Test Case Generation and Cost Analysis in Java-like Languages

Elvira Albert and Miguel G-Zamalloa

Complutense University of Madrid (Spain)

1st Prometidos Summer School

Madrid, Sept 19, 2011
The COSTA Team

Staff:

E. Albert P. Arenas J. Correas S. Genaim G-Zamalloa G. Puebla Zanardini
UCM  UCM  UCM  UCM  UCM  UPM  UPM

PhD Students:

D. Alonso A. Flores A. Masud J. Rojas G. Roman
UCM  UCM  UPM  UPM  UPM
Part 1: Resource Usage Analysis (Elvira)
- Introduction to Resource Usage Analysis
- Generation of cost relations
- Closed-form upper and lower bounds
- Treatment of fields
- Conclusions and future work
- Main publications
Part 1: Resource Usage Analysis (Elvira)
- Introduction to Resource Usage Analysis
- Generation of cost relations
- Closed-form upper and lower bounds
- Treatment of fields
- Conclusions and future work
- Main publications

Part 2: Test Case Generation (Miky)
- Introduction to Test Case Generation
- CLP-based TCG of Java-like Programs
- Compositionality
- Abstraction-guided TCG
- Resource-driven TCG
- Demo of PET
Part 1: Resource Usage Analysis
static cost analysis

bound the cost of executing program $P$ on any input data $\bar{x}$ without having to actually run $P(\bar{x})$
Introduction: Resource Usage Analysis

**static cost analysis**

bound the cost of executing program $P$ on any input data $\overline{x}$ without having to actually run $P(\overline{x})$

- reasoning about execution cost is difficult and error-prone
- cost analysis, or resource analysis or complexity analysis should be automatic
Introduction: Resource Usage Analysis

**static cost analysis**

bound the cost of executing program $P$ on any input data $\overline{x}$ without having to actually run $P(\overline{x})$

- reasoning about execution cost is difficult and error-prone
- cost analysis, or resource analysis or complexity analysis should be automatic
- the resources considered:
  - number of executed instructions
  - memory usage
  - number of calls to a specific method (billable events)
  - termination (it guarantees the existence of an upper bound)
Different kinds of cost can be considered:

- worst case $\rightarrow$ upper bound
- best case $\rightarrow$ lower bound
- average case $\rightarrow$ requires probabilistic study
Different kinds of cost can be considered:
- worst case → upper bound
- best case → lower bound
- average case → requires probabilistic study

Two classes of bounds can be considered:
- non-asymptotic (or concrete, or micro-analysis)
- asymptotic (or macro-analysis)
Different kinds of cost can be considered:
- worst case → upper bound
- best case → lower bound
- average case → requires probabilistic study

Two classes of bounds can be considered:
- non-asymptotic (or concrete, or micro-analysis)
- asymptotic (or macro-analysis)

Analysis results can be
- platform-independent
- platform-dependent → WCET, energy
Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.

His system, metric was able to compute:
- interesting results, but for
- restricted class of functional programs
Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.

His system, metric was able to compute:
- interesting results, but for
- restricted class of functional programs

Also, the seminal work on abstract interpretation [Cousot & Cousot'77] mentions performance analysis as an application.
Work on automatic cost analysis dates back to 1975, with the seminal work of Wegbreit.

His system, metric was able to compute:
  - interesting results, but for
  - restricted class of functional programs

Also, the seminal work on abstract interpretation [Cousot & Cousot'77] mentions performance analysis as an application.

Since then, a number of analyses and systems have been built which extend the capabilities of cost analysis:
  - functional programs [Le Metayer’88, Rosendahl’89, Wadler’88, Sands’95, Benzinger’04, Hofmann’10]
  - logic programs [Debray and Lin’93, Navas et al’07]
  - imperative programs [Adachi et al’79, Albert et al’07, Gulwani’09]
A classical approach [Wegbreit’75] 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
A classical approach [Wegbreit’75] 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

The current situation is that

- Most work has concentrated on the first phase
- The difficulties of the second phase have been overseen
- Practical usage of cost analysis requires both!
- In COSTA we address both phases.
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
```java
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];

        while ( j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }

        a[j-1]=value;
    }
}
```
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
static void sort(int a[]) {
    for (int i=a.length-2; i > -1; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]< value ) {
            a[j-1]= a[j];
            j++;
        }
        a[j-1]= value;
    }
}

sort(a)=\(q*(a-1) + p*(a-1)^2\)
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--)
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value)
            a[j-1]=a[j];
            j++;
    a[j-1]=value;
}

\text{Worst-Case (UB)}
\begin{align*}
\text{sort}(a) &= q \ast (a-1) + p \ast (a-1)^2
\end{align*}

\text{Best-Case (LB)}
\begin{align*}
\text{sort}(a) &= q \ast (a-1)
\end{align*}
COSTA - Worst/Best Case

Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[sort(a) = B(a, i)\] \quad \{i=a-2, \ a\geq0\}

\[B(a, i) = 0\] \quad \{i<0\}

\[B(a, i) = C(a, j) + B(a, i')\] \quad \{i\geq0, \ j=i+1, \ i'=i-1\}

\[C(a, j) = 0\] \quad \{j\geq a\}

\[C(a, j) = 0\] \quad \{j<a\}

\[C(a, j) = C(a, j')\] \quad \{j<a, \ j'=j+1\}
Cost Analysis in Java-like Languages

```
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
```

\[
sort(a) = B(a, i) \quad \{ i=a-2, \ a \geq 0 \}
\]

\[
B(a, i) = 0 \quad \{ i<0 \}
\]

\[
B(a, i) = q + C(a, j) + B(a, i') \quad \{ i \geq 0, \ j=i+1, \ i'=i-1 \}
\]

\[
C(a, j) = 0 \quad \{ j \geq a \}
\]

\[
C(a, j) = 0 \quad \{ j < a \}
\]

\[
C(a, j) = p + C(a, j') \quad \{ j < a, \ j'=j+1 \}
\]
static void sort(int a[]) {
    for (int i=a.length - 2; i >= 0; i --) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j ++;
        }
        a[j-1]=value;
    }
}

sort(a) = B(a, i)   \{i=a−2, a≥0\}
B(a, i) = 0 \{i<0\}
B(a, i) = C(a, j)+B(a, i') \{i≥0, j=i+1, i'=i−1\}

C(a, j) = 0 \{j≥a\}
C(a, j) = 0 \{j<a\}
C(a, j) = C(a, j') \{j<a, j'=j+1\}
COSTA - Worst/Best Case

Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

sort(a) = B(a, i) \quad \{i=a-2, \quad a \geq 0\}
B(a, i) = 0 \quad \{i<0\}
B(a, i) = q + C(a, j) + B(a, i') \quad \{i \geq 0, \quad j=i+1, \quad i'=i-1\}

C(a, j) = 0 \quad \{j \geq a\}
C(a, j) = 0 \quad \{j < a\}
C(a, j) = p + C(a, j') \quad \{j < a, \quad j'=j+1\}
static void sort(int a[]) {
    for (int i = a.length - 2; i >= 0; i --) {
        int j = i + 1;
        int value = a[i];
        while (j < a.length && a[j] < value) {
            a[j - 1] = a[j];
            j ++;
        }
        a[j - 1] = value;
    }
}

\[
\text{sort}(a) = B(a, i) \quad \{i = a - 2, \ a \geq 0\} \\
B(a, i) = 0 \quad \{i < 0\} \\
B(a, i) = q + C(a, j) + B(a, i') \quad \{i \geq 0, \ j = i + 1, \ i' = i - 1\} \\
\]

\[
C(a, j) = 0 \quad \{j \geq a\} \\
C(a, j) = 0 \quad \{j < a\} \\
C(a, j) = p + C(a, j') \quad \{j < a, \ j' = j + 1\}
\]
Cost Analysis in Java-like Languages

```
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j ++;
        }
        a[j-1]=value;
    }
}
```

\[
\begin{align*}
\text{sort}(a) &= B(a, i) & \{i=a-2, \ a \geq 0\} \\
B(a, i) &= 0 & \{i<0\} \\
B(a, i) &= \mathbb{q} + C(a, j) + B(a, i') & \{i \geq 0, \ j=i+1, \ i'=i-1\} \\
C(a, j) &= 0 & \{j \geq a\} \\
C(a, j) &= 0 & \{j < a\} \\
C(a, j) &= \mathbb{p} + C(a, j') & \{j < a, \ j'=j+1\}
\end{align*}
\]
Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[\begin{align*}
    \text{sort}(a) &= B(a,i) \quad \{i=a-2, \ a \geq 0\} \\
    B(a,i) &= 0 \quad \{i<0\} \\
    B(a,i) &= q + C(a,j) + B(a,i') \quad \{i \geq 0, \ j=i+1, \ i'=i-1\} \\
    C(a,j) &= 0 \quad \{j \geq a\} \\
    C(a,j) &= 0 \quad \{j < a\} \\
    C(a,j) &= p + C(a,j') \quad \{j < a, \ j'=j+1\}
\end{align*}\]
Program → Static Analysis → Cost Relations → CRs Solver → Best/Worst Case

static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--)
    {  
      int j=i+1;
      int value=a[i];
      while ( j<a.length && a[j]<value )
      {  
        a[j-1]=a[j];
        j++;
      }  
      a[j-1]=value;
    }
}

sort(a) = B(a, i) \text{  } \{ i=a-2, \ a \geq 0 \} \\
B(a, i) = 0 \text{  } \{ i<0 \} \\
B(a, i) = q + C(a, j) + B(a, i') \text{  } \{ i \geq 0, \ j=i+1, \ i'=i-1 \} \\
C(a, j) = 0 \text{  } \{ j \geq a \} \\
C(a, j) = 0 \text{  } \{ j < a \} \\
C(a, j) = p + C(a, j') \text{  } \{ j < a, \ j'=j+1 \}
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

sort(a) = B(a,i) \quad \{ i=a-2, \ a \geq 0 \}

B(a,i) = 0 \quad \{ i<0 \}
B(a,i) = q + C(a,j) + B(a,i') \quad \{ i \geq 0, \ j=i+1, \ i'=i-1 \}

C(a,j) = 0 \quad \{ j \geq a \}
C(a,j) = 0 \quad \{ j < a \}
C(a,j) = p + C(a,j') \quad \{ j < a, \ j'=j+1 \}
COSTA - Worst/Best Case

Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

static void sort(int a[]) {
  for (int i=a.length-2; i>=0; i--) {
    int j=i+1;
    int value=a[i];
    while (j<a.length && a[j]<value) {
      a[j-1]=a[j];
      j++;
    }
    a[j-1]=value;
  }
}

\[
\text{sort}(a) = B(a, i) \quad \{i=a-2, \ a \geq 0\}
\]

\[
B(a, i) = \begin{cases} 
0 & \{i < 0\} \\
q + C(a, j) + B(a, i') & \{i \geq 0, \ j=i+1, \ i'=i-1\} 
\end{cases}
\]

\[
C(a, j) = \begin{cases} 
0 & \{j \geq a\} \\
0 & \{j < a\} \\
p + C(a, j') & \{j < a, \ j'=j+1\} 
\end{cases}
\]
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[
\text{sort}(a) = B(a, i) \quad \{i=a-2, \ a\geq 0\}
\]

\[
B(a, i) = 0 \quad \{i<0\}
\]

\[
B(a, i) = q + C(a, j) + B(a, i') \quad \{i\geq 0\}, \ j=i+1, \ i'=i-1\}
\]

\[
C(a, j) = 0 \quad \{j\geq a\}
\]

\[
C(a, j) = 0 \quad \{j<a\}
\]

\[
C(a, j) = p + C(a, j') \quad \{j<a, \ j'=j+1\}
\]
Cost Analysis in Java-like Languages

static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[\text{sort}(a) = B(a, i) \quad \{i=a-2, \ a \geq 0\}\]

\[B(a, i) = 0 \quad \{i<0\}\]
\[B(a, i) = (\overline{q}) + C(a, j) + B(a, i') \quad \{i \geq 0, \ j=i+1, \ i'=i-1\}\]

\[C(a, j) = 0 \quad \{j \geq a\}\]
\[C(a, j) = 0 \quad \{j < a\}\]
\[C(a, j) = (\overline{p}) + C(a, j') \quad \{j < a, \ j'=j+1\}\]
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[
\begin{align*}
\text{sort}(a) &= B(a, i) & \{ i=a-2, \ a \geq 0 \} \\
B(a, i) &= 0 & \{ i<0 \} \\
B(a, i) &= \color{red}q + C(a, j) + B(a, i') & \{ i \geq 0, \ j=i+1, \ i'=i-1 \} \\
C(a, j) &= 0 & \{ j \geq a \} \\
C(a, j) &= 0 & \{ j<a \} \\
C(a, j) &= \color{red}p + C(a, j') & \{ j<a, \ j'=j+1 \}
\end{align*}
\]

Elvira Albert
Cost Analysis in Java-like Languages
COSTA - Worst/Best Case

Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

```
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}
```

\[
sort(a) = B(a, i)
\]

\[
B(a, i) = 0\quad \{i < 0\}
\]

\[
B(a, i) = q + C(a, j) + B(a, i')\quad \{i \geq 0, j=i+1, i'=i-1\}
\]

\[
C(a, j) = 0\quad \{j \geq a\}
\]

\[
C(a, j) = 0\quad \{j < a\}
\]

\[
C(a, j) = p + C(a, j')\quad \{j < a, j'=j+1\}
\]
static void sort(int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while ( j<a.length && a[j]<value ) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[
\begin{align*}
\text{sort}(a) &= B(a, i) \quad \{i=a-2, \; a \geq 0\} \\
B(a, i) &= 0 \quad \{i<0\} \\
B(a, i) &= q + C(a, j) + B(a, i') \quad \{i \geq 0, \; j=i+1, \; i'=i-1\} \\
C(a, j) &= 0 \quad \{j \geq a\} \\
C(a, j) &= 0 \quad \{j<a\} \\
C(a, j) &= p + C(a, j') \quad \{j<a, \; j'=j+1\}
\end{align*}
\]
Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

static void sort (int a[]) {
    for (int i=a.length-2; i>=0; i--) {
        int j=i+1;
        int value=a[i];
        while (j<a.length && a[j]<value) {
            a[j-1]=a[j];
            j++;
        }
        a[j-1]=value;
    }
}

\[
\text{sort}(a) = B(a, i) \quad \begin{cases} 
    i = a-2, & a \geq 0 \\
    B(a, i) = 0 & i < 0 \\
    B(a, i) = \mathbb{Q} + C(a, j) + B(a, i') & i \geq 0, \; j = i+1, \; i' = i-1 \\
\end{cases}
\]

\[
\text{C}(a, j) = 0 \quad \begin{cases} 
    j \geq a \\
    C(a, j) = 0 & j < a \\
    C(a, j) = \mathbb{P} + C(a, j') & j < a, \; j' = j+1 \\
\end{cases}
\]
COSTA - Worst/Best Case

Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

Worst-Case (UB)

\[
\begin{align*}
    \text{sort}(a) &= q \times (a-1) + p \times (a-1)^2 \\
    B(a, i) &= q \times (i+1) + p \times (i+1) \times (a-1) \\
    C(a, j) &= p \times (a-j)
\end{align*}
\]

\[
\begin{align*}
    \text{sort}(a) &= B(a, i) & \{i=a-2, \ a\geq 0\} \\
    B(a, i) &= 0 & \{i<0\} \\
    B(a, i) &= q + C(a, j) + B(a, i') & \{i\geq 0, \ j=i+1, \ i'=i-1\} \\
    C(a, j) &= 0 & \{j\geq a\} \\
    C(a, j) &= 0 & \{j<a\} \\
    C(a, j) &= p + C(a, j') & \{j<a, \ j'=j+1\}
\end{align*}
\]
Program

Static Analysis

Cost Relations

CRs Solver

Best/Worst Case

Worst-Case (UB)

sort(a) = \( q \ast (a-1) \)

\( B(a, i) = q \ast (i+1) \)

\( C(a, j) = 0 \)

Best-Case (LB)

sort(a) = \( q \ast (a-1) \)

\( B(a, i) = q \ast (i+1) \)

\( C(a, j) = 0 \)

Worst-Case (UB)

sort(a) = \( B(a, i) \)

\( B(a, i) = 0 \)

\( B(a, i) = q + C(a, j) + B(a, i') \) \( \{i \geq 0, j=i+1, i'=i-1\} \)

COSTA - Worst/Best Case

Elvira Albert

Cost Analysis in Java-like Languages
sort(a) = B(a, i)

B(a, i) = 0

B(a, i) = \( q + C(a, j) + B(a, i') \)

C(a, j) = 0

C(a, j) = 0

C(a, j) = \( p + C(a, j') \)

\{ i = a - 2, \ a \geq 0 \}

\{ i < 0 \}

\{ i \geq 0, \ j = i + 1, \ i' = i - 1 \}

\{ j \geq a \}

\{ j < a \}

\{ j < a, \ j' = j + 1 \}
\[ \text{sort}(a) = B(a, i) \]
\[ B(a, i) = 0 \quad \{ i = a - 2, \ a \geq 0 \} \]
\[ B(a, i) = q + C(a, j) + B(a, i') \quad \{ i < 0 \} \]
\[ C(a, j) = 0 \quad \{ j \geq a \} \]
\[ C(a, j) = 0 \quad \{ j < a \} \]
\[ C(a, j) = p + C(a, j') \quad \{ j < a, \ j' = j + 1 \} \]

- Why not using directly Computer Algebra Systems?
Solving CRs - Computer Algebra Systems

\[
sort(a) = B(a, i) \quad \{i = a-2, \ a \geq 0\}
\]
\[
B(a, i) = 0 \quad \{i < 0\}
\]
\[
B(a, i) = q + C(a, j) + B(a, i') \quad \{i \geq 0, \ j = i + 1, \ i' = i - 1\}
\]
\[
C(a, j) = 0 \quad \{j \geq a\}
\]
\[
C(a, j) = 0 \quad \{j < a\}
\]
\[
C(a, j) = p + C(a, j') \quad \{j < a, \ j' = j + 1\}
\]

CAS can obtain an exact closed-form solution for:

\[
P(0) = 0
\]
\[
P(n) = E + P(n - 1) + \cdots + P(n - 1)
\]
deterministic, 1 base-case, 1 recursive case, 1 argument
Solving CRs - Computer Algebra Systems

\[ sort(a) = B(a, i) \]
\[ B(a, i) = 0 \]
\[ B(a, i) = q + C(a, j) + B(a, i') \]
\[ C(a, j) = 0 \]
\[ C(a, j) = 0 \]
\[ C(a, j) = p + C(a, j') \]
\[ \{ i = a - 2, \ a \geq 0 \} \]
\[ \{ i < 0 \} \]
\[ \{ i \geq 0, \ j = i + 1, \ i' = i - 1 \} \]
\[ \{ j \geq a \} \]
\[ \{ j < a \} \]
\[ \{ j < a, \ j' = j + 1 \} \]

- Why not using directly Computer Algebra Systems?
- CRs are not deterministic
Solving CRs - Computer Algebra Systems

\[
\begin{align*}
\text{sort}(a) &= B(a, i) \\
B(a, i) &= 0 \\
B(a, i) &= q + C(a, j) + B(a, i') \\
C(a, j) &= 0 \\
C(a, j) &= 0 \\
C(a, j) &= p + C(a, j')
\end{align*}
\]

\[
\begin{align*}
\{i &= a - 2, \ a \geq 0\} \\
\{i < 0\} \\
\{i \geq 0, \ j = i + 1, \ i' = i - 1\} \\
\{j \geq a\} \\
\{j < a\} \\
\{j < a, \ j' = j + 1\}
\end{align*}
\]

- Why not using directly Computer Algebra Systems?
- CRs are not deterministic
- CRs have multiple arguments
- CRs have multiple (not mutually exclusive) equations
- Thus, CRs often do not have an exact solution

Several possible runs for \(C(10, 1)\)

\[
\begin{align*}
C(10, 1) &\rightarrow C(10, 2) \\
C(10, 1) &\rightarrow C(10, 2) \rightarrow C(10, 3) \\
C(10, 1) &\rightarrow C(10, 2) \rightarrow C(10, 3) \rightarrow C(10, 4)
\end{align*}
\]
Solving CRs - Computer Algebra Systems

\[
\text{sort}(a) = B(a, i) \\
B(a, i) = 0 \\
B(a, i) = \begin{cases} 
 q + C(a, j) + B(a, i') & \{i=a-2, \ a \geq 0\} \\
 0 & \{i<0\} \\
 0 & \{i \geq 0, \ j=i+1, \ i'=i-1\} \\
C(a, j) = 0 & \{j \geq a\} \\
C(a, j) = 0 & \{j < a\} \\
C(a, j) = p + C(a, j') & \{j < a, \ j'=j+1\}
\end{cases}
\]

- Why not using **directly** Computer Algebra Systems?
- CRs are not deterministic
- CRs have multiple arguments
Solving CRs - Computer Algebra Systems

\[ \text{sort}(a) = B(a, i) \]

\[ B(a, i) = 0 \]

\[ B(a, i) = q + C(a, j) + B(a, i') \]

\[ C(a, j) = 0 \]

\[ C(a, j) = 0 \]

\[ C(a, j) = p + C(a, j') \]

- Why not using directly Computer Algebra Systems?
- CRs are not deterministic
- CRs have multiple arguments
- CRs have multiple (not mutually exclusive) equations
Solving CRs - Computer Algebra Systems

\[
\begin{align*}
sort(a) &= B(a, i) \\
B(a, i) &= 0 \\
B(a, i) &= q + C(a, j) + B(a, i') \\
C(a, j) &= 0 \\
C(a, j) &= 0 \\
C(a, j) &= p + C(a, j')
\end{align*}
\]

\{i = a - 2, \ a \geq 0\} \\
\{i < 0\} \\
\{i \geq 0, \ j = i + 1, \ i' = i - 1\} \\
\{j \geq a\} \\
\{j < a\} \\
\{j < a, \ j' = j + 1\}

- Why not using directly Computer Algebra Systems?
- CRs are not deterministic
- CRs have multiple arguments
- CRs have multiple (not mutually exclusive) equations
- Thus, CRs often do not have an exact solution
Solving CRs - Computer Algebra Systems

\[
\begin{align*}
\text{sort}(a) &= B(a, i) \\
B(a, i) &= 0 \\
B(a, i) &= \begin{cases} 
q + C(a, j) + B(a, i') & \text{if } i \geq 0, j = i + 1, i' = i - 1 \\
0 & \text{otherwise}
\end{cases} \\
C(a, j) &= 0 \\
C(a, j) &= \begin{cases} 
p + C(a, j') & \text{if } j < a, j' = j + 1 \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
i & = a - 2, a \geq 0 \\
i & < 0 \\
i & \geq 0, j = i + 1, i' = i - 1 \\
j & \geq a \\
j & < a \\
j & < a, j' = j + 1
\end{align*}
\]

CAS can obtain an exact closed-form solution for:

\[
\begin{align*}
P(0) &= 0 \\
P(n) &= E + P(n - 1) + \cdots + P(n - 1)
\end{align*}
\]

deterministic, 1 base-case, 1 recursive case, 1 argument
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = B(a, i) \]
\[ B(a, i) = 0 \]
\[ B(a, i) = \begin{cases} q + C(a, j) + B(a, i') & \{ i = a - 2, \ a \geq 0 \} \\ 0 & \{ i < 0 \} \end{cases} \]
\[ C(a, j) = 0 \]
\[ C(a, j) = \begin{cases} 0 & \{ j \geq a \} \\ p + C(a, j') & \{ j < a \} \end{cases} \]
\[ C(a, j) = \begin{cases} 0 & \{ j < a, \ j' = j + 1 \} \end{cases} \]
sort(a) = B(a, i)

\[ B(a, i) = 0 \]
\[ B(a, i) = q + C(a, j) + B(a, i') \]

\[ C(a, j) = 0 \]
\[ C(a, j) = 0 \]
\[ C(a, j) = p + C(a, j') \]

\{ i = a - 2, \ a \geq 0 \}
\{ i < 0 \}
\{ i \geq 0, \ j = i + 1, \ i' = i - 1 \}
\{ j \geq a \}
\{ j < a \}
\{ j < a, \ j' = j + 1 \}

- An evaluation for \( C(a_0, j_0) \) looks like:
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = B(a, i) \quad \{ i = a-2, \ a \geq 0 \} \]
\[ B(a, i) = 0 \quad \{ i < 0 \} \]
\[ B(a, i) = q + C(a, j) + B(a, i') \quad \{ i \geq 0, \ j = i+1, \ i' = i-1 \} \]

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j \geq a )</td>
<td>( C(a, j) = 0 )</td>
</tr>
<tr>
<td>( j &lt; a )</td>
<td>( C(a, j) = 0 )</td>
</tr>
<tr>
<td>( j &lt; a, \ j' = j+1 )</td>
<td>( C(a, j) = p + C(a, j') )</td>
</tr>
</tbody>
</table>

- An evaluation for \( C(a_0, j_0) \) looks like:

```
\[ p \rightarrow p \rightarrow p \rightarrow p \rightarrow p \]
```

- How many \( p \) has this chain?
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = B(a, i) \]
\[ B(a, i) = 0 \]
\[ B(a, i) = q + C(a, j) + B(a, i') \]
\[ C(a, j) = 0 \]
\[ C(a, j) = 0 \]
\[ C(a, j) = p + C(a, j') \]

\[ \{ i = a - 2, \ a \geq 0 \} \]
\[ \{ i < 0 \} \]
\[ \{ i \geq 0, \ j = i + 1, \ i' = i - 1 \} \]
\[ \{ j \geq a \} \]
\[ \{ j < a \} \]
\[ \{ j < a, \ j' = j + 1 \} \]

- An evaluation for \( C(a_0, j_0) \) looks like:

- How many has this chain?
• An evaluation for $C(a_0, j_0)$ looks like:

• How many $\textcircled{P}$ has this chain?
sort(a) = B(a, i) \{i=a-2, a \geq 0\}
B(a, i) = 0 \{i<0\}
B(a, i) = q + C(a, j) + B(a, i') \{i \geq 0, j = i+1, i' = i-1\}

C(a, j) = 0 \{j \geq a\}
C(a, j) = 0 \{j < a\}
C(a, j) = p + C(a, j') \{j < a, j' = j+1\}

- An evaluation for $C(a_0, j_0)$ looks like:

- How many $p$ has this chain?

$$\hat{f}(a_0, j_0) = \text{nat}(a_0 - j_0)$$
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = B(a, i) \]
\[ B(a, i) = 0 \]
\[ B(a, i) = q + \]
\[ C(a, j) = 0 \]
\[ C(a, j) = 0 \]
\[ C(a, j) = p + C(a, j') \]

- An evaluation for \( C(a_0, j_0) \) looks like:

- How many \( \hat{f}(a_0, j_0) = \text{nat}(a_0 - j_0) \) has this chain?
sort(a) = \begin{cases} B(a, i) & \text{if } i = a - 2, a \geq 0 \\ 0 & \text{otherwise} \end{cases}
\begin{align*}
B(a, i) &= 0 \\
B(a, i) &= q + \begin{cases} 0 & \text{if } i < 0 \\ C(a, j) & \text{if } j = i + 1, i' = i - 1 \end{cases} \quad (a \geq 0)
\end{align*}
C(a, j) = \begin{cases} 0 & \text{if } j \geq a \\ 0 & \text{if } j < a \end{cases}
\begin{align*}
C(a, j) &= p + C(a, j') + C(a, j') \\
\end{align*}
\begin{equation}
C(a_0, j_0) = p \quad 2^{\text{nat}(a_0 - j_0)}
\end{equation}

- An evaluation for $C(a_0, j_0)$ looks like:

- How many $p$ has this chain?

\[ \hat{f}(a_0, j_0) = \text{nat}(a_0 - j_0) \]
sort(a) = B(a, i) \quad \{i = a-2, \ a \geq 0\}

B(a, i) = 0 \quad \{i < 0\}

B(a, i) = q + C(a, j) + B(a, i') \quad \{i \geq 0, \ j = i + 1, \ i' = i - 1\}
sort(a) = B(a, i)

\[ B(a, i) = \begin{cases} 0 & \text{if } i < 0 \\ q + C(a, j) + B(a, i') & \text{if } i \geq 0, j = i + 1, i' = i - 1 \end{cases} \]
\[ \text{sort}(a) = \begin{cases} \text{B}(a, i) & \{ i = a - 2, \ a \geq 0 \} \\ \text{B}(a, i) = 0 & \{ i < 0 \} \\ \text{B}(a, i) = q + p \times \text{nat}(a-j) + \text{B}(a, i') & \{ i \geq 0, \ j = i+1, \ i' = i-1 \} \end{cases} \]
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = B(a, i) \]
\[ B(a, i) = 0 \quad \{i < 0\} \]
\[ B(a, i) = q + p \cdot \text{nat}(a - j) + B(a, i') \quad \{i \geq 0, \ j = i + 1, \ i' = i - 1\} \]

- An evaluation for \( B(a_0, i_0) \) looks like:
sort(a) = B(a, i)

\[
B(a, i) = \begin{cases} 
0 & \text{if } i < 0 \\
q + p*\text{nat}(a-j) + B(a, i') & \text{if } i \geq 0, j = i+1, i' = i-1 
\end{cases}
\]

- An evaluation for \( B(a_0, i_0) \) looks like:

- There are at most \( \text{nat}(i_0 + 1) \) circles (\textit{ranking function})
sort(a) = B(a, i)

\[ B(a, i) = 0 \quad \{i < 0\} \]
\[ B(a, i) = q + p \times \text{nat}(a-j) + B(a, i') \quad \{i \geq 0, \ j = i+1, \ i' = i-1\} \]

- An evaluation for \( B(a_0, i_0) \) looks like:

- There are at most \( \text{nat}(i_0 + 1) \) circles (\textit{ranking function})
sort(a) = B(a, i)

\[ B(a, i) = \begin{cases} 0 & \{i < 0\} \\ q + p \cdot \text{nat}(a-j) + B(a, i') & \{i \geq 0, \ j = i+1, \ i' = i-1\} \end{cases} \]

- An evaluation for \( B(a_0, i_0) \) looks like:

- There are at most \( \text{nat}(i_0 + 1) \) circles (\textit{ranking function})

- Worst-case is \* \( \text{nat}(i_0 + 1) \)
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = B(a, i) \]

\[ B(a, i) = 0 \quad \{i < 0\} \]
\[ B(a, i) = q + p \cdot \text{nat}(a - j) + B(a, i') \quad \{i \geq 0, \; j = i + 1, \; i' = i - 1\} \]

Inferring the box (or maximizing expression)

What is the maximum value that \( q + p \cdot \text{nat}(a - j) \) can take in terms of the initial values \( \langle a_0, i_0 \rangle \)?

Elvira Albert

Cost Analysis in Java-like Languages
sort(a) = B(a, i) \{i = a - 2, a \geq 0\}

B(a, i) = 0 \{i < 0\}

B(a, i) = \textcircled{q} + \textcircled{p} \times \text{nat}(a - j) + B(a, i') \{i \geq 0, j = i + 1, i' = i - 1\}

Inferring the box (or maximizing expression)

- What is the maximum value that \(\textcircled{q} + \textcircled{p} \times \text{nat}(a - j)\) can take in terms of the initial values \(\langle a_0, i_0 \rangle\)?
  - It is \(\textcircled{q} + \textcircled{p} \times \text{nat}(a_0 - 1)\)
\[ \text{sort}(a) = B(a, i) \]
\[ B(a, i) = 0 \quad \{ i < 0 \} \]
\[ B(a, i) = q + p \cdot \text{nat}(a-j) + B(a, i') \quad \{ i \geq 0, \ j = i+1, \ i' = i-1 \} \]

Inferring the box (or maximizing expression)

- What is the maximum value that \( q + p \cdot \text{nat}(a-j) \) can take in terms of the initial values \( \langle a_0, i_0 \rangle \)?
  - It is \( q + p \cdot \text{nat}(a_0 - 1) \)
  - Infer an invariant \( \langle B(a_0, i_0) \sim B(a, i), \psi \rangle \)
sort(a) = B(a, i)

\[ B(a, i) = \begin{cases} 0 & \text{if } i < 0 \\ \text{Nat}(a-j) + B(a, i') & \text{if } i \geq 0, \ j = i+1, \ i' = i-1 \end{cases} \]

Inferring the box (or maximizing expression)

- What is the maximum value that \( q + p \cdot \text{Nat}(a-j) \) can take in terms of the initial values \( \langle a_0, i_0 \rangle \)?
  - It is \( q + p \cdot \text{Nat}(a_0 - 1) \)
- Infer an invariant \( \langle B(a_0, i_0) \sim B(a, i), \Psi \rangle \)
- Use (parametric) integer programming to maximize \( a-j \) w.r.t \( \Psi \land \varphi \) and the parameters \( \langle a_0, i_0 \rangle \)
sort(a) = B(a, i) 

\[ \begin{align*}
B(a, i) &= 0 \\
B(a, i) &= \text{nat}(a-j) + B(a, i') \\
\end{align*} \]

- An evaluation for \( B(a_0, i_0) \) looks like:

- There are at most \( \text{nat}(i_0 + 1) \) circles (\textit{ranking function})

\[ B(a_0, i_0) = [q + p\times\text{nat}(a_0-1)]\times\text{nat}(i_0+1) \]
Solving CRs (UBs) - PUBS

\[
\text{sort}(a) = B(a, i)
\]

\[
B(a, i) = 0 \quad \{i < 0\}
\]

\[
B(a, i) = q + p \cdot \text{nat}(a-j) + B(a, i') \quad \{i \geq 0, j = i+1, i' = i-1\}
\]

- An evaluation for \(B(a_0, i_0)\) looks like:

- There are at most \(\text{nat}(i_0 + 1)\) circles (\textit{ranking function})

\[
B(a_0, i_0) = [q + p \cdot \text{nat}(a_0-1)] \cdot \text{nat}(i_0+1)
\]
Solving CRs (UBs) - PUBS

\[
\text{sort}(a) = [q + p \ast \text{nat}(a-1)] \ast \text{nat}(i+1) \quad \{ i = a-2, \ a \geq 0 \}
\]

\[
B(a, i) = 0 \quad \{ i < 0 \}
\]

\[
B(a, i) = q + p \ast \text{nat}(a-j) + B(a, i') \quad \{ i \geq 0, \ j = i+1, \ i' = i-1 \}
\]

- An evaluation for \( B(a_0, i_0) \) looks like:

![Diagram of evaluation process]

- There are at most \( \text{nat}(i_0 + 1) \) circles (\textit{ranking function})

\[
B(a_0, i_0) = [q + p \ast \text{nat}(a_0-1)] \ast \text{nat}(i_0+1)
\]
\[ \text{sort}(a) = \left[ q + p \cdot \text{nat}(a-1) \right] \cdot \text{nat}(i+1) \{ i = a-2, \ a \geq 0 \} \]

\[ B(a, i) \begin{cases} 0 & \{ i < 0 \} \\ q + p \cdot \text{nat}(a-j) + B(a, i') & \{ i \geq 0, \ j = i+1, \ i' = i-1 \} \end{cases} \]

- An evaluation for \( B(a_0, i_0) \) looks like:

- There are at most \( \text{nat}(i_0 + 1) \) circles (ranking function)

\[ B(a_0, i_0) = \left[ q + p \cdot \text{nat}(a_0-1) \right] \cdot \text{nat}(i_0+1) \]
Solving CRs (UBs) - PUBS

\[ \text{sort}(a) = [q + p * \text{nat}(a-1)] \times \text{nat}(i+1) \quad \{i = a-2, \ a \geq 0\} \]

\[ B(a, i) = 0 \quad \{i < 0\} \]

\[ B(a, i) = q + p * \text{nat}(a-j) + B(a, i') \quad \{i \geq 0, \ j = i+1, \ i' = i-1\} \]

- An evaluation for \( B(a_0, i_0) \) looks like:

- There are at most \( \text{nat}(i_0 + 1) \) circles (\textit{ranking function})

\[ B(a_0, i_0) = [q + p * \text{nat}(a_0-1)] \times \text{nat}(i_0+1) \]
Theorem (soundness)

- Let $P(\bar{x})$ be a method,
- $M$ a cost model,
- $UB(\bar{x})$ the upper bound computed from $P$.

For any valid input $\bar{v}$, if there exists a trace $t$ from $P(\bar{v})$, then we ensure $UB(\bar{v}) \geq M(t)$
Reasoning about data stored in the *heap* is rather difficult:

```java
while (i < n) {
    i++; o.m();
}
```

```
while (i < f.n) {
    i++; o.m();
}
```

Static analysis of object fields (numeric or references)

- Field-sensitive: approximate them
- Precise but inefficient
- Field-insensitive: ignore them
- Efficient but imprecise

**Challenge:**

Develop techniques that have good balance between:

- Accuracy of analysis,
- Computational cost.
Reasoning about data stored in the heap is rather difficult:

```java
while (i < n) {
    i++;
    o.m();
}
```

$n-i$ is a ranking function.

Static analysis of object fields (numeric or references) is classified:
- Field-sensitive - approximate but precise
- Field-insensitive - ignore but efficient

Numeric fields and reference fields are used all the time in real programs.

Challenge: develop techniques that have good balance between:
- Accuracy of analysis,
- Computational cost.
Reasoning about data stored in the heap is rather difficult:

\[
\text{while } (i < n) \{ i++; o.m(); \} \quad n-i \text{ is a ranking function}
\]

\[
\text{while } (i < f.n) \{ i++; o.m(); \} \quad f.n-i \text{ ranking function?}
\]
Field-Sensitive Analysis

- Reasoning about data stored in the heap is rather difficult:
  
  ```
  while (i < n) { i++; o.m(); }  
  ```
  
  $n - i$ is a ranking function.

  ```
  while (i < f.n) { i++; o.m(); }  
  ```
  
  $f.n - i$ ranking function?

- Static analysis of object fields (numeric or references) classified:
  - **field-sensitive** - approximate them 
    precise but inefficient
  - **field-insensitive** - ignore them 
    efficient but imprecise
Reasoning about data stored in the heap is rather difficult:

```java
while (i < n) { i++; o.m(); }
```

$n-i$ is a ranking function

```java
while (i < f.n) { i++; o.m(); }
```

$f.n-i$ ranking function?

Static analysis of object fields (numeric or references) classified:

- **field-sensitive** - approximate them
  precise but inefficient
- **field-insensitive** - ignore them
  efficient but imprecise

**Numeric fields** and **reference fields** are used all the time in real programs
Reasoning about data stored in the heap is rather difficult:

```java
while (i < n) { i++; o.m(); }  // n-i is a ranking function
while (i < f.n) { i++; o.m(); }  // f.n-i ranking function?
```

Static analysis of object fields (numeric or references) classified:
- **field-sensitive** - approximate them precise but inefficient
- **field-insensitive** - ignore them efficient but imprecise

*Numeric fields* and *reference fields* are used all the time in real programs

Challenge:

Develop techniques that have good balance between:
- accuracy of analysis,
- computational cost.
Field-sensitive analysis by field-insensitive analysis

- analyze the behavior of scopes or program fragments
- model only those fields which behave as local variables
- $\Rightarrow$ AI-based static analysis to prove constancy of references
- write accesses are done through the same memory location
- $\Rightarrow$ check after analysis
- transform the code to replace local fields by variables
- infer information on the fields through associated ghost variables
1. analyze the behavior of scopes or program fragments
1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as local variables
Field-sensitive analysis by field-insensitive analysis

1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as local variables
   - the memory location does not change
Field-sensitive analysis by field-insensitive analysis

1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as local variables
   - the memory location does not change \(\Rightarrow\) AI-based static analysis to prove constancy of references
Field-sensitive analysis by field-insensitive analysis

1. Analyze the behavior of scopes or program fragments
2. Model only those fields which behave as local variables
   - The memory location does not change ⇒ AI-based static analysis to prove constancy of references
   - Write accesses are done through the same memory location
Field-sensitive analysis by field-insensitive analysis

1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as **local variables**
   - the memory location does not change ⇒ AI-based static analysis to prove constancy of references
   - write accesses are done through the same memory location ⇒ check after analysis
Field-sensitive analysis by field-insensitive analysis

1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as **local variables**
   - the memory location does not change \(\Rightarrow\) AI-based static analysis to prove constancy of references
   - write accesses are done through the same memory location \(\Rightarrow\) check after analysis

```java
while ( x != null ) {
    for(; x.c<n; x.c++)
        value[x.c]++;
    x=x.next;
}
```
Field-sensitive analysis by field-insensitive analysis

1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as local variables
   - the memory location does not change \( \Rightarrow \) AI-based static analysis to prove constancy of references
   - write accesses are done through the same memory location \( \Rightarrow \) check after analysis

```java
while ( x != null ) {
    for(; x.c < n; x.c++)
        value[x.c]++;
    x=x.next;
}
```

```java
while ( x.size > 0 ) {
    x.size++;
    y.size--;
}
```
analyze the behavior of scopes or program fragments
model only those fields which behave as local variables
• the memory location does not change ⇒ AI-based static analysis to prove constancy of references
• write accesses are done through the same memory location ⇒ check after analysis

while ( x != null ) {
  for(; x.c < n; x.c++)
    value[x.c]++;
  x=x.next;
}

while ( x.size > 0 ) {
  x.size++;
  y.size--;
}

transform the code to replace local fields by variables
Field-sensitive analysis by field-insensitive analysis

1. analyze the behavior of scopes or program fragments
2. model only those fields which behave as **local variables**
   - the memory location does not change ⇒ AI-based static analysis to prove constancy of references
   - write accesses are done through the same memory location ⇒ check after analysis

   ```
   while ( x != null ) {
       for(; x.c<n; x.c++)
           value[x.c]++;
       x=x.next;
   }
   ```

   ```
   while ( x != null ) {
       for(; g<n; g++)
           value[g]++;
       x.c=g;
       x=x.next;
   }
   ```

3. transform the code to replace local fields by variables

4. infer information on the fields through associated **ghost variables**
Conclusions

- **Cost and Termination**
  - Termination: find ranking functions for all loops in the program
  - Termination $\rightarrow$ Bounded resource consumption
  - Cost (for number of instructions) $\rightarrow$ Termination
Conclusions

- **Cost and Termination**
  - Termination: find ranking functions for all loops in the program
  - Termination $\rightarrow$ Bounded resource consumption
  - Cost (for number of instructions) $\rightarrow$ Termination

- **COSTA** has been the first cost and termination analyzer for sequential Java Bytecode
  - It deals with Java libraries
  - It checks termination and computes upper bounds
  - It allows assertions on upper bounds (and thus termination)
Conclusions

- **Cost and Termination**
  - Termination: find ranking functions for all loops in the program
  - Termination $\rightarrow$ Bounded resource consumption
  - Cost (for number of instructions) $\rightarrow$ Termination

- **COSTA** has been the first cost and termination analyzer for sequential Java Bytecode
  - It deals with Java libraries
  - It checks termination and computes upper bounds
  - It allows assertions on upper bounds (and thus termination)

- **PUBS** has been the first generic cost relation solver
  - It is written in Prolog and uses PPL
  - It can be connected to Maxima for further precision in some cases
  - It is currently able to compute upper and lower bounds
Current Projects

- **European Projects:**
    UPM (Germán Puebla) + UCM (Elvira Albert, Lou 83)

- **MEC Projects:**
    UPM (Manuel Hermenegildo)

- **CAM Projects:**
    UCM (Francisco J. López-Fraguas)
Current and Future Work

Some advanced topics:
- Modular and incremental analysis
- Support dynamically allocated data (numeric and reference fields) in cost/termination analysis of OO bytecode
- Asymptotic upper bounds
- Checking of resource usage specifications

Present and future work:
- Handle concurrency during static analysis
  - X10 language (Java-like syntax, different concurrency)
  - ABS (successor of Creol)
- Combine static and dynamic techniques
- Combine testing and resource usage
Main Publications of COSTA

- ESOP’07/LNCS + TCS’11/Elsevier: Generation of cost relations
- SAS’08/LNCS + JAR’11/Springer: Closed-form upper bounds
- ISMM’07/ACM + ISMM’09/ACM + ISMM’10/ACM: Memory consumption
- FM’09/LNCS + SAS’10/LNCS: Treatment of Heap-Allocated Data
- FOPARA’09/LNCS: Checking upper bounds
- APLAS’09/LNCS: Asymptotic bounds
- VMCAI’11/LNCS: Lower-bounds
- LCTES’11/ACM: Task-level analysis for X10
- FM’11/LNCS: Combination of static and dynamic techniques
- PEPM’11/ACM: Verified resource guarantees
- APLAS’11/LNCS: Cost analysis of concurrent OO languages