Test-Based Inference of Polynomial Loop-Bound Functions

Olha Shkaravska\textsuperscript{1}  Rody Kersten\textsuperscript{1}  Marko van Eekelen\textsuperscript{1,2}

\textsuperscript{1}Radboud University Nijmegen
\textsuperscript{2}Open University of the Netherlands

PPPJ 2010
September 17th, Vienna, Austria

This research is funded by the CHARTER project and the Laboratory for Quality Software (LaQuSo)
Presentation Outline

1. Motivation and Aim
2. Basic Procedure
3. Quadratic Example
4. Extensions
5. Prototype
6. Conclusions
<table>
<thead>
<tr>
<th></th>
<th>Presentation Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Motivation and Aim</td>
</tr>
<tr>
<td>2</td>
<td>Basic Procedure</td>
</tr>
<tr>
<td>3</td>
<td>Quadratic Example</td>
</tr>
<tr>
<td>4</td>
<td>Extensions</td>
</tr>
<tr>
<td>5</td>
<td>Prototype</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
</tr>
</tbody>
</table>
Motivation

- Loop Bound: upper-bound on the number of iterations
- Why?
  - Prove termination
  - Bounding runtime
    - Real-time systems
  - Bounding memory consumption
    - Economical motives
    - Prevent abrupt termination
  - Compiler optimisations
Loop-Bound Function (LBF)

- Expresses an upper bound on the number of loop iterations depending on (some of) the program variables
- Can be used to bound the number of iterations for *arbitrary* values of these variables
Loop-Bound Function: Example

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

- The tightest LBF for this loop is $15 - i$
- Can be used to calculate the number of loops for arbitrary $i$
- Proving this bound also proves termination
Applicable Loops

- We consider loops with conditions in the following form:

\[ C ::= sC \mid C_1 \land C_2 \mid C_1 \lor C_2 \]
\[ sC ::= e_1 [\lt, \gt, \leq, \geq, =, \neq] e_2 \]

- where \( e_i \) are arithmetical expressions
- i.e. first-order propositional logic expressions over numerical (in)equalities
Motivation and Aim

Basic Procedure

Quadratic Example

Extensions

Prototype

Conclusions

Presentation Outline

1. Motivation and Aim
2. Basic Procedure
3. Quadratic Example
4. Extensions
5. Prototype
6. Conclusions

Olha Shkaravska, Rody Kersten, Marko van Eekelen
Radboud University Nijmegen, Open University of the Netherlands
Test-Based Inference of Polynomial Loop-Bound Functions
Helicopter View

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

Rejection: repeat testing with a higher degree

Not verifiable automatically
Manual steps
Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a *well-chosen* set of input values
3. Interpolate a polynomial from the results
Test-Based Inference Procedure: Example

public void m(int i) {
    int count = 0;
    while (i < 15) {
        i++;
        count++;
    }
    return count;
}

Test runs
i=0 => count = 15
i=1 => count = 14

Find the interpolating polynomial
p(i) = 15 - i;

Degree of a loop bound (e.g. d=1)
A 1-variable polynomial $p(z)$ of degree $d$ can be written as:

$$a_0 + a_1 z + \ldots + a_d z^d = p(z)$$

We need the values of $p(z)$ in $d + 1$ pairwise different points to interpolate.

These form a system of equations with a unique solution.
The system of equations can be written as:

\[
\begin{pmatrix}
1 & z_0 & \cdots & z_0^{d-1} & z_0^d \\
1 & z_1 & \cdots & z_1^{d-1} & z_1^d \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
1 & z_{d-1} & \cdots & z_{d-1}^{d-1} & z_{d-1}^d \\
1 & z_d & \cdots & z_d^{d-1} & z_d^d
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
\vdots \\
a_{d-1} \\
a_d
\end{pmatrix}
=
\begin{pmatrix}
p(z_0) \\
p(z_1) \\
\vdots \\
p(z_{d-1}) \\
p(z_d)
\end{pmatrix}
\]

A unique interpolating polynomial exists if \( z_0, \ldots, z_d \) are pairwise different, i.e. the Vandermonde determinant of the matrix is non-zero.
Node Configuration A: 2-dimensional

- The condition ensuring existence of a unique multivariate polynomial \( p(z_1, \ldots, z_k) \) that interpolates multivariate data is not trivial.

- A condition which ensures it is NCA.

A set of nodes \( W \subset \mathbb{R}^2 \) lies in a 2-dimensional NCA if there exist lines \( \gamma_1, \ldots, \gamma_{d+1} \) in the space \( \mathbb{R}^2 \), such that:

- \( d + 1 \) nodes of \( W \) lie on \( \gamma_{d+1} \).
- \( d \) nodes of \( W \) lie on \( \gamma_d \setminus \gamma_{d+1} \).
- \( \ldots \).
- \( 1 \) node of \( W \) lies on \( \gamma_1 \setminus (\gamma_2 \cup \ldots \cup \gamma_{d+1}) \).
Typical NCA Instance: Grid

dimension = 2
degree = 2
The NCA condition is extended to higher dimensions inductively. A set of $k$-dimensional nodes is in NCA for a degree $d$ if

- It consists of a set of $d$ hyperplanes
- All lying in $(k - 1)$-dimensional NCA
- One for degree $d$, one for $d - 1$, ..., and one for degree 1
Adding Loop-Conditions to Node Search

- Test-nodes must not only satisfy NCA-configuration, but also the loop condition.
- When the loop-condition is not satisfied, the loop is executed 0 times.
- If we take such points into account, any interpolation would yield an incorrect bound.
Example with Condition $i < x$
Algorithm for Finding Test-Nodes

1. Using a global optimisation procedure, find bounding box (optional)
2. Split each dimension into \( d + 1 \) hyperplanes, i.e. construct grid
3. Check if the nodes on this grid satisfy NCA configuration
   - Yes: done
   - No: increase grid granularity
Expressing the LBF in JML

```java
// @ assignable i;
//@ loop_invvariant true;
//@ decreases 15 - i;
public void m(int i) {
    while (i < 15) {
        i++; 
    }
}
```
1. Motivation and Aim
2. Basic Procedure
3. Quadratic Example
4. Extensions
5. Prototype
6. Conclusions
Quadratic Example

- Using our method, polynomial LBFs can be inferred
- Not restricted to monotonic polynomials
- For instance, the quadratic LBF $x^2 - xi - j + 1$ for the following loop:

```java
1 while (i<x && j=x) {
2   if (j=x) { i++; j = 0; }
3   j++;
4 }
```
Remember: Test-Based Approach

1. Instrument loop with a counter
2. Do test runs for a well-chosen set of input values
3. Interpolate a polynomial from the results
Quadratic Example: Instrument With Counter (Step 1/3)

```java
public void m(int x, int i, int j) {
    while (i < x && j <= x) {
        if (j==x) { i++; j = 0;}
        j++;
    }
}

public int m(int x, int i, int j) {
    int count=0;
    while (i < x && j <= x) {
        if (j==x) { i++; j = 0;}
        j++;
        count++;
    }
    return count;
}
```
Quadratic Example: Run Tests (Step 2/3)

```java
public int m(int x, int i, int j) {
    int count = 0;
    while (i < x && j <= x) {
        if (j == x) { i++; j = 0; }
        j++;
        count++;
    }
    return count;
}
```

Test runs (dimension=3, degree=2)

1st group: degree 2 NCA on plane
- x=3, i=1, j=1 => count =6
- x=4, i=1, j=1 => count=12
- x=5, i=1, j=1 => count=20
- x=3, i=2, j=1 => count=3
- x=4, i=2, j=1 => count=8
- x=4, i=3, j=1 => count=4

2nd group: degree 1 NCA on plane
- x=3, i=1, j=2 => count=5
- x=4, i=1, j=2 => count=11
- x=4, i=2, j=2 => count=7

3rd group: degree 0 NCA on plane
- x=4, i=1, j=3 => count=10

Degree of a loop bound (e.g. d=2)
Quadratic Example: Interpolate (Step 3/3)

Test runs (dimension=3, degree=2)

1st group: degree 2 NCA on plane
x=3, i=1, j=1 => count =6
x=4, i=1, j=1 => count=12
x=5, i=1, j=1 => count=20
x=3, i=2, j=1 => count=3
x=4, i=2, j=1 => count=8
x=4, i=3, j=1 => count=4

2nd group: degree 1 NCA on plane
x=3, i=1, j=2 => count=5
x=4, i=1, j=2 => count=11
x=4, i=2, j=2 => count=7

3rd group: degree 0 NCA on plane
x=4, i=1, j=3 => count=10

Find the interpolating polynomial and generate the method annotated with the corresponding loop bound:

\[ p(x, i, j) = x^2 - x^1 - j + 1; \]

Olha Shkaravska, Rody Kersten, Marko van Eekelen
Radboud University Nijmegen, Open University of the Netherlands

Test-Based Inference of Polynomial Loop-Bound Functions
Presentation Outline

1. Motivation and Aim
2. Basic Procedure
3. Quadratic Example
4. Extensions
5. Prototype
6. Conclusions
Extensions to the basic procedure

- Dealing with LBFs with rational or real coefficients: ceiling
- Piecewise LBFs for disjunctive loop-conditions
- Branch-splitting to handle if-then-else constructs inside the loop body
Dealing with LBFs with Rational or Real Coefficients

1 while (start < end) {
2   start += 4;
3 }

- The exact number of iterations is $\lceil \frac{\text{end} - \text{start}}{4} \rceil$
- The bound is a polynomial over rationals, ceiled
- In general, when the coefficients of the polynomial LBF $p(\bar{z})$ are not naturals, the actual bound should be read as $\lceil p(\bar{z}) \rceil$
- For correct interpolation, test-nodes must lie 4 apart, this is true for all loops with a linear bound and increment $\neq 1$
We consider loop conditions in DNF over arithmetical (in)equality:
\[
\bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} (e_{lij} b e_{rij}), \text{ with } b \in \{<,>,=,\neq,\leq,\geq\}
\]

For every sub-formula \( \bigwedge_{j=1}^{m_i} (e_{lij} b e_{rij}) \) we infer and check separately a particular polynomial bound \( p_i(\bar{x}) \)

Altogether they form a piecewise loop-bound function:

\[
\begin{align*}
&\begin{cases} 
  p_1(\bar{x}) & \text{if } \bigwedge_{j=1}^{m_1} (e_{l1j} b e_{r1j})(\bar{x}) \\
  \vdots & \\
  p_n(\bar{x}) & \text{if } \bigwedge_{j=1}^{m_n} (e_{lnj} b e_{rnj})(\bar{x}) \\
  0 & \text{else}
\end{cases}
\end{align*}
\]
Branch-splitting

- **Problem**
  - Branches inside a loop may have different effect on LBF

- **Solution**
  - Analyse each branch separately
  - And pick worst-case

- Leads to overestimation

- But can handle all loops where taking one of the branches at each execution gives an upper-bound

Example of a loop that does not satisfy this property:

```plaintext
1 while (i > 0)
2   if (i % 2 == 0) i -= 3;
3   else i++; 
```
Presentation Outline

1. Motivation and Aim
2. Basic Procedure
3. Quadratic Example
4. Extensions
5. Prototype
6. Conclusions
Prototype

- We have implemented a prototype in Java
- Implements:
  - Basic procedure
  - Extensions for rational and real coefficients
  - Piecewise LBFs for disjunctive loop-conditions
- Tested on a series of case-studies
- With KeY as the external tool to verify the inferred LBFs
We have tested the prototype on some case studies, supplied by the CHARTER project.

<table>
<thead>
<tr>
<th>#loops</th>
<th>Applic.</th>
<th>%</th>
<th>Exact LBF inf.</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hunt et al</td>
<td>2</td>
<td>2</td>
<td>100%</td>
<td>2</td>
</tr>
<tr>
<td>DIANA</td>
<td>4</td>
<td>4</td>
<td>100%</td>
<td>4</td>
</tr>
<tr>
<td>CD$_X$</td>
<td>38</td>
<td>23</td>
<td>61%</td>
<td>23</td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td>29</td>
<td>66%</td>
<td>29</td>
</tr>
</tbody>
</table>
Motivation and Aim

Basic Procedure

Quadratic Example

Extensions

Prototype

Conclusions

Olha Shkaravska, Rody Kersten, Marko van Eekelen

Radboud University Nijmegen, Open University of the Netherlands

Test-Based Inference of Polynomial Loop-Bound Functions
Conclusions

- Novel, general technique for inferring loop-bound *functions*
- To our knowledge, the first to infer non-linear, non-monotonic loop-bound functions for Java
- Complementary to syntax-driven methods, since it is more general and can solve certain more complex cases, such as quadratic bounds
- Inferred LBFs are provable!
- Will be applied in the CHARTER toolchain (EU project with partners producing safety-critical software for the Aviation, Medical, Automotive and Surveillance markets) and LaQuSo (collaboration of RU with dutch technical universities)
- Download prototype from resourceanalysis.cs.ru.nl