Resource Usage Analysis and its Application to Resource Certification (Part II)

Germán Puebla[1]

joint work with

Elvira Albert[2], Puri Arenas[2],
Samir Genaim[2], and Damiano Zanardini[1]

[1] Technical University of Madrid, Spain

[2] Complutense University of Madrid, Spain

September 3–4 2009

Contents of the Lecture

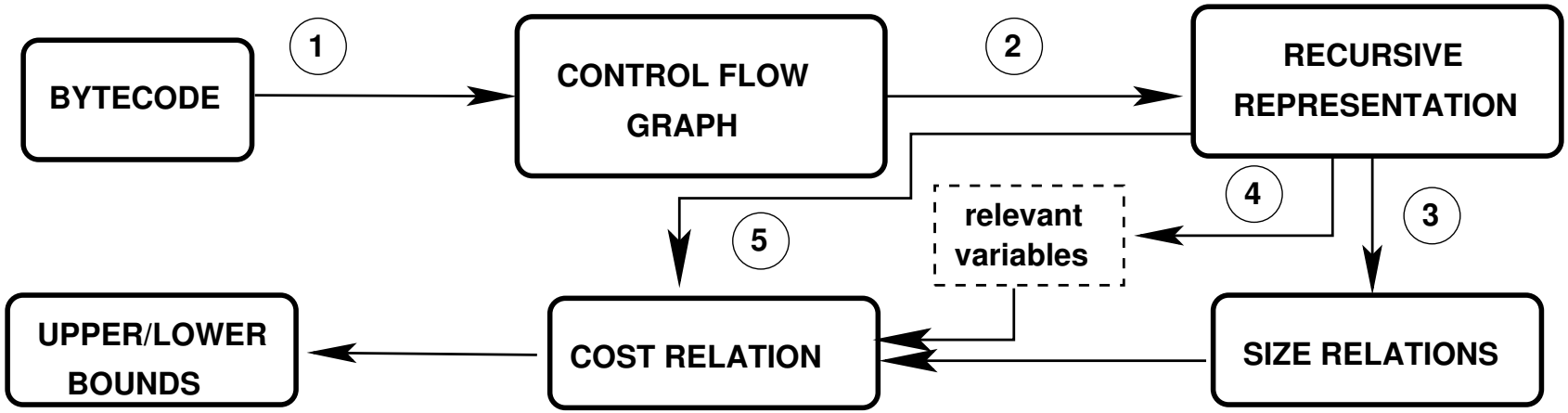
- From Java bytecode to Cost Relations
- Value Analysis of Numeric Fields
- Some Properties of Cost Relations
- From Cost Relations to Closed Forms
- Comparing Upper Bounds
- Obtaining Asymptotic Upper Bounds

The Standard Approach to Cost Analysis (Again)

The standard approach to cost analysis consists of:

1. expressing the cost of a program part in terms of other program parts, thus obtaining *recurrence relations*
2. solving the relations by obtaining a *closed-form* for the cost in terms of the input arguments

From Java bytecode to Cost Relations



- Several sources for branching: conditions, virtual methods, exceptions.
- Unstructured control flow: use of `goto`.
- JVM is a stack-based abstract machine.
- Identifying program variables that might affect the cost and eliminate other from the cost equations.

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

Recovering the Structure of a Java Bytecode Program

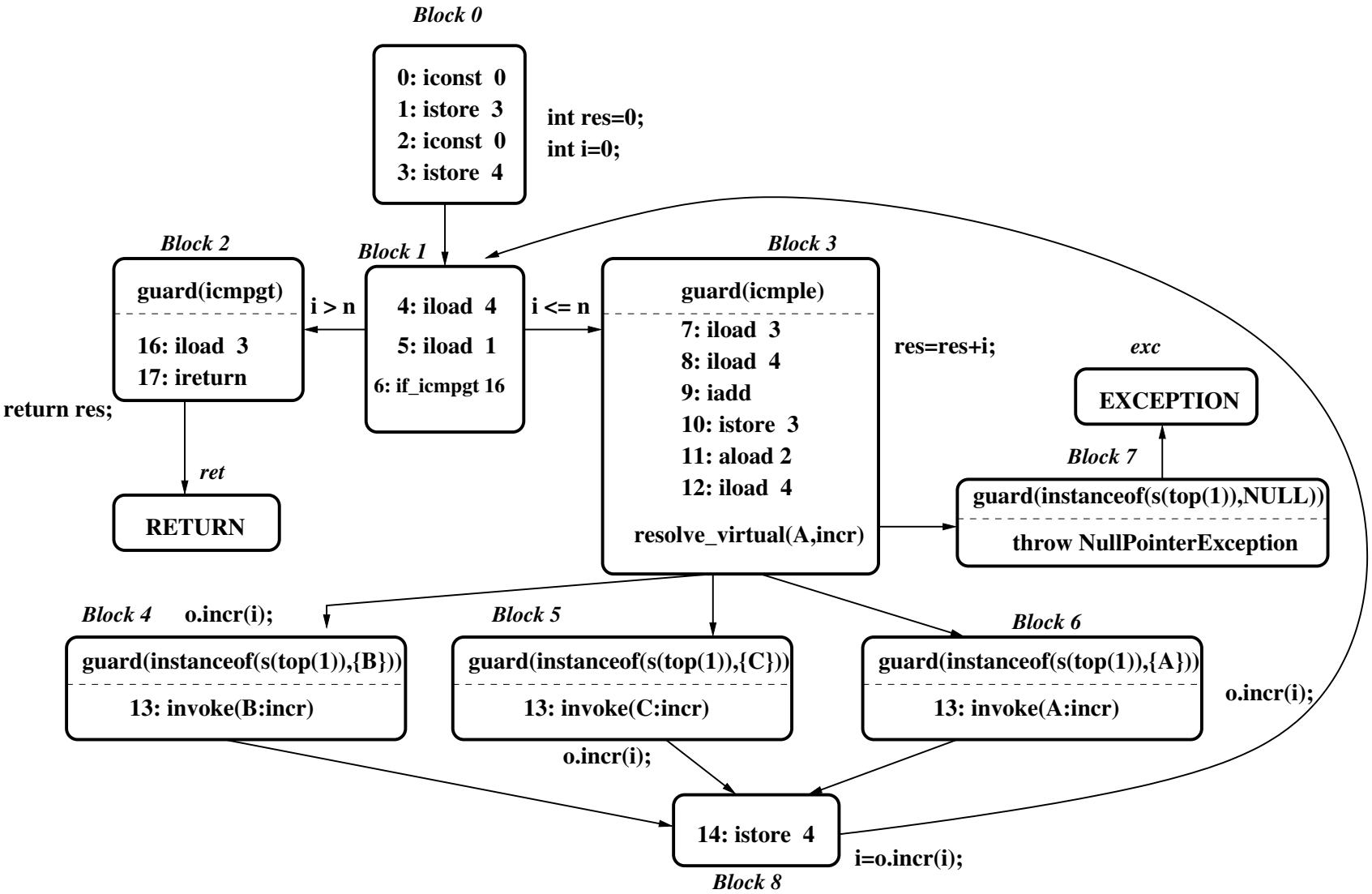
Java

```
class A {
  int incr(int i) { return i+1;}
};
class B extends A {
  int incr(int i) { return i+2;}
};
class C extends B {
  int incr(int i) { return i+3;}
};
class Main {
  int add(int n,A o) {
    int res=0; int i=0;
    while (i<=n) {
      res=res+i;i=o.incr(i);
    }
    return res;
  }
};
```

Java Bytecode of Main.add

```
 0:  iconst 0
 1:  istore 3
 2:  iconst 0
 3:  istore 4
 4:  iload 4
 5:  iload 1
 6:  if_icmpgt 16
 7:  iload 3
 8:  iload 4
 9:  iadd
10:  istore 3
11:  aload 2
12:  iload 4
13:  invokevirtual A.incr:(I)I
14:  istore 4
15:  goto 4
16:  iload 3
17:  ireturn
```

The Control Flow Graph of Main.add



Intermediate Representation of CFG

Block 3

```
guard(icmple)
---
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

Intermediate Representation of CFG

Block 3

```
guard(icmple)
---------------
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])$:-

Intermediate Representation of CFG

Block 3

guard(icmple)

```
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}$$
$$\texttt{guard(icmple}(s_0,s_1)\texttt{),}$$

Intermediate Representation of CFG

Block 3

```
guard(icmple)
---------------
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}$$
$$\texttt{guard(icmple}(s_0,s_1)\texttt{)},$$
$$\texttt{iload}(\ell_3,s_0')\texttt{,}$$
$$\texttt{iload}(\ell_4,s_1')\texttt{,}$$

Intermediate Representation of CFG

Block 3

```
guard(icmple)
---------------
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}$$
$$\texttt{guard(icmple}(s_0,s_1)),$$
$$\texttt{iload}(\ell_3,s_0'),$$
$$\texttt{iload}(\ell_4,s_1'),$$
$$\texttt{iadd}(s_0',s_1',s_0''),$$

Intermediate Representation of CFG

Block 3

```
guard(icmple)
---------------
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

$$
\begin{aligned}
&\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])\ \texttt{:-}\\
&\quad \texttt{guard}(\texttt{icmple}(s_0,s_1)),\\
&\quad \texttt{iload}(\ell_3,s_0'),\\
&\quad \texttt{iload}(\ell_4,s_1'),\\
&\quad \texttt{iadd}(s_0',s_1',s_0''),\\
&\quad \texttt{istore}(s_0'',\ell_3'),\\
&\quad \texttt{aload}(\ell_2,s_0'''),\\
&\quad \texttt{iload}(\ell_4,s_1'''),\\
&\quad (\texttt{Main.add.4}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r]);\\
&\quad\ \texttt{Main.add.5}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r]);\\
&\quad\ \texttt{Main.add.6}(\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1''',[r])).
\end{aligned}
$$

Intermediate Representation of CFG

Block 3

```
guard(icmple)
---------------
iload 3
iload 4
iadd
istore 3
aload 2
iload 4
```

Rule for Block 3

$$
\begin{aligned}
&\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}\\
&\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)},\\
&\quad \texttt{iload}(\ell_3,s_0')\texttt{,}\\
&\quad \texttt{iload}(\ell_4,s_1')\texttt{,}\\
&\quad \texttt{iadd}(s_0',s_1',s_0'')\texttt{,}\\
&\quad \texttt{istore}(s_0'',\ell_3')\texttt{,}\\
&\quad \texttt{aload}(\ell_2,s_0''')\texttt{,}\\
&\quad \texttt{iload}(\ell_4,s_1''')\texttt{,}\\
&\quad (\texttt{Main.add.4}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{;}\\
&\quad\ \texttt{Main.add.5}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{;}\\
&\quad\ \texttt{Main.add.6}(\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1''',[r]))\texttt{.}
\end{aligned}
$$

Block 4

```
guard(instanceof(
      s(top(1)),B))
---------------
invoke B.incr
```

Rule for Block 4

$$
\begin{aligned}
&\texttt{Main.add.4}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}\\
&\quad \texttt{guard(instanceof}(s_0,\texttt{B})\texttt{)},\\
&\quad \texttt{B.incr}([s_0,s_1],[s_0'])\texttt{,}\\
&\quad \texttt{Main.add.8}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0'],[r])\texttt{.}
\end{aligned}
$$

Intermediate Representations - Advantages

Rules for Blocks 3 and 4

```
Main.add.3([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0,s1],[r]) :-
  guard(icmple(s0,s1)), iload(ℓ3,s0'), iload(ℓ4,s1'), iadd(s0',s1',s0''),
  istore(s0'',ℓ3'),   aload(ℓ2,s0'''), iload(ℓ4,s1'''),
  (Main.add.4([ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1'''],[r]);
   Main.add.5([ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1'''],[r]);
   Main.add.6(ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1''',[r])).

Main.add.4([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0,s1],[r]) :-
  guard(instanceof(s0,B)), B.incr([s0,s1],[s0']),
  Main.add.8([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0'],[r]).
```

- Stack elements are like local variables, moreover many of them can be eliminated.

Intermediate Representations - Advantages

Rules for Blocks 3 and 4

```
Main.add.3([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0,s1],[r]) :-
  guard(icmple(s0,s1)), iload(ℓ3,s0'), iload(ℓ4,s1'), iadd(s0',s1',s0''),
  istore(s0'',ℓ3'),   aload(ℓ2,s0'''), iload(ℓ4,s1'''),
  (Main.add.4([ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1'''],[r]);
   Main.add.5([ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1'''],[r]);
   Main.add.6(ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1''',[r])).

Main.add.4([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0,s1],[r]) :-
  guard(instanceof(s0,B)), B.incr([s0,s1],[s0']),
  Main.add.8([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0'],[r]).
```

- Stack elements are like local variables, moreover many of them can be eliminated.
- All loops (for, while, recursive calls, etc) are represented in the same setting.

Intermediate Representations - Advantages

Rules for Blocks 3 and 4

```
Main.add.3([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0,s1],[r]) :-
  guard(icmple(s0,s1)), iload(ℓ3,s0'), iload(ℓ4,s1'), iadd(s0',s1',s0''),
  istore(s0'',ℓ3'),   aload(ℓ2,s0'''), iload(ℓ4,s1'''),
  (Main.add.4([ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1'''],[r]);
   Main.add.5([ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1'''],[r]);
   Main.add.6(ℓ0,ℓ1,ℓ2,ℓ3',ℓ4,s0''',s1''',[r])).

Main.add.4([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0,s1],[r]) :-
  guard(instanceof(s0,B)), B.incr([s0,s1],[s0']),
  Main.add.8([ℓ0,ℓ1,ℓ2,ℓ3,ℓ4,s0'],[r]).
```

- Stack elements are like local variables, moreover many of them can be eliminated.
- All loops (for, while, recursive calls, etc) are represented in the same setting.
- Defining the cost of one block (method) in terms of other block is straightforward.

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{aligned}
&\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}\\
&\quad \texttt{guard(icmple}(s_0,s_1)),\\
&\quad \texttt{iload}(\ell_3,s_0'),\\
&\quad \texttt{iload}(\ell_4,s_1'),\\
&\quad \texttt{iadd}(s_0',s_1',s_0''),\\
&\quad \texttt{istore}(s_0'',\ell_3'),\\
&\quad \texttt{aload}(\ell_2,s_0'''),\\
&\quad \texttt{iload}(\ell_4,s_1'''),\\
&\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r]).
\end{aligned}
$$

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{aligned}
&\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}\\
&\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)},\ s_0 \leq s_1 \wedge\\
&\quad \texttt{iload}(\ell_3,s_0')\texttt{,}\\
&\quad \texttt{iload}(\ell_4,s_1')\texttt{,}\\
&\quad \texttt{iadd}(s_0',s_1',s_0'')\texttt{,}\\
&\quad \texttt{istore}(s_0'',\ell_3')\texttt{,}\\
&\quad \texttt{aload}(\ell_2,s_0''')\texttt{,}\\
&\quad \texttt{iload}(\ell_4,s_1''')\texttt{,}\\
&\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{.}
\end{aligned}
$$

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{array}{l}
\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-} \\
\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)},\ \ s_0 \leq s_1 \wedge \\
\quad \texttt{iload}(\ell_3,s_0')\texttt{,}\ \ \ell_3 = s_0' \wedge \\
\quad \texttt{iload}(\ell_4,s_1')\texttt{,} \\
\quad \texttt{iadd}(s_0',s_1',s_0'')\texttt{,} \\
\quad \texttt{istore}(s_0'',\ell_3')\texttt{,} \\
\quad \texttt{aload}(\ell_2,s_0''')\texttt{,} \\
\quad \texttt{iload}(\ell_4,s_1''')\texttt{,} \\
\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{.}
\end{array}
$$

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{array}{l}
\texttt{Main.add.3([}\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1\texttt{],[}r\texttt{]) :-} \\
\quad \texttt{guard(icmple(}s_0,s_1\texttt{))}, \quad s_0 \leq s_1 \wedge \\
\quad \texttt{iload(}\ell_3,s_0'\texttt{)}, \quad \ell_3 = s_0' \wedge \\
\quad \texttt{iload(}\ell_4,s_1'\texttt{)}, \quad s_1' = \ell_4 \wedge \\
\quad \texttt{iadd(}s_0',s_1',s_0''\texttt{)}, \\
\quad \texttt{istore(}s_0'',\ell_3'\texttt{)}, \\
\quad \texttt{aload(}\ell_2,s_0'''\texttt{)}, \\
\quad \texttt{iload(}\ell_4,s_1'''\texttt{)}, \\
\quad \texttt{Main.add.4/5/6([}\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''\texttt{],[}r\texttt{])}.
\end{array}
$$

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{array}{ll}
\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-} & \\
\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)}, & s_0 \leq s_1 \wedge \\
\quad \texttt{iload}(\ell_3,s_0')\texttt{,} & \ell_3 = s_0' \wedge \\
\quad \texttt{iload}(\ell_4,s_1')\texttt{,} & s_1' = \ell_4 \wedge \\
\quad \texttt{iadd}(s_0',s_1',s_0'')\texttt{,} & s_0'' = s_0' + s_1' \wedge \\
\quad \texttt{istore}(s_0'',\ell_3')\texttt{,} & \\
\quad \texttt{aload}(\ell_2,s_0''')\texttt{,} & \\
\quad \texttt{iload}(\ell_4,s_1''')\texttt{,} & \\
\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{.} &
\end{array}
$$

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{array}{ll}
\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-} & \\
\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)}, & s_0 \leq s_1 \wedge \\
\quad \texttt{iload}(\ell_3,s_0')\texttt{,} & \ell_3 = s_0' \wedge \\
\quad \texttt{iload}(\ell_4,s_1')\texttt{,} & s_1' = \ell_4 \wedge \\
\quad \texttt{iadd}(s_0',s_1',s_0'')\texttt{,} & s_0'' = s_0' + s_1' \wedge \\
\quad \texttt{istore}(s_0'',\ell_3')\texttt{,} & \ell_3' = s_0'' \wedge \\
\quad \texttt{aload}(\ell_2,s_0''')\texttt{,} & s_0''' = \ell_2 \wedge \\
\quad \texttt{iload}(\ell_4,s_1''')\texttt{,} & s_1''' = \ell_4 \\
\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{.} &
\end{array}
$$

- Fixpoint computation.

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables:

Rule for Block 3 - Abstraction Step

$$
\begin{array}{ll}
\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-} & \\
\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)}, & s_0 \leq s_1 \wedge \\
\quad \texttt{iload}(\ell_3,s_0')\texttt{,} & \ell_3 = s_0' \wedge \\
\quad \texttt{iload}(\ell_4,s_1')\texttt{,} & s_1' = \ell_4 \wedge \\
\quad \texttt{iadd}(s_0',s_1',s_0'')\texttt{,} & s_0'' = s_0' + s_1' \wedge \\
\quad \texttt{istore}(s_0'',\ell_3')\texttt{,} & \ell_3' = s_0'' \wedge \\
\quad \texttt{aload}(\ell_2,s_0''')\texttt{,} & s_0''' = \ell_2 \wedge \\
\quad \texttt{iload}(\ell_4,s_1''')\texttt{,} & s_1''' = \ell_4 \\
\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\texttt{.} &
\end{array}
$$

- Fixpoint computation.

Size Analysis

- Size analysis aims at inferring relations between the values of the variables occur in the head, and the values of the variables at each call to another block (or method) in its body.
- Abstracting bytecodes to constraints they impose on their variables: **[SPOTO et. al.]**

Rule for Block 3 - Abstraction Step

$$
\begin{array}{ll}
\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])\ \texttt{:-} & \\
\quad \texttt{guard}(\texttt{icmple}(s_0,s_1)), & s_0 \leq s_1 \wedge \\
\quad \texttt{iload}(\ell_3,s_0'), & \ell_3 = s_0' \wedge \\
\quad \texttt{iload}(\ell_4,s_1'), & s_1' = \ell_4 \wedge \\
\quad \texttt{iadd}(s_0',s_1',s_0''), & s_0'' = s_0' + s_1' \wedge \\
\quad \texttt{istore}(s_0'',\ell_3'), & \ell_3' = s_0'' \wedge \\
\quad \texttt{aload}(\ell_2,s_0'''), & s_0''' = \ell_2 \wedge \\
\quad \texttt{iload}(\ell_4,s_1'''), & s_1''' = \ell_4 \\
\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r]). &
\end{array}
$$

- Fixpoint computation.

Translating Rules to Cost Relations

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}$$
$$\quad \texttt{guard(icmple}(s_0,s_1)\texttt{)},$$
$$\quad \mathbf{B_3}\texttt{,}$$
$$\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell'_3,\ell_4,s'''_0,s'''_1],[r]) \quad \{\ell'_3{=}\ell_3{+}\ell_4, s'''_0{=}\ell_2, s'''_1{=}\ell_4\}$$

Translating Rules to Cost Relations

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r]) \texttt{ :-}$$
$$\quad \texttt{guard(icmple}(s_0,s_1)),$$
$$\quad \mathbf{B_3},$$
$$\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell'_3,\ell_4,s'''_0,s'''_1],[r]) \quad \{\ell'_3{=}\ell_3{+}\ell_4, s'''_0{=}\ell_2, s'''_1{=}\ell_4\}$$

Translating the rule into an equations(s) that defines it cost:

Translating Rules to Cost Relations

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])\ \texttt{:-}$$
$$\quad \texttt{guard(icmple}(s_0,s_1)),$$
$$\quad \mathbf{B_3},$$
$$\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\ \{\ell_3'=\ell_3+\ell_4, s_0'''=\ell_2, s_1'''=\ell_4\}$$

Translating the rule into an equations(s) that defines it cost:

$$C_{Main.add.3}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1) =$$
$$\mathcal{M}(\mathbf{B_3}) + C_{Main.add.3_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3+\ell_4,\ell_4,\ell_2,\ell_4) \ \mid \ \textit{if} \quad s_0 \leq s_1$$

Translating Rules to Cost Relations

$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])$:-
  $\texttt{guard}(\texttt{icmple}(s_0,s_1))$,
  $\mathbf{B_3}$,
  $\texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell'_3,\ell_4,s'''_0,s'''_1],[r])$ $\{\ell'_3{=}\ell_3{+}\ell_4, s'''_0{=}\ell_2, s'''_1{=}\ell_4\}$

Translating the rule into an equations(s) that defines it cost:

$$C_{Main.add.3}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1) = \mathcal{M}(\mathbf{B_3}) + C_{Main.add.3_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3+\ell_4,\ell_4,\ell_2,\ell_4) \mid \textit{if} \quad s_0 \leq s_1$$

$$C_{Main.add.3_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1) = \begin{cases} C_{Main.add.4_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1) & \text{if } \ell_2 \in B \\ C_{Main.add.5_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1) & \text{if } \ell_2 \in C \\ C_{Main.add.6_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1) & \text{if } \ell_2 \in A \end{cases}$$

Translating Rules to Cost Relations

$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])$:-
  $\texttt{guard}(\texttt{icmple}(s_0,s_1))$,
  $\mathbf{B_3}$,
  $\texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell'_3,\ell_4,s'''_0,s'''_1],[r])$ $\{\ell'_3{=}\ell_3{+}\ell_4, s'''_0{=}\ell_2, s'''_1{=}\ell_4\}$

Translating the rule into an equations(s) that defines it cost:

$$C_{Main.add.3}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4) = \mathcal{M}(\mathbf{B_3}) + C_{Main.add.3_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3+\ell_4,\ell_4) \mid \textit{if} \quad \ell_4 \leq \ell_1$$

$$C_{Main.add.3_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4) = \begin{cases} C_{Main.add.4_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4) & \text{if } \ell_2 \in B \\ C_{Main.add.5_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4) & \text{if } \ell_2 \in C \\ C_{Main.add.6_{cont}}(\ell_0,\ell_1,\ell_2,\ell_3,\ell_4) & \text{if } \ell_2 \in A \end{cases}$$

Translating Rules to Cost Relations

$$\texttt{Main.add.3}([\ell_0,\ell_1,\ell_2,\ell_3,\ell_4,s_0,s_1],[r])\ \texttt{:-}$$
$$\quad \texttt{guard(icmple}(s_0,s_1)),$$
$$\quad \mathbf{B_3},$$
$$\quad \texttt{Main.add.4/5/6}([\ell_0,\ell_1,\ell_2,\ell_3',\ell_4,s_0''',s_1'''],[r])\ \{\ell_3'=\ell_3+\ell_4, s_0'''=\ell_2, s_1'''=\ell_4\}$$

Translating the rule into an equations(s) that defines it cost:

$$C_{Main.add.3}(\ell_1,\ell_2,\ell_4) = \mathcal{M}(\mathbf{B_3}) + C_{Main.add.3_{cont}}(\ell_1,\ell_2,\ell_4) \ \mid\ \textit{if}\ \ \ell_4 \leq \ell_1$$

$$C_{Main.add.3_{cont}}(\ell_1,\ell_2,\ell_4) = \begin{cases} C_{Main.add.4_{cont}}(\ell_1,\ell_2,\ell_4) & \text{if } \ell_2 \in B \\ C_{Main.add.5_{cont}}(\ell_1,\ell_2,\ell_4) & \text{if } \ell_2 \in C \\ C_{Main.add.6_{cont}}(\ell_1,\ell_2,\ell_4) & \text{if } \ell_2 \in A \end{cases}$$

Translating Rules to Cost Relations - Cont.

$$
\begin{array}{lcll}
C_{\mathtt{add}}(\ell_1,\ell_2) & = & C_{\mathtt{add}^0}(\ell_1,\ell_2) & \\
C_{\mathtt{add}^0}(\ell_1,\ell_2) & = & \mathtt{T}_0 + C_{\mathtt{add}^1}(\ell_1,\ell_2,\ell_4') & \ell_4' = \mathtt{0} \\
C_{\mathtt{add}^1}(\ell_1,\ell_2,\ell_4) & = & \mathtt{T}_1 + C_{\mathtt{add}^1\_\mathtt{cont}}(\ell_1,\ell_2,\ell_4) & \\
C_{\mathtt{add}^1\_\mathtt{cont}}(\ell_1,\ell_2,\ell_4) & = & \left\{ \begin{array}{l} C_{\mathtt{add}^2}() \\ C_{\mathtt{add}^3}(\ell_1,\ell_2,\ell_4) \end{array} \right. & \begin{array}{l} \ell_4 > \ell_1 \\ \ell_4 \leq \ell_1 \end{array} \\
C_{\mathtt{add}^2}() & = & \mathtt{T}_2 & \\
C_{\mathtt{add}^3}(\ell_1,\ell_2,\ell_4) & = & \mathtt{T}_3 + C_{\mathtt{add}^3\_\mathtt{cont}}(\ell_1,\ell_2,\ell_4) & \\
C_{\mathtt{add}^3\_\mathtt{cont}}(\ell_1,\ell_2,\ell_4) & = & \left\{ \begin{array}{l} C_{\mathtt{add}^4}(\ell_1,\ell_2,\ell_4) \\ C_{\mathtt{add}^5}(\ell_1,\ell_2,\ell_4) \\ C_{\mathtt{add}^6}(\ell_1,\ell_2,\ell_4) \end{array} \right. & \begin{array}{l} \ell_2 \in \mathtt{B} \\ \ell_2 \in \mathtt{C} \\ \ell_2 \in \mathtt{A} \end{array} \\
C_{\mathtt{add}^4}(\ell_1,\ell_2,\ell_4) & = & \mathtt{T}_4 + C_{\mathtt{B:incr}}(\ell_2,\ell_4) + C_{\mathtt{add}^8}(\ell_1,\ell_2,\mathtt{s}_0) & \mathtt{s}_0 = \ell_4 + 2 \\
C_{\mathtt{add}^5}(\ell_1,\ell_2,\ell_4) & = & \mathtt{T}_5 + C_{\mathtt{C:incr}}(\ell_2,\ell_4) + C_{\mathtt{add}^8}(\ell_1,\ell_2,\mathtt{s}_0) & \mathtt{s}_0 = \ell_4 + 3 \\
C_{\mathtt{add}^6}(\ell_1,\ell_2,\ell_4) & = & \mathtt{T}_6 + C_{\mathtt{A:incr}}(\ell_2,\ell_4) + C_{\mathtt{add}^8}(\ell_1,\ell_2,\mathtt{s}_0) & \mathtt{s}_0 = \ell_4 + 1 \\
C_{\mathtt{add}^8}(\ell_1,\ell_2,\mathtt{s}_0) & = & \mathtt{T}_8 + C_{\mathtt{add}^1}(\ell_1,\ell_2,\mathtt{s}_0) &
\end{array}
$$

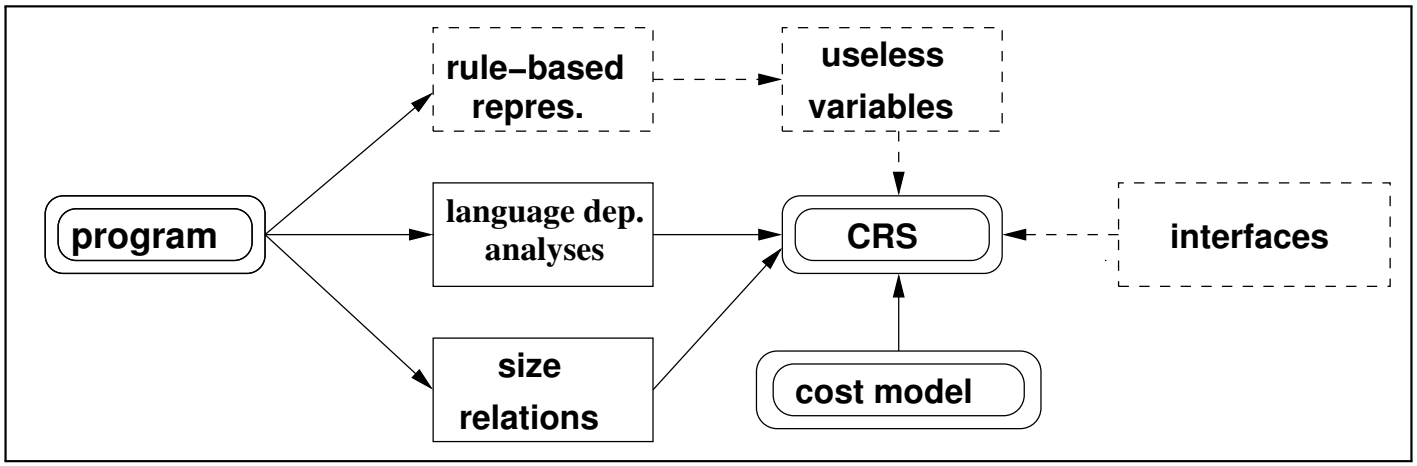
Value Analysis of Numeric Fields: Accuracy of Results

Bench.	**Ru**	$\mathbf{L}_n$	$\mathbf{R}_s$	$\mathbf{R}_i$
lang	315	13	13	0
util	685	24	24	0
beans	90	3	3	0
math	662	15	12	1
text	1743	24	20	1
awt	4524	90	87	0
io	716	6	5	2
security	58	1	1	0
total	8793	176	165	4

Value Analysis of Numeric Fields: Overhead Introduced

Bench.	$\mathbf{T}_{rca}$	$\mathbf{T}_{tr}$	$\mathbf{T}_{gh}$	$\mathbf{T}_{i}$	$\mathbf{T}_{s}$	**SD**
lang	0.12	0.01	0.02	3.33	5.47	1.64
util	0.58	7.88	4.90	20.21	39.36	1.95
beans	0.05	0.00	0.00	1.42	1.65	1.16
math	0.22	0.18	0.17	7.84	9.84	1.26
text	0.79	0.34	0.37	37.33	141.04	3.78
awt	2.44	7.56	7.59	98.56	248.49	2.52
io	0.61	0.49	0.27	17.79	23.94	1.35
security	0.03	0.01	0.00	0.90	0.98	1.09
total	4.84	16.47	13.32	187.38	470.77	2.51

Cost Relations: A Unified View



An Example

Prolog

```
merge(This,[ ],R):- !,R = This.
merge([D|Next],[OD|ONext],R):-
   D>OD, !, R = [OD|T],
   merge([D|Next],ONext,T).
merge([D],O,R):- !,R = [D|O].
merge([D|Next],O,R):-
   R = [D|T], merge(Next,O,T).
```

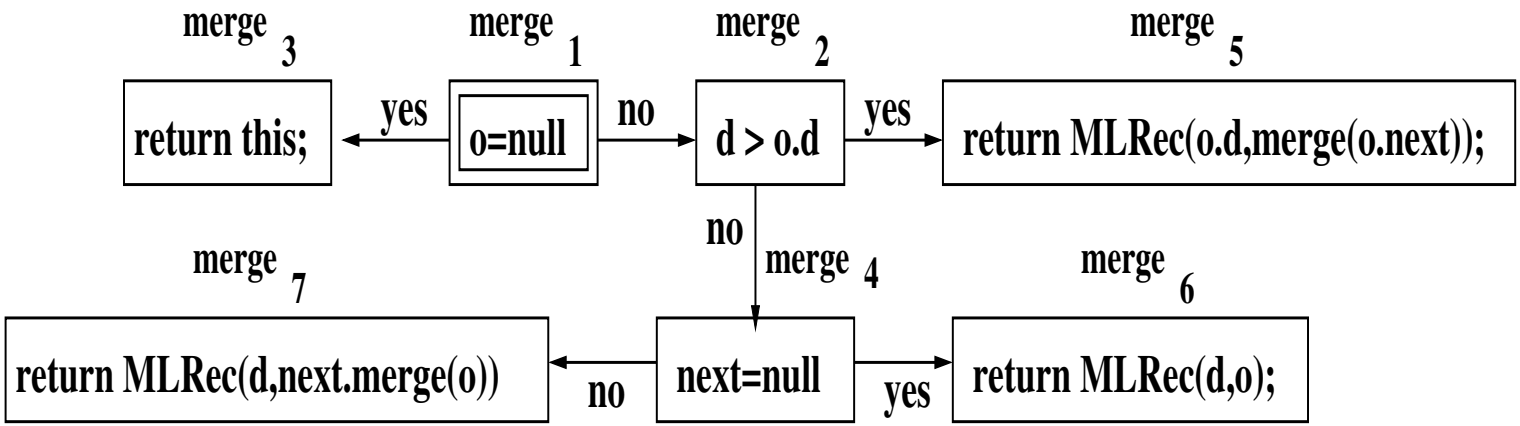
Haskell

```
merge this [ ] = this
merge (d:next) o =
  if (d> od) then
    (od: merge (d:next) onext)
  else if (next==[ ]) then (d:o)
       else (d:merge next o)
  where (od:onext) = o
```

Java (recursive)

```
public class MLRec {
private int d;
private MLRec next;
public MLRec(int d, MLRec next){
  this.d = d;
  this.next = next;
}
public MLRec merge(MLRec o) {
  if (o == null) return this;      (1)m
  else if (d>o.d)
   return
       new MLRec(o.d,merge(o.next)); (2)m
  else if (next = null)
   return new MLRec(d,o);  (3)m
  else return
   new MLRec(d,next.merge(o)); (4)m
}
```

An Example (Cont.)



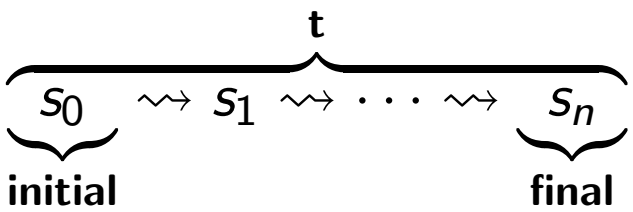
$(1)_m$ $merge(this, o)=k_1$	$\{this \geq 1, o=0\}$
$(2)_m$ $merge(this, o)=k_3+merge(this, o')$	$\{this \geq 1, o \geq 1, o>o', o' \geq 0\}$
$(3)_m$ $merge(this, o)=k_2$	$\{this=1, o \geq 1\}$
$(4)_m$ $merge(this, o)=k_4+merge(this', o)$	$\{this \geq 2, this>this', this' \geq 0, o \geq 1\}$
$\mathcal{M}_{\text{ninst}}$	$k_1=4, k_2=26, k_3=26, k_4=29$

Execution Cost

- State transition system **(STS)**: A set of states Σ and a binary relation $\rightsquigarrow \subseteq \Sigma \times \Sigma$.

$$s_i \rightsquigarrow s_j$$

- A **state** is final iff it has no successors.
- A **trace** has the form:



- Cost: We assign different *labels* ($l \in \mathcal{L}$) to different transitions. Labels contain the observable information to measure the cost.

$$s_i \rightsquigarrow_l s_j$$

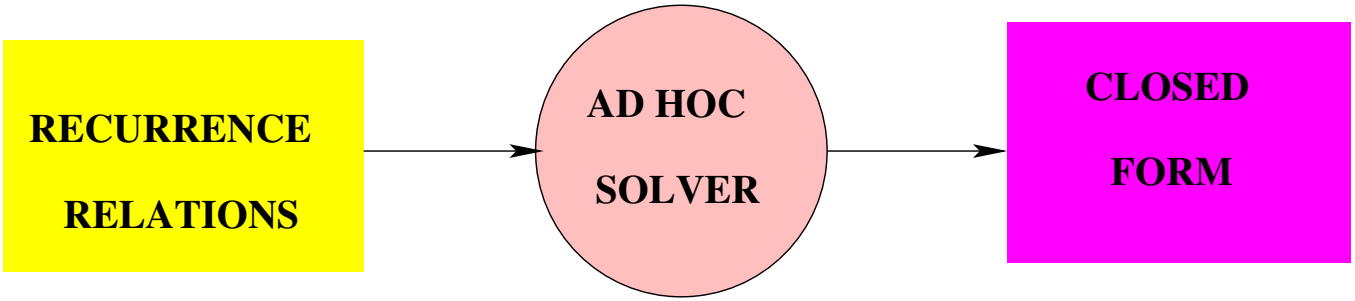
Execution Model

- A Cost Model is a function $\mathcal{M} : \mathcal{L} \rightarrow \mathbb{Q}^+$.
- Given a trace $t \equiv s_0 \rightsquigarrow_{l_0} s_1 \rightsquigarrow_{l_1} \ldots s_{n-1} \rightsquigarrow_{l_{n-1}} s_n$, we define the cost of t as the cost of the corresponding labels:

$$Cost(t, \mathcal{M}) = \mathcal{M}(l_0) + \ldots + \mathcal{M}(l_{n-1})$$

- As an example, for Java Bytecode: labels can be bytecode instructions, and possible cost models are:
  - $\mathcal{M}(l) = 1$ (count the number of bytecode instructions executed).
  - $\mathcal{M}(l) = v * size(c)$, if $\mathtt{l} = \mathtt{new\ c}$ or $\mathtt{l} = \mathtt{newarray}\ c$ and v is the length of the array (count the amount of heap allocated during the execution).
  - $\mathcal{M}(l) = 1$, if $\mathtt{l} = \mathtt{invokevirtual\ m}$ (counts the number of calls to the method m).

From CRs to Closed Forms: Approach 1

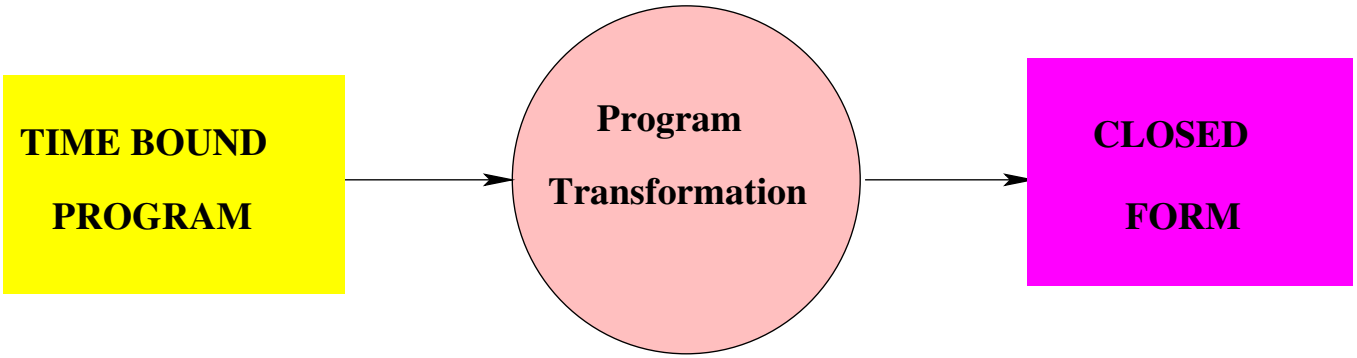


Ad-hoc Solver

consists in implementing ad-hoc solvers based on mathematical techniques for recurrence relations [Wegbreit 75, Debray and Lin'93].

- It is a simple approach, but
- covers a rather limited class of recursions
- produces too large expressions
- a relevant case is the PURRS system [Bagnara et al'05]
  - computes upper bounds $\longrightarrow$ simpler expressions, but
  - still limited to recurrence relations

From CRs to Closed Forms: Approach 2

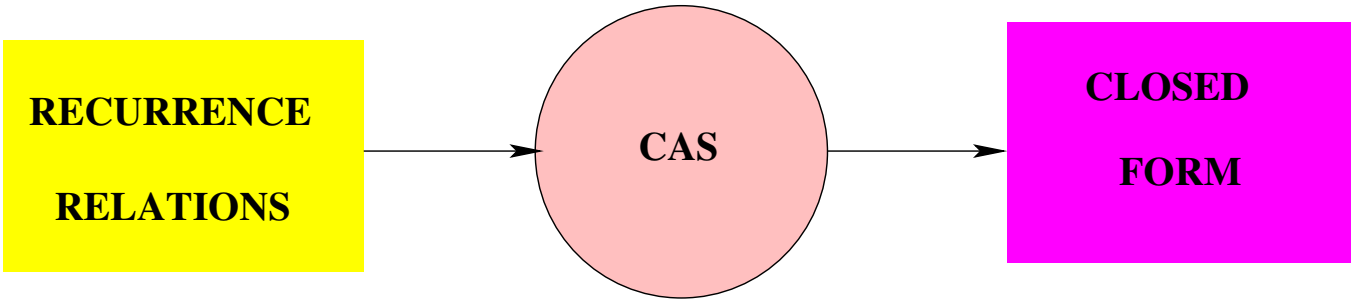


Program transformation approach

consists in regarding recurrence relations as programs and try to transform them into equivalent, non-recursive programs.
[Le Metayer'88, Rosendahl'89]

- Appealing approach, but
- can only cover some classes of recursion

From CRs to Closed Forms: Approach 3

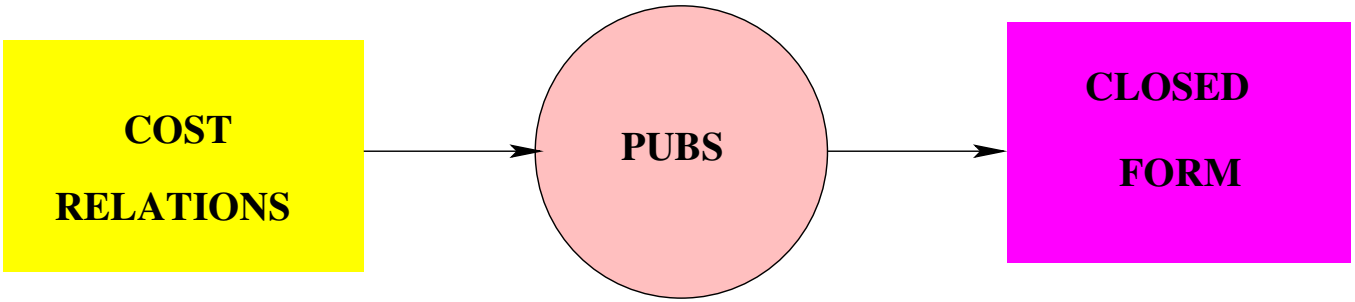


CAS: **C**omputer **A**lgebra **S**ystems

consists in relying on Computer Algebra Systems (Mathematica, Maple, Maxima,...) [Wadler'88, Sands'95, Benzinger'04, Albert et al'07]

- Obtains precise solutions, but
- Not always applicable
- Requires human intervention
- Produces too large expressions

From CRs to Closed Forms: Our Approach



PUBS: **P**ractical **U**pper **B**ound **S**olver

The main features of our approach are:

- Regard cost relations as (nondeterministic) programs.
- Provide an operational semantics based on evaluation trees.
- Reason on the shape and contents of evaluation trees.
- Perform partial evaluation for obtaining direct recursion.
- Compute ranking functions for bounding the height of trees.
- Obtain loop invariants for maximizing expressions.

Cost Analysis does not Produce Recurrence Relations

(a) Not mutually exclusive

Equations are not mutually exclusive. Even if the original program is deterministic, as a result of abstraction.

Cost Analysis does not Produce Recurrence Relations

(a) Not mutually exclusive

Equations are not mutually exclusive. Even if the original program is deterministic, as a result of abstraction.

(b) Inexact Size-Relations

Dealing with real-life programs, non-linear data structures results in constraints rather than exact values.

Cost Analysis does not Produce Recurrence Relations

(a) Not mutually exclusive

Equations are not mutually exclusive. Even if the original program is deterministic, as a result of abstraction.

(b) Inexact Size-Relations

Dealing with real-life programs, non-linear data structures results in constraints rather than exact values.

(c) Multiple-Arguments

Usually, relations have several input arguments. The number of iterations may depend on several of them.

Running Example

```
void del(List l, int p, int a[], int la, int b[], int lb){
  while (l!=null) {
    if (l.data<p) {
      la=rm_vec(l.data, a, la);
    } else {
      lb=rm_vec(l.data, b, lb);
    }
    l=l.next;
  }
}
int rm_vec(int e, int a[], int la){
  int i=0;
  while (i<la && a[i]<e)  i++;
  for (int j=i; j<la−1; j++) a[j]=a[j+1];
  return la−1;
}
```

(1) $Del(l, a, la, b, lb)=1+C(l, a, la, b, lb)$
$\{b\geq lb, lb\geq 0, a\geq la, la\geq 0, l\geq 0\}$
(2) $C(l, a, la, b, lb)=2$ $\{a\geq la, b\geq lb, b\geq 0, a\geq 0, l=0\}$
(3) $C(l, a, la, b, lb)=$
$25+D(a, la, 0)+E(la, j)+C(l', a, la-1, b, lb)$
$\{a\geq 0, a\geq la, b\geq lb, j\geq 0, b\geq 0, l>l', l>0\}$
(4) $C(l, a, la, b, lb)=$
$24+D(b, lb, 0)+E(lb, j)+C(l', a, la, b, lb-1)$
$\{a\geq 0, a\geq la, b\geq lb, j\geq 0, b\geq 0, l>l', l>0\}$
(5) $D(a, la, i)=3$ $\{i\geq la, a\geq la, i\geq 0\}$
(6) $D(a, la, i)=8$ $\{i<la, a\geq la, i\geq 0\}$
(7) $D(a, la, i)=10+D(a, la, i+1)$ $\{i<la, a\geq la, i\geq 0\}$
(8) $E(la, j)=5$ $\{j\geq la-1, j\geq 0\}$
(9) $E(la, j)=15+E(la, j+1)$ $\{j<la-1, j\geq 0\}$

Cost Relations

- Arguments are replaced by their *size*
- Cost relations return the execution cost
- There is a recursive relation per loop in the original program

(a) Not mutually exclusive

(1) $Del(l, a, la, b, lb) = 1 + C(l, a, la, b, lb)$
$\{b \geq lb, lb \geq 0, a \geq la, la \geq 0, l \geq 0\}$

(2) $C(l, a, la, b, lb) = 2$ $\{a \geq la, b \geq lb, b \geq 0, a \geq 0, l = 0\}$

(3) $C(l, a, la, b, lb) =$
$25 + D(a, la, 0) + E(la, j) + C(l', a, la - 1, b, lb)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(4) $C(l, a, la, b, lb) =$
$24 + D(b, lb, 0) + E(lb, j) + C(l', a, la, b, lb - 1)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(5) $D(a, la, i) = 3 \quad \{i \geq la, a \geq la, i \geq 0\}$

(6) $D(a, la, i) = 8 \quad \{i < la, a \geq la, i \geq 0\}$

(7) $D(a, la, i) = 10 + D(a, la, i + 1)$ $\{i < la, a \geq la, i \geq 0\}$

(8) $E(la, j) = 5 \quad \{j \geq la - 1, j \geq 0\}$

(9) $E(la, j) = 15 + E(la, j + 1) \quad \{j < la - 1, j \geq 0\}$

(a) Not mutually exclusive

(1) $Del(l, a, la, b, lb) = 1 + C(l, a, la, b, lb)$
$\{b \geq lb, lb \geq 0, a \geq la, la \geq 0, l \geq 0\}$

(2) $C(l, a, la, b, lb) = 2$ $\{a \geq la, b \geq lb, b \geq 0, a \geq 0, l = 0\}$

(3) $C(l, a, la, b, lb) =$
$25 + D(a, la, 0) + E(la, j) + C(l', a, la-1, b, lb)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(4) $C(l, a, la, b, lb) =$
$24 + D(b, lb, 0) + E(lb, j) + C(l', a, la, b, lb-1)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(5) $D(a, la, i) = 3$ $\{i \geq la, a \geq la, i \geq 0\}$

(6) $D(a, la, i) = 8$ $\{i < la, a \geq la, i \geq 0\}$

(7) $D(a, la, i) = 10 + D(a, la, i+1)$ $\{i < la, a \geq la, i \geq 0\}$

(8) $E(la, j) = 5$ $\{j \geq la-1, j \geq 0\}$

(9) $E(la, j) = 15 + E(la, j+1)$ $\{j < la-1, j \geq 0\}$

(a) Not mutually exclusive

(1) $Del(l, a, la, b, lb) = 1 + C(l, a, la, b, lb)$
$\{b \geq lb, lb \geq 0, a \geq la, la \geq 0, l \geq 0\}$

(2) $C(l, a, la, b, lb) = 2 \ \{a \geq la, b \geq lb, b \geq 0, a \geq 0, l = 0\}$

(3) $C(l, a, la, b, lb) =$
$25 + D(a, la, 0) + E(la, j) + C(l', a, la-1, b, lb)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(4) $C(l, a, la, b, lb) =$
$24 + D(b, lb, 0) + E(lb, j) + C(l', a, la, b, lb-1)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(5) $D(a, la, i) = 3 \quad \{i \geq la, a \geq la, i \geq 0\}$

(6) $D(a, la, i) = 8$

(7) $D(a, la, i) = 10 + D(a, la, i+1)$

(8) $E(la, j) = 5 \quad \{j \geq la-1, j \geq 0\}$

(9) $E(la, j) = 15 + E(la, j+1) \quad \{j < la-1, j \geq 0\}$

Important observation

The upper bound is lost if we remove equations!

(b) Inexact Size Relations

(1) $Del(l, a, la, b, lb) = 1 + C(l, a, la, b, lb)$
$\{b \geq lb, lb \geq 0, a \geq la, la \geq 0, l \geq 0\}$

(2) $C(l, a, la, b, lb) = 2 \ \{a \geq la, b \geq lb, b \geq 0, a \geq 0, l = 0\}$

(3) $C(l, a, la, b, lb) =$
$25 + D(a, la, 0) + E(la, j) + C(l', a, la-1, b, lb)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(4) $C(l, a, la, b, lb) =$
$24 + D(b, lb, 0) + E(lb, j) + C(l', a, la, b, lb-1)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(5) $D(a, la, i) = 3 \quad \{i \geq la, a \geq la, i \geq 0\}$

(6) $D(a, la, i) = 8 \quad \{i < la, a \geq la, i \geq 0\}$

(7) $D(a, la, i) = 10 + D(a, la, i+1) \{i < la, a \geq la, i \geq 0\}$

(8) $E(la, j) = 5 \quad \{j \geq la-1, j \geq 0\}$

(9) $E(la, j) = 15 + E(la, j+1) \quad \{j < la-1, j \geq 0\}$

(c) Multiple Arguments

(1) $Del(l, a, la, b, lb) = 1 + C(l, a, la, b, lb)$
$\{b \geq lb, lb \geq 0, a \geq la, la \geq 0, l \geq 0\}$

(2) $C(l, a, la, b, lb) = 2 \ \{a \geq la, b \geq lb, b \geq 0, a \geq 0, l = 0\}$

(3) $C(l, a, la, b, lb) =$
$25 + D(a, la, 0) + E(la, j) + C(l', a, la - 1, b, lb)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(4) $C(l, a, la, b, lb) =$
$24 + D(b, lb, 0) + E(lb, j) + C(l', a, la, b, lb - 1)$
$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$

(5) $D(a, la, i) = 3 \quad \{i \geq la, a \geq la, i \geq 0\}$

(6) $D(a, la, i) = 8 \quad \{i < la, a \geq la, i \geq 0\}$

(7) $D(a, la, i) = 10 + D(a, la, i + 1) \{i < la, a \geq la, i \geq 0\}$

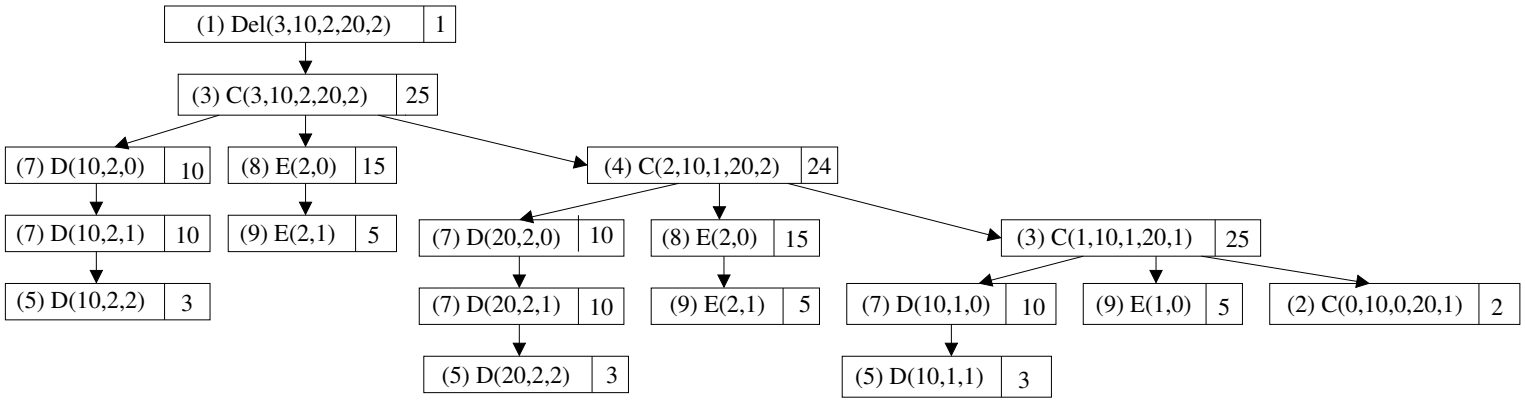
(8) $E(la, j) = 5 \quad \{j \geq la - 1, j \geq 0\}$

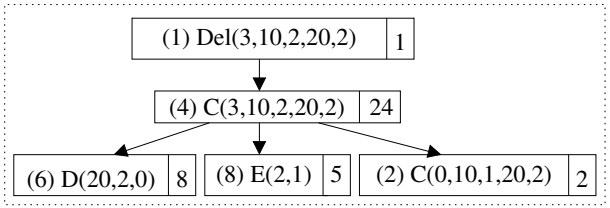
(9) $E(la, j) = 15 + E(la, j + 1) \quad \{j < la - 1, j \geq 0\}$

Evaluation Trees

Evaluation proceeds by expanding calls to cost relations

1. look for an equation whose guard is satisfied
2. look for an assignment to the variables in rhs which are compatible with the size relations

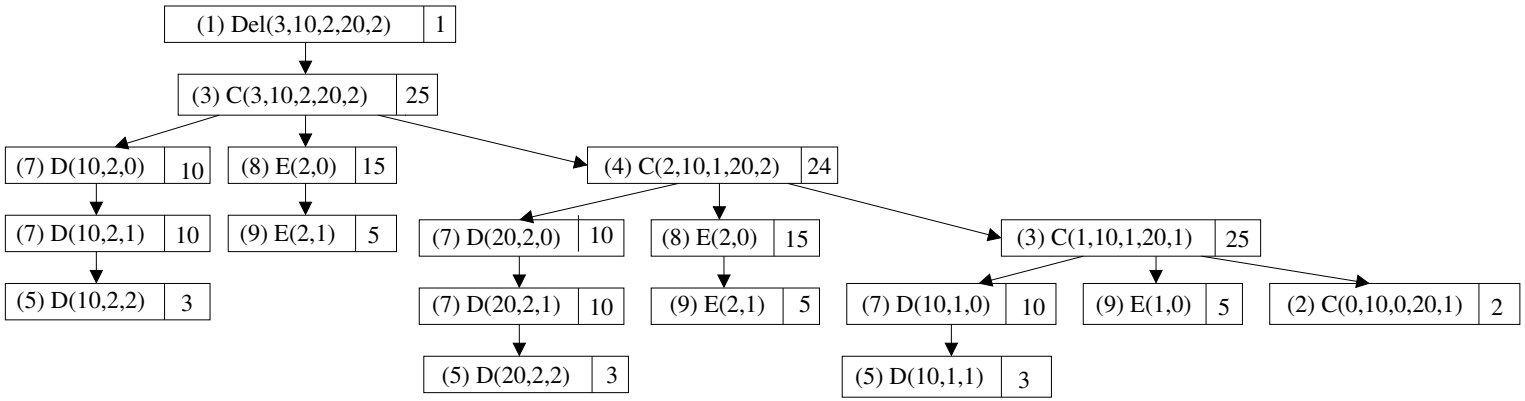


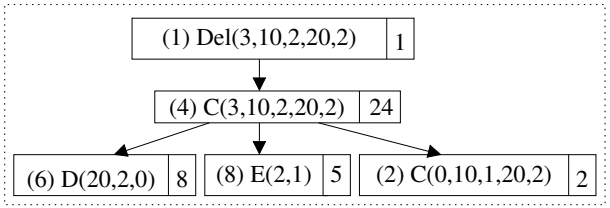


Evaluation Trees

Evaluation proceeds by expanding calls to cost relations

1. look for an equation whose guard is satisfied
2. look for an assignment to the variables in rhs which are compatible with the size relations



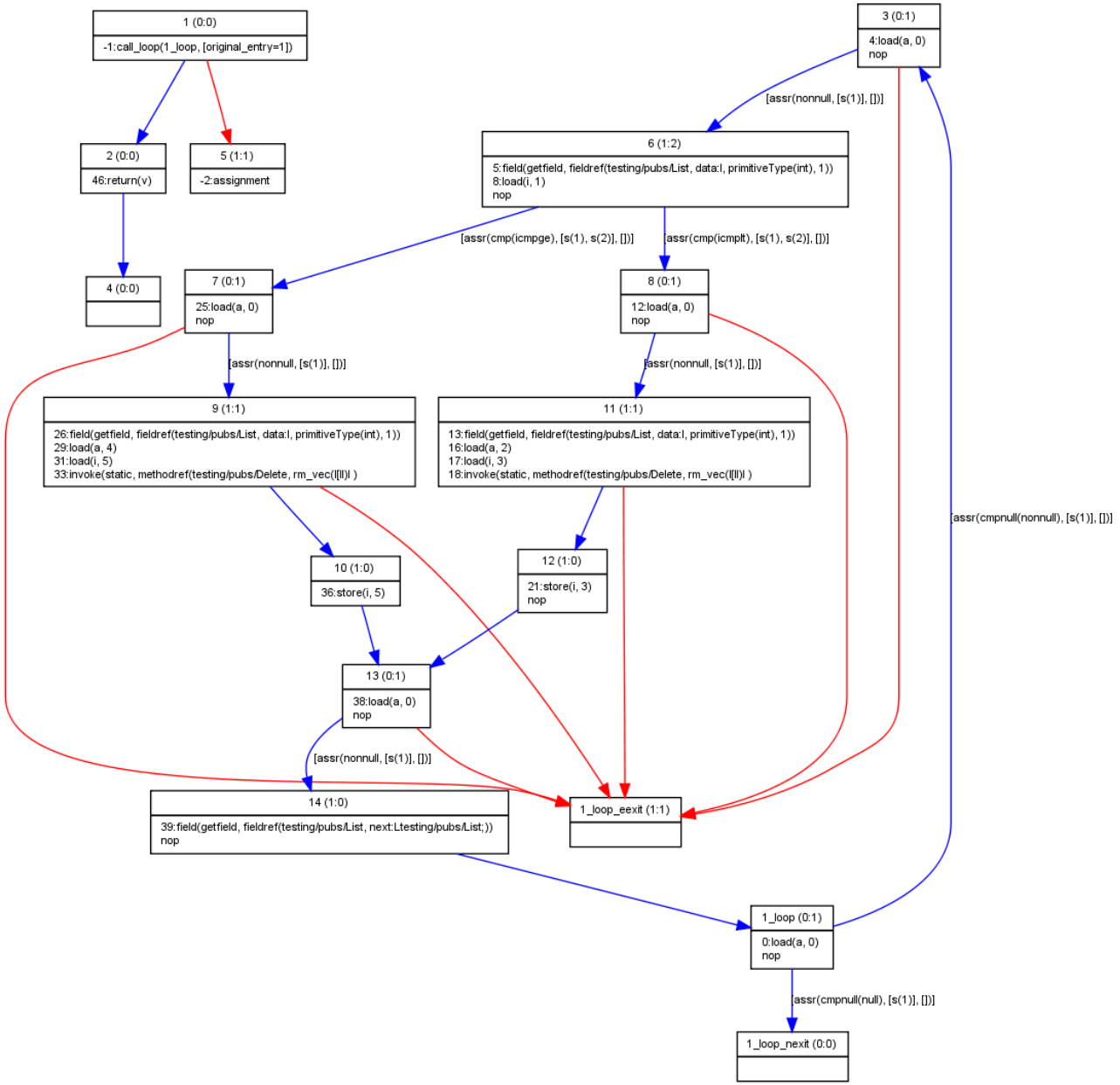


Since both conditions 1 and 2 are non-deterministic, many evaluation trees may exist for a call such as $Del(3, 10, 2, 20, 2)$.

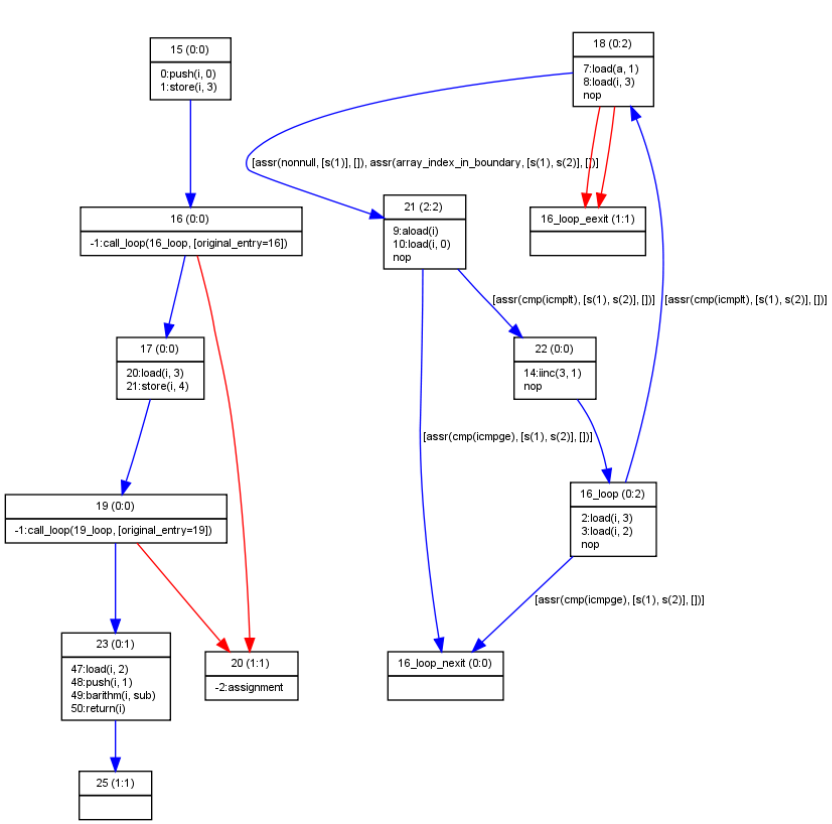
Compositionality of Upper Bounds

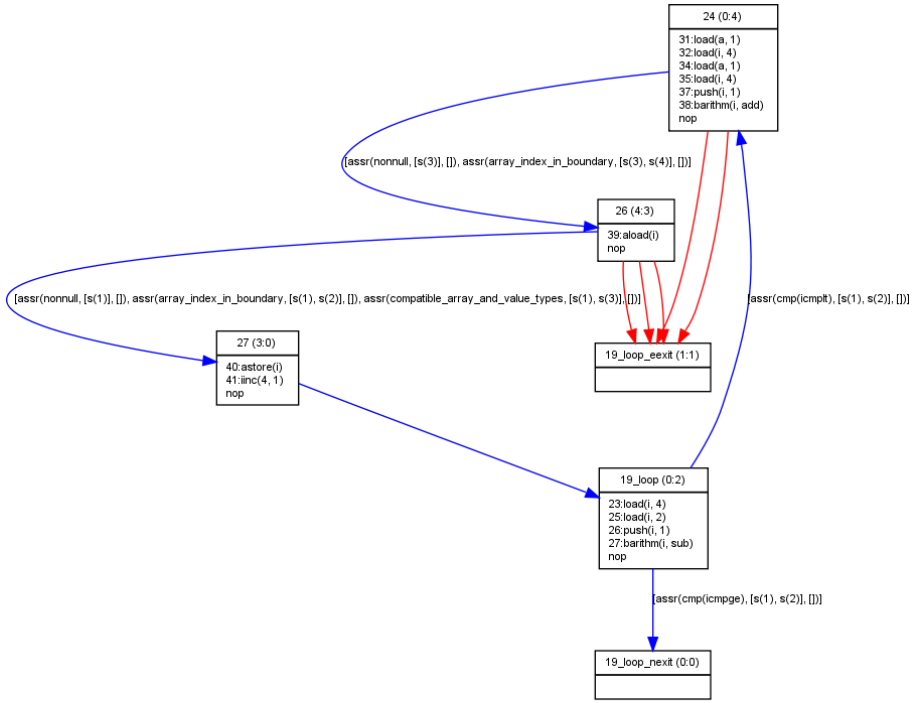
- If the equations are directly recursive, it is possible to process one cost relation at a time.
- This is possible thanks to the compositionality of upper bounds.
- Even for simple programs, obtaining direct recursion and process relations bottom-up must be done automatically.
- We do this using partial evaluation.
- This is possible as long as all SCCs have a covering point.

Control Flow Graphs for the `Del` method

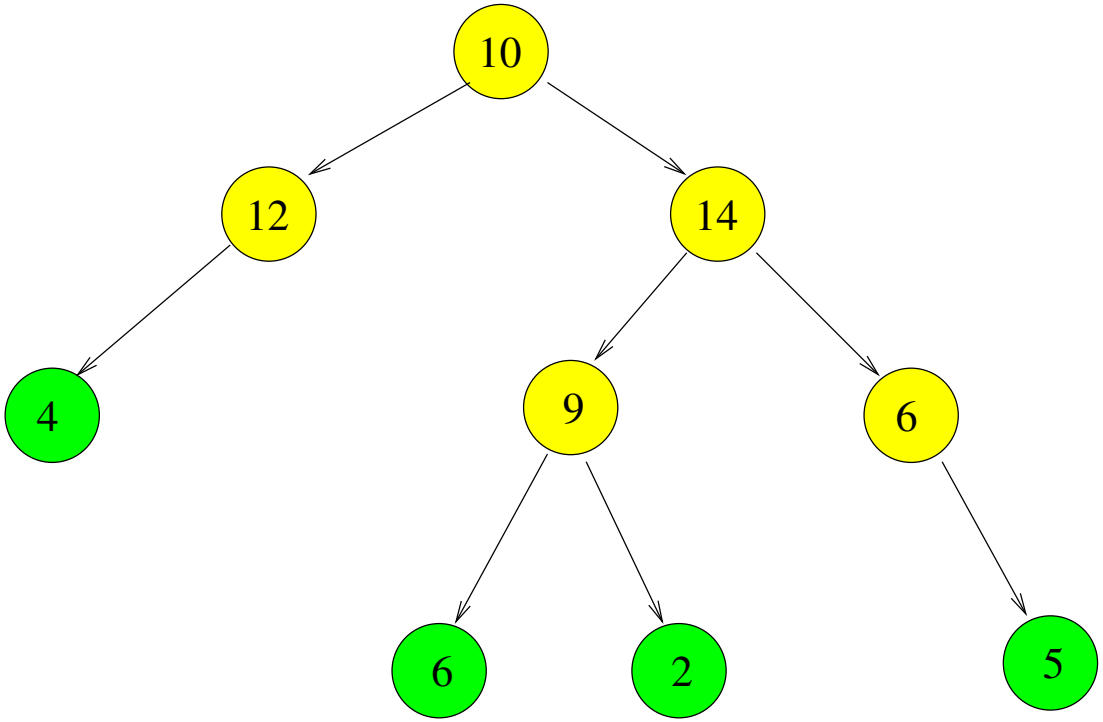


Control Flow Graphs for the `rm_vec` method





Node Count Upper Bound



$C^+(\overline{x}) = internal^+(\overline{x}) * costr^+(\overline{x}) + leaf^+(\overline{x}) * costnr^+(\overline{x})$

- $internal^+$ and $leaf^+$ are approximated using ranking functions
- $costr^+$ and $costnr^+$ are approximated using loop invariants

Self-Contained CR for Relation C

- If we have already computed closed-form upper bounds for D and E, we can obtain a Self-Contained CR for C:

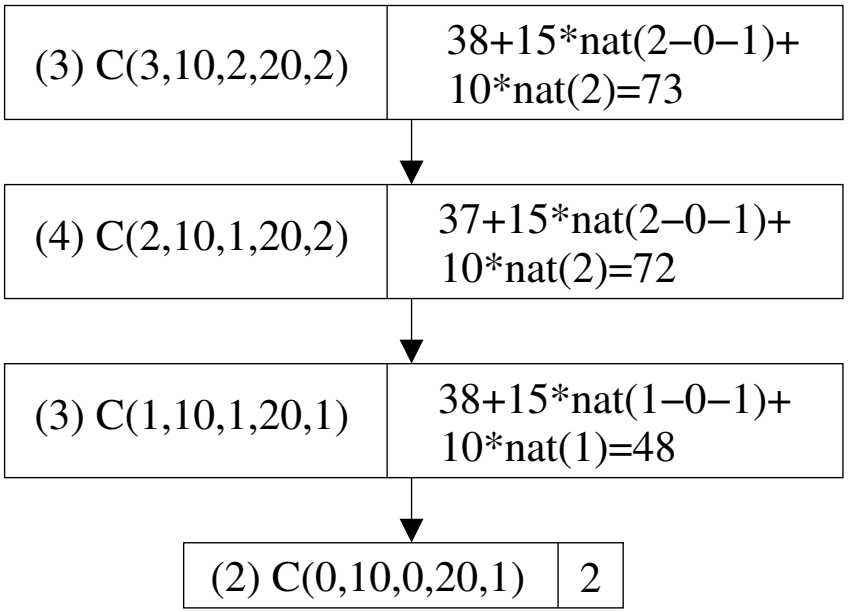
$$(2)\, C(l, a, la, b, lb) = \boxed{2}$$
$$\{a \geq la, b \geq lb, b \geq 0, a \geq 0, l = 0\}$$

$$(3)\, C(l, a, la, b, lb) = \boxed{38 + 15 * \mathrm{nat}(la - j - 1) + 10 * \mathrm{nat}(la)} + C(l', a, la - 1, b, lb)$$
$$\{a \geq 0, a \geq la, b \geq lb, j \geq 0, b \geq 0, l > l', l > 0\}$$

$$(4)\, C(l, a, la, b, lb) = \boxed{37 + 15 * \mathrm{nat}(lb - j - 1) + 10 * \mathrm{nat}(lb)} + C(l', a, la, b, lb - 1)$$
$$\{b \geq 0, b \geq lb, a \geq la, j \geq 0, a \geq 0, l > l', l > 0\}$$

Reduced Graph for Our Example

- After obtaining upper bounds for D and E we have:



- Here, it must happen that:
  - $internal^+(3, 10, 2, 20, 2) \geq 3$
  - $costr^+(3, 10, 2, 20, 2) \geq 73$
  - $leaf^+(3, 10, 2, 20, 2) \geq 1$
  - $costnr^+(3, 10, 2, 20, 2) \geq 48$

Bounding the Number of Nodes

We can use the following formulas, where:

- b stands for the branching factor, and
- in any evaluation tree for a call $C(\overline{v})$, it is limited by the maximum number of recursive calls which occur in a single equation for C.

$$leaf^{+}(\overline{x}) = b^{h^{+}(\overline{x})} \qquad internal^{+}(\overline{x}) = \begin{cases} h^{+}(\overline{x}) & b{=}1 \\ \frac{b^{h^{+}(\overline{x})}-1}{b-1} & b{\geq}2 \end{cases}$$

Bounding the Height of Trees

- Given an evaluation tree $T \in Trees(C(\overline{v}), \mathcal{S})$ for a cost relation C, consecutive nodes in any branch of T represent consecutive recursive calls which occur during the evaluation of $C(\overline{v})$.
- Therefore, bounding the height of a tree may be reduced to bounding consecutive recursive calls.
- The notion of *loop* in a cost relation, which we introduce below, is used to model consecutive calls.

Definition

Let $\mathcal{E} = \langle C(\overline{x}) = \exp + \sum_{i=1}^{k} C(\overline{y_i}), \varphi \rangle$ be an equation for a cost relation C. Then, $Loops(\mathcal{E}) = \{ \langle C(\overline{x}) \rightarrow C(\overline{y}_i), \varphi' \rangle \mid \varphi' = \bar{\exists} \overline{x} \cup \overline{y}_i . \varphi, i = 1 \cdots k \}$ is the set of *loops* induced by $\mathcal{E}$.

Similarly, $Loops(C) = \cup_{\mathcal{E} \in def(\mathcal{S}, C)} Loops(\mathcal{E})$.

Example of Loops

Example

Eqs. (3) and (4) induce the following two loops:

$$(3)\langle C(l, a, la, b, lb) \rightarrow \quad C(l', a, la', b, lb)$$
$$\varphi_1' = \{a \geq 0, a \geq la, b \geq lb, b \geq 0, l > l', l > 0, la' = la - 1\}\rangle$$

$$(4)\langle C(l, a, la, b, lb) \rightarrow \quad C(l', a, la, b, lb')$$
$$\varphi_2' = \{b \geq 0, b \geq lb, a \geq la, a \geq 0, l > l', l > 0, lb' = lb - 1\}\rangle$$

Ranking Functions

- Bounding the number of consecutive recursive calls is extensively used in the context of termination analysis.
- It is usually done by proving that there is a function f from the loop's arguments to a *well-founded* partial order which decreases in any two consecutive calls and which guarantees the absence of infinite traces, and thus termination.
- These functions are usually called *ranking functions*.
- We propose to use the ranking function to generate a h^+ function.

Definition

A function $f : \mathbb{Z}^n \mapsto \mathbb{Z}$ is a *ranking function* for a loop $\langle C(\bar{x}) \rightarrow C(\bar{y}), \varphi \rangle$ if $\varphi \models f(\bar{x}) > f(\bar{y})$ and $\varphi \models f(\bar{x}) \geq 0$.

Examples of Ranking Functions

Example

The function $f_C(l, a, la, b, lb){=}l$ is a ranking function for C. Note that φ'_1 and φ'_2 in the above loops of C contain the constraints $\{l{>}l', l{>}0\}$ which is enough to guarantee that f_C is decreasing and well-founded. The height of the evaluation tree for $C(3, 10, 2, 20, 2)$ is precisely predicted by $f_C(3, 10, 2, 20, 2){=}3$.

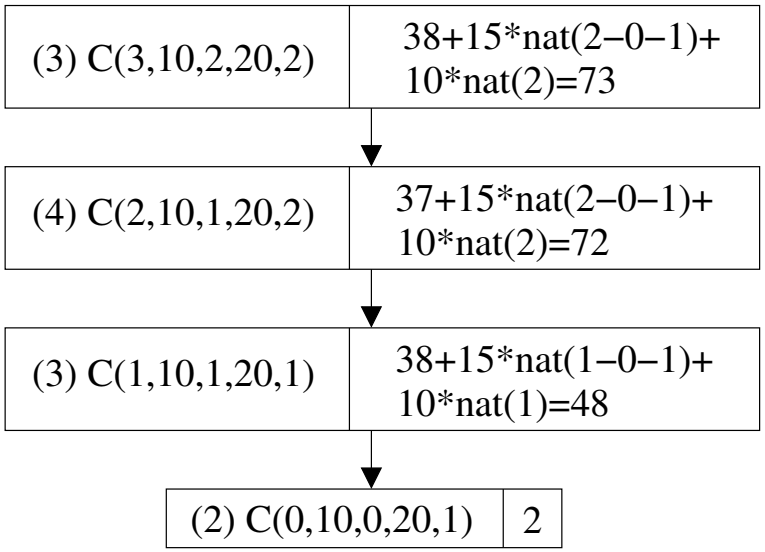
Example

Ranking functions may involve several arguments, e.g., $f_D(a, la, i){=}la{-}i$ is a ranking function for $\langle D(a, la, i) \rightarrow D(a, la, i'), \{i'{=}i{+}1, i{<}la, a{\geq}la, i{\geq}0\}\rangle$ which comes from Eq. (7).

More Accurate Tree Heights

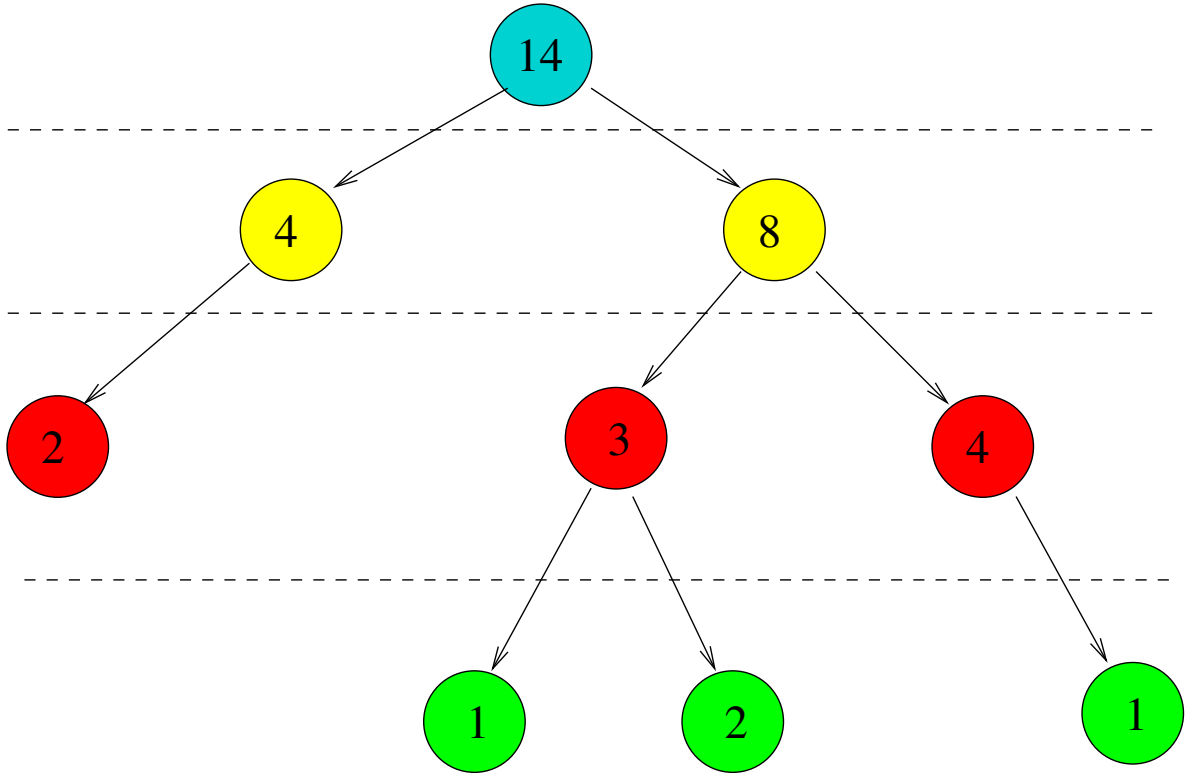
- Note that ranking functions might return a negative value when is applied to values which correspond to base cases (leaves of the tree). Therefore, we define $h^+(\overline{x})$=nat$(f_C(\overline{x}))$.
- Function nat guarantees that negative values are lifted to 0 and, therefore, they provide a correct approximation for the height of evaluation trees with a single node.
- If the difference between the value of the ranking function in each two consecutive calls is larger than a constant $\delta>1$, then $\lceil \text{nat}(\frac{f_C(\bar{x})}{\delta}) \rceil$ is a tighter upper bound.
- A more interesting case, if each loop $\langle C(\overline{x}) \rightarrow C(\overline{y}), \varphi \rangle \in \mathit{Loops}(C)$ satisfies $\varphi \models f_C(\overline{x}) \geq k * f_C(\overline{y})$ where $k>1$, then the height of the tree is bounded by $\lceil \log_k(\text{nat}(f_C(\overline{v})+1)) \rceil$.

Estimating the Cost per Node



- All expressions in the nodes are instances of the expressions which appear in the corresponding equations.
- Thus, computing $costr^{+}(\overline{x})$ and $costnr^{+}(\overline{x})$ can be done by first finding an upper bound of such expressions and then combining them through a *max* operator.
- We first compute *invariants* for the values that the expression variables can take w.r.t. the initial values, and use them to derive upper bounds for such expressions.

Divide and Conquer



$C^+(\overline{x}) = I^+(\overline{x}) * costI^+(\overline{x})$

Provided that Sum_Level$(T, k) \geq$ Sum_Level$(T, k + 1)$

Experimental Results (Accuracy)

Benchmark		$\#_{eq}$		T	Upper Bound
Polynomial*	abc	23	(3)	13	216
DivByTwo	ab	9	(3)	3	$8\log_2(\text{nat}(2x-1)+1)+14$
Factorial^M		8	(2)	3	$9\text{nat}(x)+4$
ArrayRev^M	a	9	(3)	4	$14\text{nat}(x)+12$
Concat^M	ac	14	(5)	4	$11\text{nat}(x)+11\text{nat}(y)+25$
Incr^M	ac	28	(5)	13	$19\text{nat}(x+1)+9$
ListRev^M	abc	9	(3)	29	$13\text{nat}(x)+8$
MergeList	abc	21	(4)	4	$29\text{nat}(x+y)+26$
Power		8	(2)	18	$10\text{nat}(x)+4$
Cons*	ab	22	(2)	3	$22\text{nat}(x-1)+24$
EvenDigits	abc	18	(5)	6	$\text{nat}(x)(8\log_2(\text{nat}(2x-3)+1)+24)+9\text{nat}(x)+9$
ListInter	abc	37	(9)	9	$\text{nat}(x)(10\text{nat}(y)+43)+21$
SelectOrd	ac	19	(6)	59	$\text{nat}(x-2)(17\text{nat}(x-2)+34)+9$
FactSum	a	17	(5)	27	$\text{nat}(x+1)(9\text{nat}(x)+16)+6$
Delete	abc	33	(9)	8 125	$3+\text{nat}(l)\max(38+15\text{nat}(la-1)+10\text{nat}(la),$ $37+15\text{nat}(lb-1)+10\text{nat}(lb))$
MatMult^M	ac	19	(7)	23	$\text{nat}(y)(\text{nat}(x)(27\text{nat}(x))+10)+17$
Hanoi		9	(2)	4	$20(2^{\text{nat}(x)})\text{-}17$
Fibonacci^M		8	(2)	5	$18(2^{\text{nat}(x-1)})\text{-}13$
BST*	ab	31	(4)	26	$96(2^{\text{nat}(x)})\text{-}49$

Experimental Results (scalability)

Benchmark		$\#^{c}_{eq}$		T_{pe}	T_{ub}	Rat.
Polynomial*	abc	346	(70)	174	649	2.4
DivByTwo	ab	323	(68)	166	596	2.4
Factorial^{M}		314	(66)	165	590	2.4
ArrayRev^{M}	a	305	(64)	165	579	2.4
Concat^{M}	ac	296	(62)	158	538	2.4
Incr^{M}	ac	282	(58)	155	490	2.3
ListRev^{M}	abc	254	(54)	144	415	2.2
MergeList	abc	245	(52)	138	406	2.2
Power		223	(48)	125	371	2.2
Cons*	ab	214	(46)	123	359	2.3
EvenDigits	abc	191	(44)	115	322	2.3
ListInter	abc	173	(40)	110	298	2.4
SelectOrd	ac	136	(32)	86	198	2.1
FactSum	a	117	(27)	76	173	2.1
Delete	abc	100	(23)	71	165	2.4
MatMult^{M}	ac	67	(15)	27	40	1.0
Hanoi		48	(8)	23	17	0.8
Fibonacci^{M}		39	(6)	20	13	0.8
BST*	ab	31	(4)	19	7	0.9

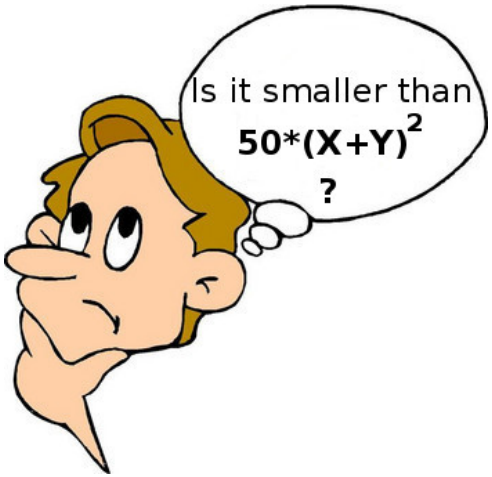
Upper-bounds checker

$$
\begin{array}{l}
3{*}\mathsf{nat}(x{+}y){*}3{+}(\mathsf{nat}(x)^2/5{+}30{*}\mathsf{nat}(x){+}\\
15{*}\mathsf{nat}(x{-}y)){+}\mathit{log}(\mathsf{nat}(y)){*}\mathsf{nat}(x{+}4){+}\\
11{*}\mathsf{nat}(x{+}y)){+}\mathit{log}(\mathsf{nat}(x)){*}\mathsf{nat}(x{+}4){+}\\
12{*}\mathsf{nat}(x{+}y){*}3{+}(\mathsf{nat}(x)^2/5{+}30{*}\mathsf{nat}(x){+}\\
3{*}\mathsf{nat}(x{-}y)){+}\mathit{log}(\mathsf{nat}(x{+}y)){*}\mathsf{nat}(x{+}4){+}\\
5{*}\mathsf{nat}(x{+}y)){+}\mathit{log}(\mathsf{nat}(x{-}y)){*}\mathsf{nat}(x{+}4)
\end{array}
$$

- Upper-bounds might be difficult to read for ordinary users.

Upper-bounds checker

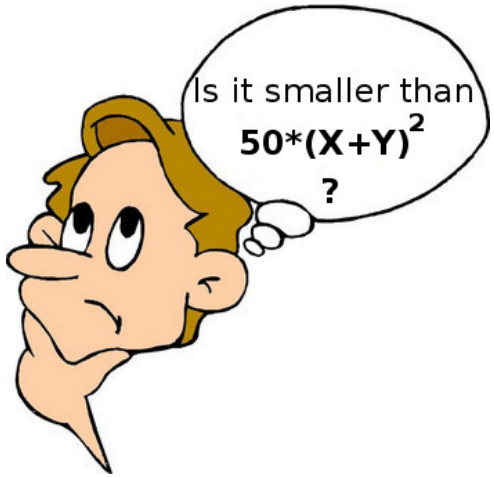
$$
\begin{aligned}
&3{*}\mathsf{nat}(x{+}y){*}3{+}(\mathsf{nat}(x)^2/5{+}30{*}\mathsf{nat}(x){+}\\
&15{*}\mathsf{nat}(x{-}y)){+}\mathit{log}(\mathsf{nat}(y)){*}\mathsf{nat}(x{+}4){+}\\
&11{*}\mathsf{nat}(x{+}y)){+}\mathit{log}(\mathsf{nat}(x)){*}\mathsf{nat}(x{+}4){+}\\
&12{*}\mathsf{nat}(x{+}y){*}3{+}(\mathsf{nat}(x)^2/5{+}30{*}\mathsf{nat}(x){+}\\
&3{*}\mathsf{nat}(x{-}y)){+}\mathit{log}(\mathsf{nat}(x{+}y)){*}\mathsf{nat}(x{+}4){+}\\
&5{*}\mathsf{nat}(x{+}y)){+}\mathit{log}(\mathsf{nat}(x{-}y)){*}\mathsf{nat}(x{+}4)
\end{aligned}
$$



- Upper-bounds might be difficult to read for ordinary users.
- In PCC we do not want to understand them, just to check if they meet the safety policy - compare to the expected cost.

Upper-bounds checker

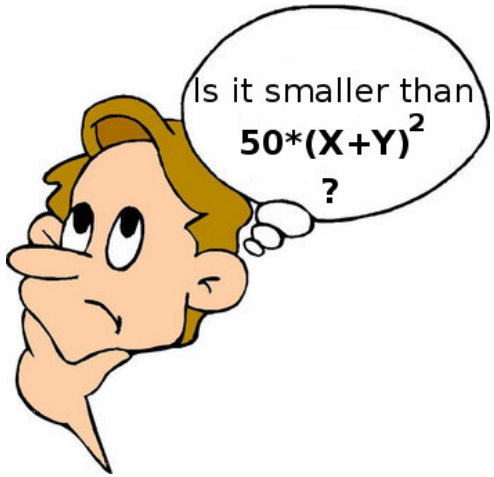
$$
\begin{array}{l}
3{*}\mathrm{nat}(x{+}y){*}3{+}(\mathrm{nat}(x)^2/5{+}30{*}\mathrm{nat}(x){+}\\
15{*}\mathrm{nat}(x{-}y)){+}\mathit{log}(\mathrm{nat}(y)){*}\mathrm{nat}(x{+}4){+}\\
11{*}\mathrm{nat}(x{+}y)){+}\mathit{log}(\mathrm{nat}(x)){*}\mathrm{nat}(x{+}4){+}\\
12{*}\mathrm{nat}(x{+}y){*}3{+}(\mathrm{nat}(x)^2/5{+}30{*}\mathrm{nat}(x){+}\\
3{*}\mathrm{nat}(x{-}y)){+}\mathit{log}(\mathrm{nat}(x{+}y)){*}\mathrm{nat}(x{+}4){+}\\
5{*}\mathrm{nat}(x{+}y)){+}\mathit{log}(\mathrm{nat}(x{-}y)){*}\mathrm{nat}(x{+}4)
\end{array}
$$



- Upper-bounds might be difficult to read for ordinary users.
- In PCC we do not want to understand them, just to check if they meet the safety policy - compare to the expected cost.
- Comparing is easy when everything is linear: we use linear constraints solver to check if $e_1 > e_2$.

Upper-bounds checker

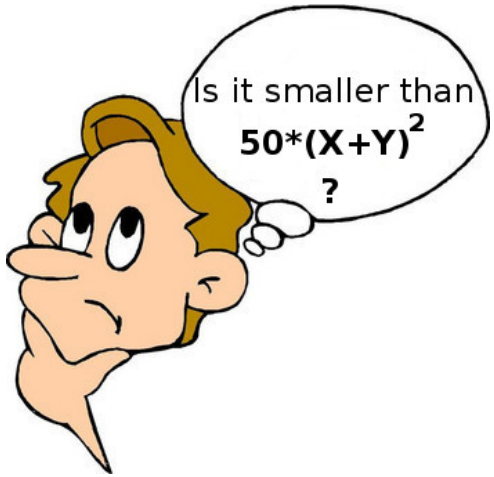
$$
\begin{array}{l}
3{*}\mathrm{nat}(x{+}y){*}3{+}(\mathrm{nat}(x)^2/5{+}30{*}\mathrm{nat}(x){+}\\
15{*}\mathrm{nat}(x{-}y)){+}\mathit{log}(\mathrm{nat}(y)){*}\mathrm{nat}(x{+}4){+}\\
11{*}\mathrm{nat}(x{+}y)){+}\mathit{log}(\mathrm{nat}(x)){*}\mathrm{nat}(x{+}4){+}\\
12{*}\mathrm{nat}(x{+}y){*}3{+}(\mathrm{nat}(x)^2/5{+}30{*}\mathrm{nat}(x){+}\\
3{*}\mathrm{nat}(x{-}y)){+}\mathit{log}(\mathrm{nat}(x{+}y)){*}\mathrm{nat}(x{+}4){+}\\
5{*}\mathrm{nat}(x{+}y)){+}\mathit{log}(\mathrm{nat}(x{-}y)){*}\mathrm{nat}(x{+}4)
\end{array}
$$



- Upper-bounds might be difficult to read for ordinary users.
- In PCC we do not want to understand them, just to check if they meet the safety policy - compare to the expected cost.
- Comparing is easy when everything is linear: we use linear constraints solver to check if $e_1 > e_2$.
- Comparing non-linear expressions is rather difficult and expensive: $2^x + 3 * (x + y)^2 > 2^x + x^2 + \mathit{log}(y)$?.

Upper-bounds checker

$$
\begin{array}{l}
3{*}\mathrm{nat}(x{+}y){*}3{+}(\mathrm{nat}(x)^2/5{+}30{*}\mathrm{nat}(x){+}\\
15{*}\mathrm{nat}(x{-}y)){+}\mathit{log}(\mathrm{nat}(y)){*}\mathrm{nat}(x{+}4){+}\\
11{*}\mathrm{nat}(x{+}y)){+}\mathit{log}(\mathrm{nat}(x)){*}\mathrm{nat}(x{+}4){+}\\
12{*}\mathrm{nat}(x{+}y){*}3{+}(\mathrm{nat}(x)^2/5{+}30{*}\mathrm{nat}(x){+}\\
3{*}\mathrm{nat}(x{-}y)){+}\mathit{log}(\mathrm{nat}(x{+}y)){*}\mathrm{nat}(x{+}4){+}\\
5{*}\mathrm{nat}(x{+}y)){+}\mathit{log}(\mathrm{nat}(x{-}y)){*}\mathrm{nat}(x{+}4)
\end{array}
$$



- Upper-bounds might be difficult to read for ordinary users.
- In PCC we do not want to understand them, just to check if they meet the safety policy - compare to the expected cost.
- Comparing is easy when everything is linear: we use linear constraints solver to check if $e_1 > e_2$.
- Comparing non-linear expressions is rather difficult and expensive: $2^x + 3 * (x + y)^2 > 2^x + x^2 + log(y)$?.
- We have developed a practical checker for non-linear expressions.

Asymptotic Cost Analysis

- Adapt **the notion of asymptotic complexity** to cover the analysis of realistic programs.
- A **novel transformation** from *non-asymptotic* cost functions into asymptotic form.
- **Integrate the above transformation** within the process of obtaining upper bounds.
- **Implement the method** withing the COSTA system.

An Example: From non-asymptotic to asymptotic

Non-asymptotic cost expression

$$5+7*\mathsf{nat}(3x+1)*\max(\{100*\mathsf{nat}(x)^2*\mathsf{nat}(y)^4, 11*3^{\mathsf{nat}(y-1)}*\mathsf{nat}(x+5)^2\})+$$
$$2*\log(\mathsf{nat}(x+2))*2^{\mathsf{nat}(y-3)}*\log(\mathsf{nat}(y+4))*\mathsf{nat}(2x-2y)$$

Asymptotic cost expression

$$3^{\mathsf{nat}(y)} * \mathsf{nat}(x)^3$$

An Example: Computing an asymptotic UB from a CR

Cost Relation:

$$\langle C(a,b) = \underline{\mathsf{nat}(a+1)^2}\ ,\ \{a{\geq}0, b{\geq}0\}\rangle$$
$$\langle C(a,b) = \underline{\mathsf{nat}(a{-}b){+}\log(\mathsf{nat}(a{-}b))}{+}C(a',b')\ ,\{a{\geq}0, b{\geq}0, a'{=}a{-}2, b'{=}b{+}1\}\rangle$$
$$\langle C(a,b) = \underline{2^{\mathsf{nat}(a+b)}{+}\mathsf{nat}(a){*}\log(\mathsf{nat}(a))}{+}C(a',b')\ ,$$
$$\{a{\geq}0, b{\geq}0, a'{=}a{+}1, b'{=}b{-}1\}\rangle$$

Asymptotic Cost Relation:

$$\begin{array}{lcll}
\langle C_A(a,b) & = & \mathsf{nat}(a)^2 & , \{a{\geq}0, b{\geq}0\}\rangle \\
\langle C_A(a,b) & = & \mathsf{nat}(a{-}b){+}C_A(a',b') & , \{a{\geq}0, b{\geq}0, a'{=}a{-}2, b'{=}b{+}1\}\rangle \\
\langle C_A(a,b) & = & 2^{\mathsf{nat}(a+b)}{+}C_A(a',b') & , \{a{\geq}0, b{\geq}0, a'{=}a{+}1, b'{=}b{-}1\}\rangle
\end{array}$$

Ranking function: $\mathsf{nat}(2a+3b{+}1)$.

Invariant: $\{0{\leq}a{\leq}a_0, 0{\leq}b{\leq}b_0, a{\geq}0, b{\geq}0\}$

Max.: $\mathsf{nat}(a)^2 \mapsto \mathsf{nat}(a_0)^2$, $\mathsf{nat}(a{-}b) \mapsto \mathsf{nat}(a_0)$, $2^{\mathsf{nat}(a+b)} \mapsto 2^{\mathsf{nat}(a_0+b_0)}$

Asymptotic UB: $2^{\mathsf{nat}(a+b)} * \mathsf{nat}(2a+3b)$.

Bench.	T_{ub}	T_{aub}	$Size_{ub}$	$Size_{aub}$	#Eq	$\frac{Size_{ub}}{\#Eq}$	$\frac{Size_{aub}}{\#Eq}$	$\frac{Size_{ub}}{Size_{aub}}$
BST	0	0	23	4	31	0.14	0.13	5.75
Fibonacci	0	0	47	9	39	1.21	0.23	5.22
Hanoi	0	0	67	14	48	1.39	0.29	4.78
MatMult	0	0	152	38	67	2.27	0.56	4.00
Delete	0	4	320	65	100	3.20	0.65	4.92
FactSum	4	4	717	95	117	6.12	0.81	7.54
SelectOrd	0	4	1447	155	136	10.63	1.14	9.33
ListInter	4	16	3804	257	173	21.98	1.48	14.80
EvenDigits	4	20	7631	400	191	39.95	2.09	19.07
Cons	12	32	15268	585	214	71.34	2.73	26.09
Power	24	40	24265	588	223	108.81	2.63	41.26
MergeList	96	60	48536	828	245	198.10	3.37	58.61
ListRev	140	76	48545	829	254	191.12	3.26	58.55
Incr	×	112	×	1126	282	×	3.99	×
Concat	×	164	×	1538	296	×	5.19	×
ArrayRev	×	232	×	2127	305	×	6.97	×
Factorial	×	284	×	2130	314	×	6.78	×
DivByTwo	×	328	×	2135	323	×	6.60	×
Polynomial	×	436	×	2971	346	×	8.58	×
MergeSort	×	440	×	3234	385	×	8.40	×

Conclusions

- We have presented COSTA, an academic system able to
  - Compute upper bounds of resource usage
  - Check them against user-provided policies
- Some features of COSTA:
  - it produces simple and reasonably accurate upper bounds
  - for a large class of programs
  - is reasonably scalable
- For more information on the upcoming tool release, please visit `http://costa.ls.fi.upm.es`
- COSTA also includes contributions from: Diego Alonso, Jesús Correas, Miguel Gómez-Zamalloa, Israel Herraiz, Diana Ramírez, and Guillermo Román
- The COSTA system is funded in part by the EU, under the FET projects ICT-231620 HATS and IST-15905 MOBIUS, by the MCINN under TIN-2008-05624 DOVES and the TIN-2005-09207 MERIT projects, and CAM under the S-0505/TIC/0407 PROMESAS project.