Critical and High Assurance Requirements Transformed through Engineering Rigour

ARTEMIS Embedded Computing Systems Initiative project

April 2009 – June 2012

3 times in a row the highest ratings

Project partners: aicas GmbH, Atego Ltd, Chalmers University of Technology, Impronova AB, Lero at Dundalk Institute of Technology, Luminis, NLR, QRTECH AB, Radboud Universiteit Nijmegen, The Open Group, Universiteit Twente
CHARTER

- Aim: improve certification process of critical embedded systems
- Automotive, health care, avionics, surveillance
- By means of:
  - Model Driven Development
  - Rule Based Compilation
  - Formal Verification
  - Focus on Real-Time Java (JamaicaVM)
- Radboud University’s focus was on resource analysis and formal specification of the Real-Time Java API
Resource Analysis in CHARTER

- Loop-Bound Analysis
  - Prove termination
  - Prerequisite for other resource analysis
- Memory Usage Analysis (Heap and Stack)
  - In safety and security critical applications: to prevent abrupt termination due to the lack of memory, because output and intermediate structures are too large (DOS attack)
  - To optimise memory management, e.g. by allocation in advance chunks of a heap
  - To be able to physically configure embedded systems for small devices in an optimal way
ResAna: Loop Bound Analysis

Loop Bounds Inference module

COSTA++

VeriFlux
ResAna: Heap Analysis

- Loop Bounds
- Inference module
- COSTA++
- VeriFlux
ResAna: Stack Analysis

- Loop Bounds
- Inference module
- COSTA++
- VeriFlux
Loop Bound Analysis

- ResAna
- Loop Bounds Inference module
- COSTA++
- VeriFlux
Ranking Function

- Decreases in every basic block
- Here: in every loop iteration
- Bounded by zero

```java
1 while (i < 15) {
2     i++;
3 }
```

- Ranking function for the loop above is $15 - i$
Prerequisite for Resource Analysis

1 while (i < 15) {
2     consumeResource();
3     i++;
4 }

```java
while (i < 15) {
    consumeResource();
    i++;
}
```
Applicable Loops

- The basic polynomial interpolation method considers loops with conditions in the following form:

\[ C := sC \mid C_1 \land C_2 \]

\[ sC := e_1 [\lt, \gt, \leq, \geq, =, \neq] e_2 \]

- where \( e_i \) are arithmetical expressions
- i.e. conjunctions over arithmetical (in)equalities
Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a set of $N_d^k = \binom{d+k}{k}$ input values satisfying NCA and the exit condition
3. Interpolate a polynomial from the results
Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a set of $N_d^k = \binom{d+k}{k}$ input values satisfying \textbf{NCA} and the exit condition
3. Interpolate a polynomial from the results
Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a set of $N_d^k = \binom{d+k}{k}$ input values satisfying NCA and the exit condition
3. Interpolate a polynomial from the results
Quadratic Example

```java
public int m(int a, int b, int c) {
    int count=0;
    while (a > 0 && c <= b && c > 0) {
        if ( c == b ) { a--; c = 0; }
        c++;
        count++;
    }
    return count;
}
```

Test runs

1\textsuperscript{st} group: degree 2 NCA on plane
- \(a=1, b=1, c=1\) => count=1
- \(a=1, b=1, c=2\) => count=2
- \(a=1, b=1, c=3\) => count=3
- \(a=1, b=2, c=2\) => count=1
- \(a=1, b=2, c=3\) => count=2
- \(a=1, b=3, c=3\) => count=1

2\textsuperscript{nd} group: degree 1 NCA on plane
- \(a=2, b=1, c=1\) => count=2
- \(a=2, b=1, c=2\) => count=4
- \(a=2, b=2, c=2\) => count=3

3\textsuperscript{rd} group: degree 0 NCA on plane
- \(a=3, b=1, c=1\) => count=3

Find the interpolating polynomial and generate the method annotated with the corresponding ranking function:
\(RF(a, b, c) = a*b - c + 1\)
Soundness

- The procedure itself is unsound
- Use external prover to verify the inferred ranking functions
- KeY: http://www.key-project.org/
- Ranking function can be expressed in JML as a decreases clause

```java
1  //@ decreases i < 15 ? 15 - i : 0;
2  while (i < 15) {
3    i++;
4  }
```
Helicopter View

Java source → Test-based inference procedure → Annotated generated method with a chosen loop → External checking tool (KeY) → Verified RF

Not verifiable automatically
Manual steps

Rejection: repeat testing with a higher degree
Further Reading

O. Shkaravska, R. Kersten, M. van Eekelen.
Test-Based Inference of Polynomial Loop-Bound Functions.
PPPJ’10: Proceedings of the 8th International Conference on
the Principles and Practice of Programming in Java

http://resourceanalysis.cs.ru.nl/
Heap Analysis

- ResAna
- Loop Bounds
- Inference module
- COSTA++
- VeriFlux
The COSTA System

**COSTA** = **COSt** and **Termination Analyzer** for Java Bytecode

**Universidad Complutense de Madrid (UCM)**
- Elvira Albert
- Puri Arenas
- Samir Genaim
- Diego Alonso
- Jesús Correas
- Miguel Gómez-Zamalloa

**Universidad Politécnica de Madrid (UPM)**
- Germán Puebla
- Damiano Zanardini
- Abu Naser Masud
- Diana Ramrez
- José Miguel Rojas
- Guillermo Román

**Aim**: Compute an upper bound to the **cost** of a given program in terms of the **size** of the input.
The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```java
public void f(int n) {
    while (n > 0) {
        int[] array = new int[n];
        n--;
    }
}
```

2. Compute a nonrecursive form of the solution (closed form)

\[ f(n) = 4n^2 \]
The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```java
public void f(int n) {
    while (n > 0) {
        int[] array = new int[n];
        n--;
    }
}
```

Recurrence relation:

\[
\begin{align*}
    f(n) &= 0 & \{n \leq 0\} \\
    f(n) &= 4n + f(n-1) & \{n > 0\}
\end{align*}
\]

2. Compute a nonrecursive form of the solution (closed form)

\[ f(n) = 4n^2 \]
The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

```java
public void f(int n) {
    while (n > 0) {
        int[] array = new int[n];
        n--;
    }
}
```

Recurrence relation:

\[
f(n) = \begin{cases} 
0 & \text{if } n \leq 0 \\
4n + f(n-1) & \text{if } n > 0 
\end{cases}
\]

2. Compute a nonrecursive form of the solution (closed form)

\[f(n) = 4n^2\]
The COSTA System

Approach based on [Wegbreit 1975]:

1. Given a program and a cost model, produce a set of equations specifying the cost of the program.

   ```java
   public void f(int n) {
       while (n > 0) {
           int[] array = new int[n];
           n--;
       }
   }
   ``

   Recurrence relation:
   \[
   f(n) = \begin{cases} 
   0 & \text{if } n \leq 0 \\
   4n + f(n-1) & \text{if } n > 0 
   \end{cases}
   \]

2. Compute a nonrecursive form of the solution (closed form)

   \[ f(n) = 4n^2 \]
PUBS System

PUBS = Practical Upper Bounds Solver

\[
\begin{align*}
  f(n) &= 0 & \{n \leq 0\} \\
  f(n) &= 4n + f(n-1) & \{n > 0\}
\end{align*}
\]
Overview of COSTA architecture

Input program

COSTA

CRS

PUBS

Closed form
Our extensions

- Interpolation-based height analysis (COSTA++)
- Simplification of bounds
- Correct array-size analysis
- Virtual-machine specialisation
- PUBS is based on the notion of \textit{evaluation trees}.

\[
\begin{align*}
T(x) &= 1 & \{ x \leq 1 \} \\
T(x) &= 2x + T(x-1) + T(x-2) & \{ x > 1 \}
\end{align*}
\]
Interpolation-based height analysis

- **Call-chain height** $h(x)$: Upper-bound to the maximum number of unfoldings that may be undergone to reach a base case.
- It is closely related to the concept of **ranking function**.

\[ h(x) = x \]

A. Podelski, A. Rybalchenko.  
A Complete Method for the Synthesis of Linear Ranking Functions.  
5th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2004
Interpolation-based height analysis

- **Call-chain height** $h(x)$: Upper-bound to the maximum number of unfoldings that may be undergone to reach a base case.
- It is closely related to the concept of **ranking function**.

\[
T(x) \rightarrow T(x-1) \rightarrow T(x-2) \rightarrow \ldots \rightarrow T(0)
\]

\[h(x) = x\]

A. Podelski, A. Rybalchenko.

*A Complete Method for the Synthesis of Linear Ranking Functions.*

5th International Conference on Verification, Model Checking and Abstract Interpretation, VMCAI 2004
Interpolation-based height analysis

- The length of the maximal call chain associated with a CRS may not linearly depend on the sizes of the arguments.

Example

\[
T(x, y) = \text{nat}(x) \quad \{x = 0, y = 0\}
\]

\[
T(x, y) = \text{nat}(x) + T(x - 1, x - 1) \quad \{x > 0, y = 0\}
\]

\[
T(x, y) = \text{nat}(x) + T(x, y - 1) \quad \{x \geq 0, y > 0\}
\]
Interpolation-based height analysis

- The length of the maximal call chain associated with a CRS may not linearly depend on the sizes of the arguments.

Example

\[
T(x, y) = \text{nat}(x) \quad \{x = 0, y = 0\}
\]

\[
T(x, y) = \text{nat}(x) + T(x-1, x-1) \quad \{x > 0, y = 0\}
\]

\[
T(x, y) = \text{nat}(x) + T(x, y-1) \quad \{x \geq 0, y > 0\}
\]

\[
h(x) = 0.5x^2 + 0.5x + y
\]
Simplification of bounds

\[(n) \ast
  ((n) \ast (c((\text{java.lang.Object}, 1)) +
    c((\text{java.lang.Object}, 2)))) +
  (n) \ast (c((\text{java.lang.Object}, 1)) +
    c((\text{java.lang.Object}, 2)))) +
  (n) \ast (c((\text{java.lang.Object}, 1)) +
    c((\text{java.lang.Object}, 2))))\]

\[\Downarrow\]

\[6n^2 \ast (\text{java.lang.Object})\]
Correct array-size analysis

- Arrays are packed into data structures of the virtual machine.
- Currently Costa calculates memory cost for an array using
  \[ \text{new } T[n] \Rightarrow n \ast c(\text{size(primitiveType}(T), 1)) \]
- Cost model should be
  \[ \text{new } T[n] \Rightarrow \text{array}(n, c(\text{size(primitiveType}(T), 1))) \]
Jamaica VM

- Real-Time Java virtual machine
- Developed by Aicas GmbH (partner in CHARTER)
- Real-Time Garbage Collection by using an innovative memory allocator
- Many RTOS: VxWorks, QNX, Linux-variants, RTEMS
- Support for multicore execution
Virtual-machine specialisation

- An array is saved as an 8-tree with data stored only at the leaves (JamaicaVM)
- Every object/memory chunk in JamaicaVM is 32 bytes
Virtual-machine specialisation

<table>
<thead>
<tr>
<th>( 9 \leq n \leq 16 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>gc</td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>monitor</td>
</tr>
<tr>
<td>count</td>
</tr>
<tr>
<td>0..7</td>
</tr>
<tr>
<td>8..15</td>
</tr>
</tbody>
</table>

- \( a[0] \)
- \( a[1] \)
- \( a[2] \)
- \( a[3] \)
- \( a[4] \)
- \( a[5] \)
- \( a[6] \)
- \( a[7] \)
- \( a[8] \)
- \( a[9] \)
- \( a[10] \)
- \( a[11] \)
- \( a[12] \)
- \( a[13] \)
- \( a[14] \)
- \( a[15] \)
Virtual-machine specialisation

\[ 65 \leq n \leq 128 \]

- gc
- type
- monitor
- count
- 0.63
- 64..127
- 64..71
- 72..79
- 80..87
- 88.95
- 96..103
- 104..111
- 112..117
- 118..127
- a[0]
- a[8]
- a[16]
- a[24]
- a[33]
- a[40]
- a[48]
- a[56]
- a[57]
- a[58]
- a[59]
- a[60]
- a[61]
- a[62]
- a[63]
- a[64]
- a[65]
- a[66]
- a[67]
- a[68]
- a[69]
- a[70]
- a[71]
Virtual-machine specialisation

- Specialise COSTA’s results for JamaicaVM
- Calculate sound upper-bounds for array sizes:

\[
\text{arrayblocks}(1..4) = 1
\]

\[
\text{arrayblocks}(n) = \left\lceil \frac{n}{8} \right\rceil + \text{arrayblocks}\left(\left\lceil \frac{n}{8} \right\rceil \right)
\]

- Replace symbolic object sizes by actual sizes

\[
6n^2 \times (\text{java.lang.Object})
\]

\[
\downarrow
\]

\[
192n^2
\]
Further reading

E. Albert, P. Arenas, S. Genaim, G. Puebla., D.Zanardini
Cost Analysis of Java Bytecode.
16th European Symposium on Programming, ESOP 2007

M. Montenegro, O. Shkaravska, M. van Eekelen, R. Peña
Interpolation-based height analysis for improving a recurrence solver.
2nd International Workshop on Foundational Practical Aspects of Resource Analysis, FOPARA 2011

http://costa.ls.fi.upm.es/
http://resourceanalysis.cs.ru.nl/
Stack Analysis

- ResAna
- Loop Bounds
- Inference module
- COSTA++
- VeriFlux

Rody Kersten
Making Resource Analysis Practical for Real-Time Java
October 25, 2012
VeriFlux

- Aicas GmbH Karlsruhe
- Combines various static analyses in one tool
  - Data-flow
  - Control-flow
  - ...
- Finds various problems
  - Run-time exceptions
  - Deadlocks
  - Race conditions
  - ...

http://www.aicas.com/veriflux.html
Stack Analysis

- COSTA++ outputs symbolic upper bound on the depth of recursion for a given method (height of the call tree)
- Use as measured by annotation
  - Ranking function
  - Similar to decreases annotation
  - But for recursion instead of iterations
- Use VeriFlux’s data-flow analysis to combine with input values
- Use VeriFlux to combine with stack-frame sizes
Stack Analysis

```java
static int fib(int n) {
    if (n < 2)
        return n;
    return fib(n-1) + fib(n-2);
}

public static void main(String[] args) {
    fib(21);
}
```
Stack Analysis

```java
1 //@ measured_by n;
2 static int fib(int n) {
3     if (n < 2)
4         return n;
5     return fib(n-1) + fib(n-2);
6 }
```

```java
1 public static void main(String [] args) {
2     fib(21);
3 }
```
Stack Analysis

```java
//@ measured by n;
static int fib(int n) {
    if (n < 2)
        return n;
    return fib(n - 1) + fib(n - 2);
}
```

```java
public static void main(String[] args) {
    fib(21);
}
```

\[21 \cdot 56 + 36 = 1212 \text{ bytes}\]
Practical Experience

- Loop-Bound Inference tested on several case-studies: could analyse 66% of the loops
  - Collision detector from Hunt et al. *Provably correct loops bounds for realtime Java programs*, JTRES’06
  - DIANA avionics package
  - $CD_X$ Collision Detector package
- ResAna was used by NLR in the development of a safety-critical avionics application
  - Found easy to use
  - Feedback led to several improvements
Future Work

- Loop Bound Analysis
  - Larger case study to identify most needed improvements
  - Combine with syntactical pattern-matching method (Fulara and Jakubczyk. *Practically Applicable Formal Methods*, SOFSEM’10)
  - Use JML annotations

- Heap Space Analysis
  - Specialisation for OpenJDK
Conclusions

- Various resources analyses combined into one tool: ResAna
  - Loop-bound analysis
    - Polynomial ranking functions for loops
    - Based on interpolation of test-run results
  - Heap analysis
    - Enhanced version of COSTA
    - Interpolation-based height analysis
    - Simplification of bounds
    - Correct array-size analysis
    - Virtual-machine specialisation
  - Stack analysis
    - Use COSTA to generate measured by annotation
    - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana
Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate measured by annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana
Conclusions

Various resources analyses combined into one tool: ResAna

Loop-bound analysis
- Polynomial ranking functions for loops
- Based on interpolation of test-run results

Heap analysis
- Enhanced version of COSTA
- Interpolation-based height analysis
- Simplification of bounds
- Correct array-size analysis
- Virtual-machine specialisation

Stack analysis
- Use COSTA to generate measured by annotation
- Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana
Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate measured by annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana
Conclusions

- Various resources analyses combined into one tool: ResAna
- Loop-bound analysis
  - Polynomial ranking functions for loops
  - Based on interpolation of test-run results
- Heap analysis
  - Enhanced version of COSTA
  - Interpolation-based height analysis
  - Simplification of bounds
  - Correct array-size analysis
  - Virtual-machine specialisation
- Stack analysis
  - Use COSTA to generate measured by annotation
  - Use VeriFlux to calculate stack usage

http://resourceanalysis.cs.ru.nl/resana