n, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle), \\
\quad f_d(\langle i,a\rangle,\langle i,a\rangle), \\
\quad r := a. \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n > 1}, \\
\quad g(\langle n\rangle,\langle s_0\rangle), \\
\quad \text{intValue}_1(\langle s_0\rangle,\langle s_0\rangle) \\
\quad a := a + s_0, \\
\quad n := n/2, \\
\quad f_c(\langle n,a\rangle,\langle n,a\rangle). \\
\\
f_c(\langle n,a\rangle,\langle n,a\rangle)::= \\
\quad \mathbf{n \leq 1}.
\end{array}
$$

From Intermediate representation to cost equations

$m(\langle n\rangle, \langle r\rangle)::=$
  $\mathbf{n > 0}$,
  $s_0 :=$ new Tree^1;
  $s_1 := n - 1$,
  $m(\langle s_1\rangle, \langle s_1\rangle)$,
  $s_2 := n - 1$,
  $m(\langle s_2\rangle, \langle s_2\rangle)$,
  $f(\langle n\rangle, \langle s_3\rangle)$,
  $init(\langle s_0, s_1, s_2, s_3\rangle, \langle\rangle)$,
  $r = s_0$.

$m(\langle n\rangle, \langle r\rangle)::=$
  $\mathbf{n \leq 0}$,
  $r :=$ null.

$m(n) = size(\mathrm{Tree}^1) + m(n-1) + m(n-1) + f(n) + init(1, n-1, n-1, s_3)$ $\{n>0\}$

$m(n) = 0$ $\{n\leq 0\}$

From Intermediate representation to cost equations

$m(\langle n\rangle, \langle r\rangle) ::=$
$\quad \mathbf{n > 0},$
$\quad s_0 := \text{new } \text{Tree}^1;$
$\quad s_1 := n - 1,$
$\quad m(\langle s_1\rangle, \langle s_1\rangle),$
$\quad s_2 := n - 1,$
$\quad m(\langle s_2\rangle, \langle s_2\rangle),$
$\quad f(\langle n\rangle, \langle s_3\rangle),$
$\quad init(\langle s_0, s_1, s_2, s_3\rangle, \langle\rangle),$
$\quad r = s_0.$

$m(\langle n\rangle, \langle r\rangle) ::=$
$\quad \mathbf{n \leq 0},$
$\quad r := \text{null}.$

$$m(n) = \text{size}(\text{Tree}^1) + m(n-1) + m(n-1) + f(n) + init(1, n-1, n-1, s_3) \quad \{n>0\}$$
$$m(n) = 0 \quad \{n \leq 0\}$$

From Intermediate representation to cost equations

$m(\langle n\rangle, \langle r\rangle) ::=$
  $\mathbf{n > 0}$,
  $s_0 := \text{new } \text{Tree}^1$;
  $s_1 := n - 1$,
  $m(\langle s_1\rangle, \langle s_1\rangle)$,
  $s_2 := n - 1$,
  $m(\langle s_2\rangle, \langle s_2\rangle)$,
  $f(\langle n\rangle, \langle s_3\rangle)$,
  $\text{init}(\langle s_0, s_1, s_2, s_3\rangle, \langle\rangle)$,
  $r = s_0$.

$m(\langle n\rangle, \langle r\rangle) ::=$
  $\mathbf{n \leq 0}$,
  $r := \text{null}$.

$m(n) = \text{size}(\text{Tree}^1) + m(n-1) + m(n-1) + f(n) + \text{init}(1, n-1, n-1, s_3)$ $\{n>0\}$

$m(n) = 0$ $\{n\leq 0\}$

From Intermediate representation to cost equations

$m(\langle n\rangle, \langle r\rangle) ::=$
$\quad \mathbf{n > 0},$
$\quad s_0 := \text{new } \mathrm{Tree}^1;$
$\quad s_1 := n - 1,$
$\quad m(\langle s_1\rangle, \langle s_1\rangle),$
$\quad s_2 := n - 1,$
$\quad m(\langle s_2\rangle, \langle s_2\rangle),$
$\quad f(\langle n\rangle, \langle s_3\rangle),$
$\quad \mathrm{init}(\langle s_0, s_1, s_2, s_3\rangle, \langle\rangle),$
$\quad r = s_0.$

$m(\langle n\rangle, \langle r\rangle) ::=$
$\quad \mathbf{n \leq 0},$
$\quad r := \mathsf{null}.$

$$m(n) = \mathrm{size}(\mathrm{Tree}^1) + m(n-1) + m(n-1) + f(n) + \mathrm{init}(1, n-1, n-1, s_3) \quad \{n>0\}$$

$$m(n) = 0 \quad \{n \leq 0\}$$

From Intermediate representation to cost equations

$m(\langle n\rangle, \langle r\rangle) ::=$
$\quad \mathbf{n > 0},$
$\quad s_0 := \text{new } \text{Tree}^1;$
$\quad s_1 := n - 1,$
$\quad m(\langle s_1\rangle, \langle s_1\rangle),$
$\quad s_2 := n - 1,$
$\quad m(\langle s_2\rangle, \langle s_2\rangle),$
$\quad f(\langle n\rangle, \langle s_3\rangle),$
$\quad init(\langle s_0, s_1, s_2, s_3\rangle, \langle\rangle),$
$\quad r = s_0.$

$m(\langle n\rangle, \langle r\rangle) ::=$
$\quad \mathbf{n \leq 0},$
$\quad r := \text{null}.$

$$m(n) = \text{size}(\text{Tree}^1) + m(n-1) + m(n-1) + f(n) + init(1, n-1, n-1, s_3) \quad \{n>0\}$$

$$m(n) = 0 \quad \{n \leq 0\}$$

From Intermediate representation to cost equations

$m(\langle n\rangle, \langle r\rangle) ::=$
  $\mathbf{n > 0}$,
  $s_0 :=$ new Tree^1;
  $s_1 := n - 1$,
  $m(\langle s_1\rangle, \langle s_1\rangle)$,
  $s_2 := n - 1$,
  $m(\langle s_2\rangle, \langle s_2\rangle)$,
  $f(\langle n\rangle, \langle s_3\rangle)$,
  $\mathrm{init}(\langle s_0, s_1, s_2, s_3\rangle, \langle\rangle)$,
  $r = s_0$.

$m(\langle n\rangle, \langle r\rangle) ::=$
  $\mathbf{n \leq 0}$,
  $r :=$ null.

$$
\begin{array}{lll}
m(n) = \mathrm{size}(\mathrm{Tree}^1) + & & \{n>0\} \\
\qquad m(n-1) + m(n-1) + f(n) + & & \\
\qquad \mathrm{init}(1, n-1, n-1, s_3) & & \\
m(n) = 0 & & \{n \leq 0\} \\
f(n) = f_c(n, 0) + f_d(n, a') & & \{\} \\
f_c(n, a) = g(n) + f_c(n/2, a') & & \{n>1\} \\
f_c(n, a) = 0 & & \{n \leq 1\} \\
f_d(i, a) = h(i) + f_d(i/2, a') & & \{i>1\} \\
f_d(i, a) = 0 & & \{i \leq 0\} \\
g(n) = \mathrm{size}(\mathrm{Integer}^2) + \mathrm{size}(\mathrm{Integer}^3) & & \{\} \\
h(n) = \mathrm{size}(\mathrm{Long}^4) & & \{\} \\
\mathrm{init}(\mathrm{this}, l, r, d) = 0 & & \{\}
\end{array}
$$

From Cost Relations to Upper Bounds

solutions computed for f

$total(f) = total(f_c) + total(f_d)$

```
for (; n>1 ; n=n/2 ) a += g(n).intValue();
for (; i>1 ; i=i/2 ) a *= h(i).intValue();
```

$f_c(n) = log(n) * (size(\mathrm{Integer}^3) + size(\mathrm{Integer}^2))$
$f_d(i) = log(i) * size(\mathrm{Long}^4)$

From Cost Relations to Upper Bounds

solutions computed for f

$total(f) = total(f_c) + total(f_d)$

```
for (; n>1 ; n=n/2 ) a += g(n).intValue();
for (; i>1 ; i=i/2 ) a *= h(i).intValue();
```

$f_c(n) = log(n) * (size(\mathrm{Integer}^3) + size(\mathrm{Integer}^2))$
$f_d(i) = log(i) * size(\mathrm{Long}^4)$

solutions computed for m

$m(n) = 2^n * (\underbrace{total(f)}_{exp\ \ times} + size(\mathrm{Tree}^1))$

```
new Tree(m(n-1),m(n-1),f(n));
```

Why Live Heap Space Analysis is Different?

Basic idea in total allocation:

$$\mathbf{total}(\{\mathbf{m_1}; \mathbf{m_2}\}) = \mathbf{total}(\mathbf{m_1}) + \mathbf{total}(\mathbf{m_2})$$

Why Live Heap Space Analysis is Different?

Basic idea in total allocation:

$$\mathbf{total}(\{\mathbf{m_1}; \mathbf{m_2}\}) = \mathbf{total}(\mathbf{m_1}) + \mathbf{total}(\mathbf{m_2})$$

While total memory allocation is an accumulative resource, the live heap space increases and decreases along an execution

Why Live Heap Space Analysis is Different?

Basic idea in total allocation:

$$\mathbf{total}(\{\mathbf{m_1}; \mathbf{m_2}\}) = \mathbf{total}(\mathbf{m_1}) + \mathbf{total}(\mathbf{m_2})$$

While total memory allocation is an accumulative resource, the live heap space increases and decreases along an execution

Basic idea in live Heap Space Analysis:

$$\mathbf{peak}(\{\mathbf{m_1}; \mathbf{m_2}\}) = \mathbf{max}(\mathbf{peak}(\mathbf{m_1}), \mathbf{escaped}(\mathbf{m_1}) + \mathbf{peak}(\mathbf{m_2}))$$

Why Live Heap Space Analysis is Different?

Basic idea in total allocation:

$$\mathbf{total}(\{\mathbf{m_1}; \mathbf{m_2}\}) = \mathbf{total}(\mathbf{m_1}) + \mathbf{total}(\mathbf{m_2})$$

While total memory allocation is an accumulative resource, the live heap space increases and decreases along an execution

Basic idea in live Heap Space Analysis:

$$\mathbf{peak}(\{\mathbf{m_1}; \mathbf{m_2}\}) = \mathbf{max}(\mathbf{peak}(\mathbf{m_1}), \mathbf{escaped}(\mathbf{m_1}) + \mathbf{peak}(\mathbf{m_2}))$$

1. STEP 1: Escaped memory analysis
2. STEP 2: Peak consumption cost relations

Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
  • for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
    • a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

Example

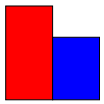
```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

g peak
g escaped



1 2

Example

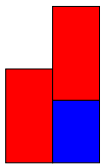
```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
    • a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

g peak
g escaped



1 2

Example

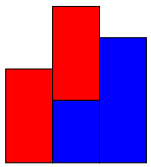
```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

g peak
g escaped



1 2 3

Example

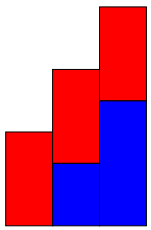
```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
    • a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

g peak
g escaped



Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```

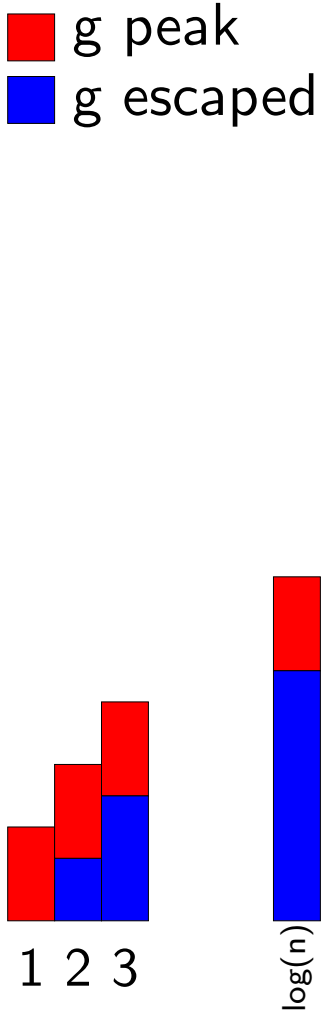
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
    • a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



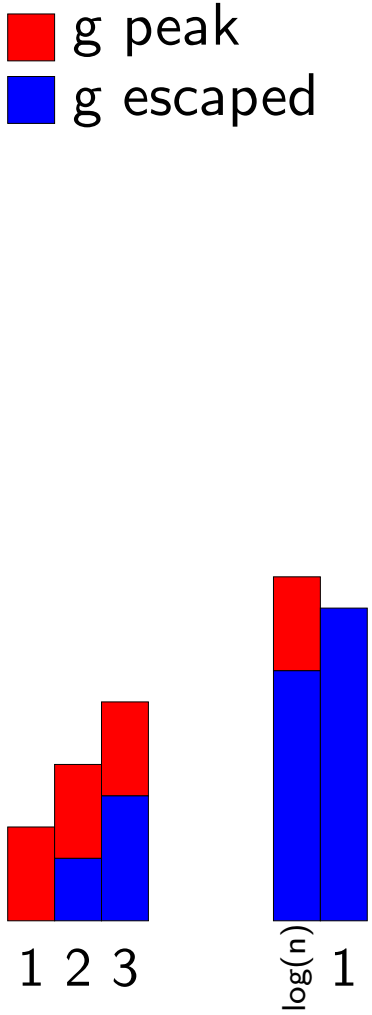
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



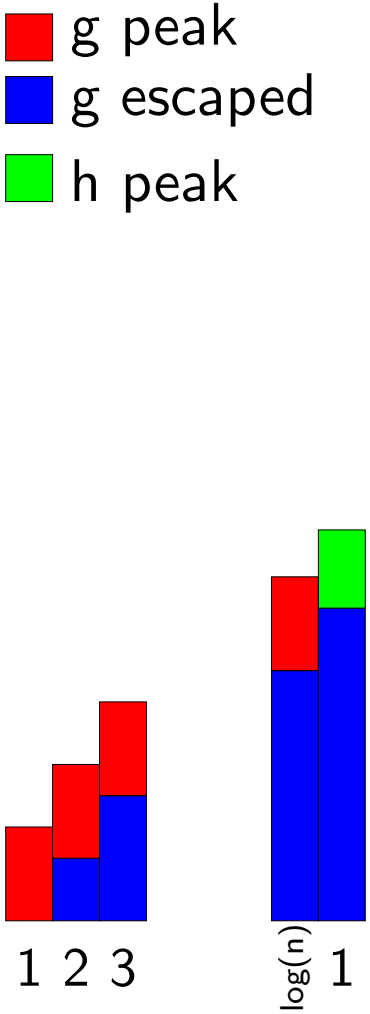
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
     • a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



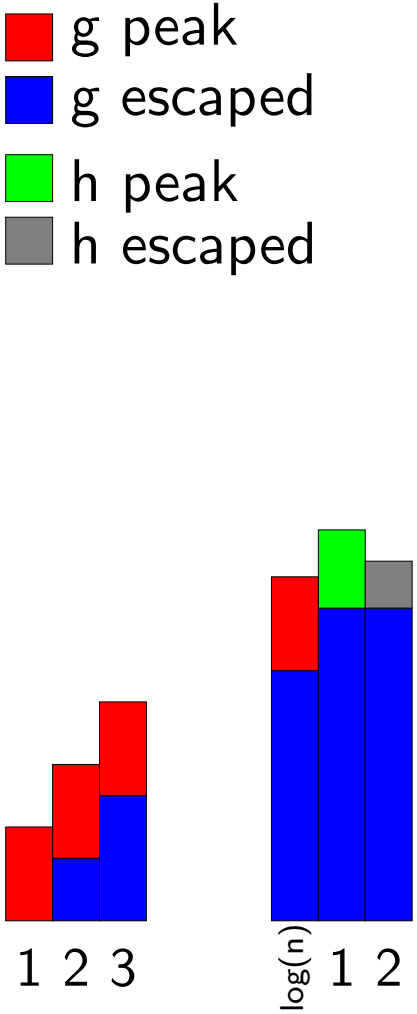
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



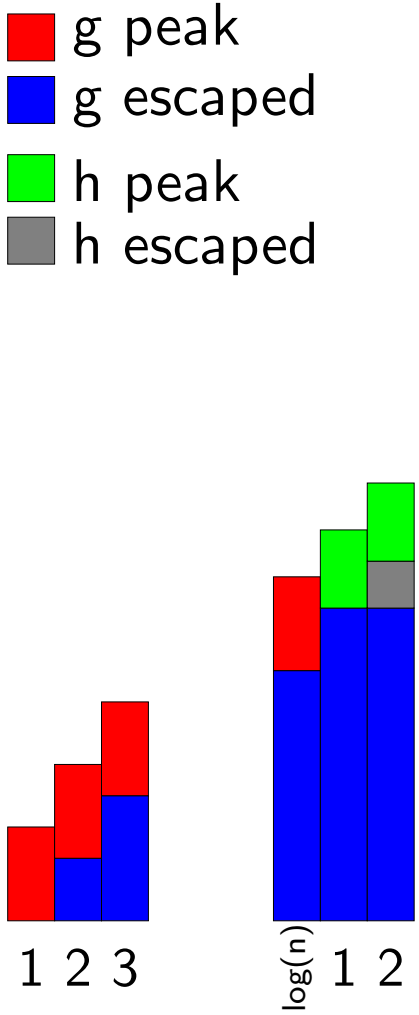
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
     • a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



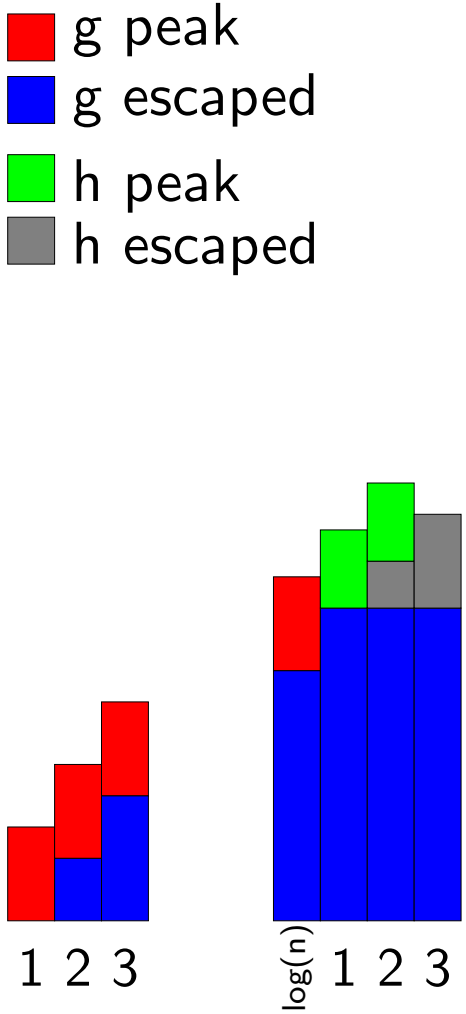
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



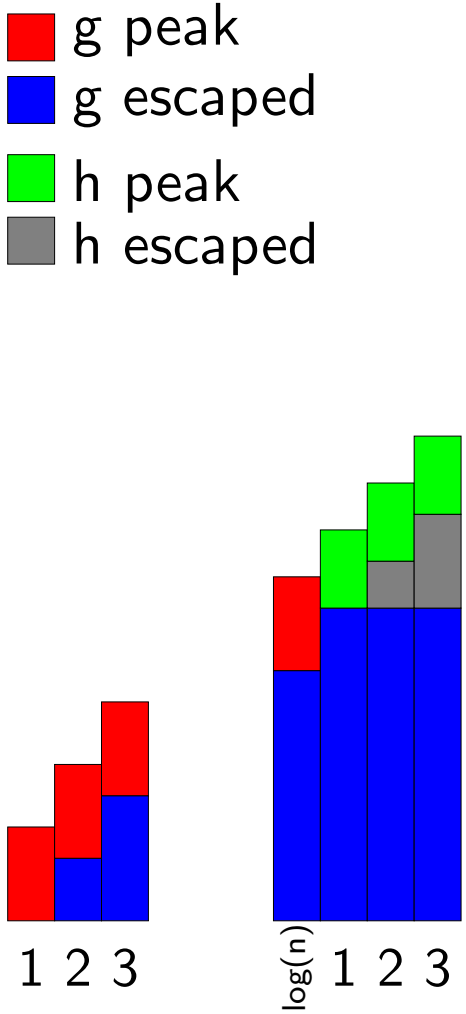
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
     • a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



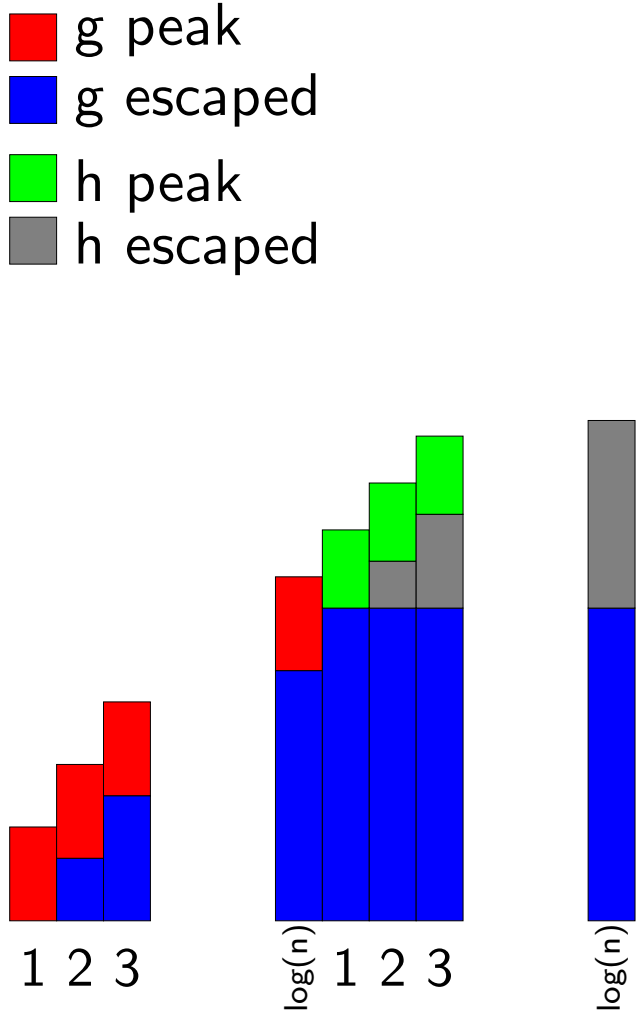
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



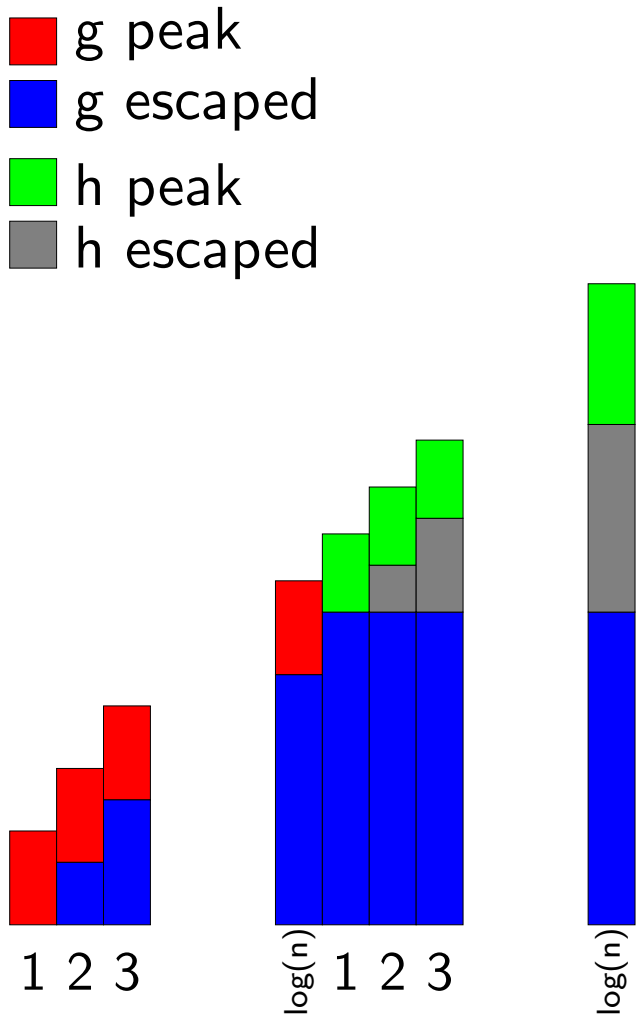
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
    • a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



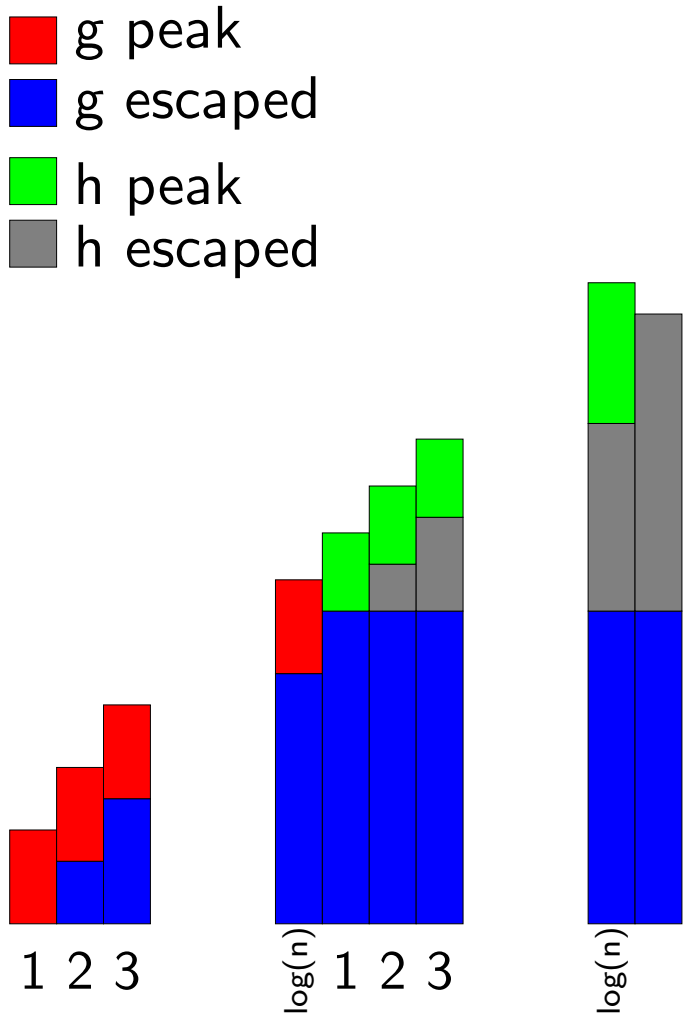
Example

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for (;n>1;n=n/2)
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
  • return a;
  }

  static Integer g(int n) {
    Integer x=new Integer(n);
    return new Integer(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long(n-1);
  }
}
```



Step 1: Inference of Escaped Memory Bounds

- Infer upper bounds on the total memory consumption
- Remove from the bounds the collectable objects
- The set of collectable objects can be approximated from the information computed by escape analysis

Step 1: Inference of Escaped Memory Bounds

- Infer upper bounds on the total memory consumption
- Remove from the bounds the collectable objects
- The set of collectable objects can be approximated from the information computed by escape analysis

collectable objects from m

the objects that have been created along the execution of m and will not be in the memory upon exit from m

Collectable Objects

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new
      Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for(; n>1;n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer^2(n);
    return new Integer^3(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long^4(n-1);
  }
} // end of class Test
```

$$\mathit{collectable}(\mathrm{g}) = \{\mathrm{Integer}^2\}$$
$$\mathit{collectable}(\mathrm{h}) = \emptyset$$
$$\mathit{collectable}(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$
$$\mathit{collectable}(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$

Collectable Objects

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new
      Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for(; n>1;n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }
```

-
```
  static Integer g(int n) {
    Integer x=new Integer²(n);
    return new Integer³(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long⁴(n-1);
  }
} // end of class Test
```

$$\mathit{collectable}(\mathrm{g}) = \{\mathrm{Integer}^2\}$$
$$\mathit{collectable}(\mathrm{h}) = \emptyset$$
$$\mathit{collectable}(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$
$$\mathit{collectable}(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$

Collectable Objects

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new
      Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for(; n>1;n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer^2(n);
    return new Integer^3(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long^4(n-1);
  }
} // end of class Test
```

$$collectable(\mathrm{g}) = \{\mathrm{Integer}^2\}$$
$$collectable(\mathrm{h}) = \emptyset$$
$$collectable(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$
$$collectable(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$

Collectable Objects

```
class Test {
  static Tree m(int n) {
    if ( n>0 ) return new
      Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for(; n>1;n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer^2(n);
    return new Integer^3(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long^4(n-1);
  }
} // end of class Test
```

$$collectable(\mathrm{g}) = \{\mathrm{Integer}^2\}$$
$$collectable(\mathrm{h}) = \emptyset$$
$$collectable(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$
$$collectable(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$

Collectable Objects

```
class Test {
• static Tree m(int n) {
    if ( n>0 ) return new
      Tree(m(n-1),m(n-1),f(n));
    else return null;
  }

  static int f(int n) {
    int a=0,i=n;
    for(; n>1;n=n/2 )
      a += g(n).intValue();
    for(; i>1; i=i/2)
      a *= h(i).intValue();
    return a;
  }

  static Integer g(int n) {
    Integer x=new Integer^2(n);
    return new Integer^3(x.intValue()+1);
  }

  static Long h(int n) {
    return new Long^4(n-1);
  }
} // end of class Test
```

$$collectable(\mathrm{g}) = \{\mathrm{Integer}^2\}$$
$$collectable(\mathrm{h}) = \emptyset$$
$$collectable(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$
$$collectable(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}$$

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

$$
\begin{aligned}
g(n) &= size(\text{Integer}^2) + size(\text{Integer}^3) \\
h(n) &= size(\text{Long}^4) \\
f(n) &= \log_2(n) * (size(\text{Integer}^2) + size(\text{Integer}^3) + size(\text{Long}^4)) \\
m(n) &= (2^n - 1) * (size(\text{Tree}^1) + \\
&\quad \log_2(n) * (size(\text{Integer}^2) + size(\text{Integer}^3) + size(\text{Long}^4)))
\end{aligned}
$$

$$
\begin{aligned}
collectable(\text{g}) &= \{\text{Integer}^2\} \\
collectable(\text{h}) &= \emptyset \\
collectable(\text{f}) &= \{\text{Integer}^2, \text{Integer}^3, \text{Long}^4\} \\
collectable(\text{m}) &= \{\text{Integer}^2, \text{Integer}^3, \text{Long}^4\}
\end{aligned}
$$

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

$$
\begin{aligned}
g(n) &= size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) \\
h(n) &= size(\mathrm{Long}^4) \\
f(n) &= \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)) \\
m(n) &= (2^n - 1) * (size(\mathrm{Tree}^1) + \\
&\quad \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)))
\end{aligned}
$$

$$
\begin{aligned}
&collectable(\mathrm{g}) = \{\mathrm{Integer}^2\} \\
&collectable(\mathrm{h}) = \emptyset \\
&collectable(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\} \\
&collectable(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}
\end{aligned}
$$

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

$$
\begin{aligned}
\check{g}(n) &= size(\mathrm{Integer}^3) \\
h(n) &= size(\mathrm{Long}^4) \\
f(n) &= \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)) \\
m(n) &= (2^n - 1) * (size(\mathrm{Tree}^1) + \\
&\quad \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)))
\end{aligned}
$$

$$
\begin{aligned}
&collectable(\mathrm{g}) = \{\mathrm{Integer}^2\} \\
&collectable(\mathrm{h}) = \emptyset \\
&collectable(\mathrm{f}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\} \\
&collectable(\mathrm{m}) = \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}
\end{aligned}
$$

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

$$
\begin{aligned}
\check{g}(n) &= size(\mathrm{Integer}^3) \\
\check{h}(n) &= size(\mathrm{Long}^4) \\
f(n) &= \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)) \\
m(n) &= (2^n - 1) * (size(\mathrm{Tree}^1) + \\
&\quad \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)))
\end{aligned}
$$

$$
\begin{aligned}
collectable(\mathrm{g}) &= \{\mathrm{Integer}^2\} \\
collectable(\mathrm{h}) &= \emptyset \\
collectable(\mathrm{f}) &= \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\} \\
collectable(\mathrm{m}) &= \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}
\end{aligned}
$$

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

$$
\begin{aligned}
\check{g}(n) &= size(\mathrm{Integer}^3) \\
\check{h}(n) &= size(\mathrm{Long}^4) \\
\check{f}(n) &= 0 \\
m(n) &= (2^n - 1) * (size(\mathrm{Tree}^1) + \\
&\quad \log_2(n) * (size(\mathrm{Integer}^2) + size(\mathrm{Integer}^3) + size(\mathrm{Long}^4)))
\end{aligned}
$$

$$
\begin{aligned}
collectable(\mathrm{g}) &= \{\mathrm{Integer}^2\} \\
collectable(\mathrm{h}) &= \emptyset \\
collectable(\mathrm{f}) &= \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\} \\
collectable(\mathrm{m}) &= \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}
\end{aligned}
$$

Bounds on Escaped Memory

escaped memory of a procedure p

- given the set of collectable objects after return from p, $collectable(p)$,
- given the upper-bound for the total memory allocation $total(p) = exp$
- the escaped memory upper-bound:
  $escaped(p) = exp[\forall c^i \in collectable(p).size(c^i) \mapsto 0]$.

$$\begin{aligned}
\check{g}(n) =& size(\mathrm{Integer}^3)\\
\check{h}(n) =& size(\mathrm{Long}^4)\\
\check{f}(n) =& 0\\
\check{m}(n) =& (2^n - 1) * size(\mathrm{Tree}^1)
\end{aligned}$$

$$\begin{aligned}
collectable(\mathrm{g}) &= \{\mathrm{Integer}^2\}\\
collectable(\mathrm{h}) &= \emptyset\\
collectable(\mathrm{f}) &= \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}\\
collectable(\mathrm{m}) &= \{\mathrm{Integer}^2, \mathrm{Integer}^3, \mathrm{Long}^4\}
\end{aligned}$$

Step 2: Inference of Live Memory Bounds

- Build cost relations which capture the basic idea

$$\mathbf{peak}(\{\mathbf{m_1};\mathbf{m_2}\}) = \mathbf{max}(\mathbf{peak}(\mathbf{m_1}), \mathbf{escaped}(\mathbf{m_1}) + \mathbf{peak}(\mathbf{m_2}))$$

Step 2: Inference of Live Memory Bounds

- Build cost relations which capture the basic idea

$$\mathbf{peak}(\{\mathbf{m_1};\mathbf{m_2}\}) = \mathbf{max}(\mathbf{peak}(\mathbf{m_1}), \mathbf{escaped}(\mathbf{m_1}) + \mathbf{peak}(\mathbf{m_2}))$$

- Consider a rule $\boxed{\mathrm{p{::=}g,b_1,\ldots,b_n}}$. Its *peak consumption* equation is $peak(p) = \mathcal{T}(b_1,\ldots,b_n),\ \varphi_r$ where $\mathcal{T}$ is defined as follows:

Step 2: Inference of Live Memory Bounds

- Build cost relations which capture the basic idea

$$\mathbf{peak}(\{\mathbf{m_1};\mathbf{m_2}\}) = \mathbf{max}(\mathbf{peak}(\mathbf{m_1}), \mathbf{escaped}(\mathbf{m_1}) + \mathbf{peak}(\mathbf{m_2}))$$

- Consider a rule $\boxed{\mathrm{p{::=}g,b_1,\ldots,b_n}}$. Its *peak consumption* equation is $peak(p) = \mathcal{T}(b_1,\ldots,b_n),\ \varphi_r$ where $\mathcal{T}$ is defined as follows:

peak cost relation

$\mathcal{T}(b_1,\ldots,b_n)::=$

- if b_1 is a call, then $max(peak(b_1), escaped(b_1) + \mathcal{T}(b_2,\ldots,b_n))$
- if b_1 is an instruction, then $total(b_1) + \mathcal{T}(b_2,\ldots,b_n)$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) +$$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_m(n-1), escaped_m(n-1) +$$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_f(n), escaped_f(n))))$$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_f(n), escaped_f(n))))$$

- We remove the max operators and replace esc_m by the formula:

$$peak_m(n) = size(\mathrm{Tree}^1) + peak_m(n-1)$$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_f(n), escaped_f(n))))$$

- We remove the max operators and replace esc_m by the formula:

$$peak_m(n) = size(\mathrm{Tree}^1) + peak_m(n-1)$$
$$peak_m(n) = size(\mathrm{Tree}^1) + esc_m(n-1) + peak_m(n-1)$$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$\begin{aligned} peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) + \\ max(peak_m(n-1), escaped_m(n-1) + \\ max(peak_f(n), escaped_f(n)) \;\;) \;\;) \end{aligned}$$

- We remove the max operators and replace esc_m by the formula:

$$peak_m(n) = size(\mathrm{Tree}^1) + peak_m(n-1)$$
$$peak_m(n) = size(\mathrm{Tree}^1) + esc_m(n-1) + peak_m(n-1)$$
$$peak_m(n) = size(\mathrm{Tree}^1) + esc_m(n-1) + esc_m(n-1) + peak_f(n)$$

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$peak_m(n) = size(\mathrm{Tree}^1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_m(n-1), escaped_m(n-1) + max(peak_f(n), escaped_f(n))))$$

- We remove the max operators and replace esc_m by the formula:

$$peak_m(n) = size(\mathrm{Tree}^1) + peak_m(n-1)$$
$$peak_m(n) = size(\mathrm{Tree}^1) + esc_m(n-1) + peak_m(n-1)$$
$$peak_m(n) = size(\mathrm{Tree}^1) + esc_m(n-1) + esc_m(n-1) + peak_f(n)$$

- An upper bound of the equations:

Detailed Example

- For the following code in m:

```
new Tree(m(n-1),m(n-1),f(n));
```

- We produce a peak consumption relation:

$$\begin{aligned} peak_m(n) = size(\mathrm{Tree}^1) + \; & max(peak_m(n-1), escaped_m(n-1) + \\ & max(peak_m(n-1), escaped_m(n-1) + \\ & max(peak_f(n), escaped_f(n)) \;\;) \;\;) \end{aligned}$$

- We remove the max operators and replace esc_m by the formula:

$$\begin{aligned} peak_m(n) &= size(\mathrm{Tree}^1) + peak_m(n-1) \\ peak_m(n) &= size(\mathrm{Tree}^1) + esc_m(n-1) + peak_m(n-1) \\ peak_m(n) &= size(\mathrm{Tree}^1) + esc_m(n-1) + esc_m(n-1) + peak_f(n) \end{aligned}$$

- An upper bound of the equations:

$$\boxed{peak_m(n) = 2^n * size(\mathrm{Tree}^1) + peak_f(n)}$$

Live Heap Bounds vs. Total Memory Bounds

solutions computed from peak cost relation

```
new Tree(m(n-1),m(n-1),f(n));
```

Live Heap Bounds vs. Total Memory Bounds

solutions computed from peak cost relation

```
new Tree(m(n-1),m(n-1),f(n));
```

$$total(m) = 2^n * (\underbrace{total(f)}_{exp\ \ times} + size(\mathrm{Tree}^1))$$

Live Heap Bounds vs. Total Memory Bounds

solutions computed from peak cost relation

```
new Tree(m(n-1),m(n-1),f(n));
```

$$total(m) = 2^n * (\underbrace{total(f)}_{exp\ \ times} + size(\mathrm{Tree}^1))$$

$$peak(m) = 2^n * size(\mathrm{Tree}^1) + \overbrace{peak(f)}^{only\ \ once}$$

Live Heap Bounds vs. Total Memory Bounds

solutions computed from peak cost relation

```
new Tree(m(n-1),m(n-1),f(n));
```

$$total(m) = 2^n * (\underbrace{total(f)}_{exp\ \ times} + size(\mathrm{Tree}^1))$$

$$peak(m) = 2^n * size(\mathrm{Tree}^1) + \overbrace{peak(f)}^{only\ \ once}$$

tighter bound for *f*

```
for (; n>1 ; n=n/2 ) a += g(n).intValue();
```

Live Heap Bounds vs. Total Memory Bounds

solutions computed from peak cost relation

```
new Tree(m(n-1),m(n-1),f(n));
```

$$total(m) = 2^n * (\underbrace{total(f)}_{exp\ \ times} + size(\mathrm{Tree}^1))$$

$$peak(m) = 2^n * size(\mathrm{Tree}^1) + \overbrace{peak(f)}^{only\ \ once}$$

tighter bound for *f*

```
for (; n>1 ; n=n/2 ) a += g(n).intValue();
```

$$total(f_c) = \ log(n) * (size(\mathrm{Integer}^3) + size(\mathrm{Integer}^2))$$

Live Heap Bounds vs. Total Memory Bounds

solutions computed from peak cost relation

```
new Tree(m(n-1),m(n-1),f(n));
```

$$total(m) = 2^n * (\underbrace{total(f)}_{exp\ \ times} + size(\mathrm{Tree}^1))$$

$$peak(m) = 2^n * size(\mathrm{Tree}^1) + \overbrace{peak(f)}^{only\ \ once}$$

tighter bound for f

```
for (; n>1 ; n=n/2 ) a += g(n).intValue();
```

$$total(f_c) = log(n) * (size(\mathrm{Integer}^3) + size(\mathrm{Integer}^2))$$
$$peak(f_c) = log(n) * size(\mathrm{Integer}^3) + size(\mathrm{Integer}^2)$$

Conclusions

- We have presented an automatic live heap space analysis for garbage-collected languages.
- It generates at compile-time cost relations which define the peak consumption of a program as a function of its input data size.
- The CRs obtained can be solved with standard UBs solvers (PUBS).
- We have successfully analyzed the JOlden benchmark suite.
- Applications: verification, certification (embedded systems, critical/real-time apps), program optimization and understanding, etc.

Future Work

- Adapt our techniques to region-based garbage collection
- The idea could be used to estimate other non-accumulative resources.

Info on Costa:

`http://costa.ls.fi.upm.es`