Ranking Functions for Loops with Disjunctive Exit-Conditions

Rody Kersten[1] Marko van Eekelen[1,2]

[1]Institute for Computing and Information Sciences (iCIS), Radboud University Nijmegen

[2]School for Computer Science, Open University of the Netherlands

2nd International Workshop on the Foundational and Practical Aspects of Resource Analysis (FOPARA'11), Madrid

May 19, 2011

Presentation Outline

Introduction

Basic Procedure

Piecewise Ranking Functions

Condition Jumping

Conclusions

Ranking Function

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

```
while (i < 15) {
  i++;
}
```

- Ranking function for the loop above is $15 - i$

Motivation and Aim

- Prove termination
- Bounding runtime
- Compiler optimisations
- Resource Analysis

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

Motivation and Aim

- Prove termination
- Bounding runtime
- Compiler optimisations
- **Resource Analysis**

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

Motivation and Aim

- Prove termination
- Bounding runtime
- Compiler optimisations
- **Resource Analysis**

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

Presentation Outline

Presentation Outline

Inference of Polynomial Loop Ranking Functions

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

Applicable Loops

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

$$C := sC \mid C_1 \wedge C_2$$
$$sC := e_1 \; [<, >, \leq, \geq, =, \neq] \; e_2$$

- 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 **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

```
public void m(int a, int b, int c) {
        while (a > 0 && c <= b && c > 0) {
                if ( c == b ) { a−−; c = 0; }
                c++;
        }
}
```

```
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;
}
```

Expected degree of polynomial (here: d=2)

Test runs

1st 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

2nd 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

3rd 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

```
//@ decreases i < 15 ? 15 - i : 0;
while (i < 15) {
  i++;
}
```

Presentation Outline

Any loop ranking function is piecewise...

```
while (i < 15) {
  i++;
}
```

Its ranking function is actually:

$$\begin{cases} 15 - \mathtt{i} & \text{if } (\mathtt{i} < 15) \\ 0 & \text{else} \end{cases}$$

Non-Trivial Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

It's ranking function is non-trivially piecewise:

$$\begin{cases} 20 - \mathtt{i} & \text{if } (\mathtt{i} > 0) \land (\mathtt{i} < 20) \\ \mathtt{i} - 50 & \text{if } \mathtt{i} > 50 \\ 0 & \text{else} \end{cases}$$

Expressing Piecewise Ranking Functions in JML

```
//@ decreases (i>0&&i<20) ? 20-i : (i>50 ? i-50 : 0);
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

Applicable Loops

- The extended method considers loops with conditions in the following form:

$$C := sC \mid C_1 \wedge C_2 \mid C_1 \vee C_2$$
$$sC := e_1 \; [<, >, \leq, \geq, =, \neq] \; e_2$$

- where e_i are arithmetical expressions
- i.e. first-order propositional logic expressions over arithmetical (in)equalities

Extending the Basic Procedure: Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

1. Split up the condition into disjunctive parts:
   - $i > 0 \wedge i < 20 \wedge \neg(i > 50)$
   - $i > 50 \wedge \neg(i > 0 \wedge i < 20)$
   - $i > 0 \wedge i < 20 \wedge i > 50$
2. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

1. Split up the condition into disjunctive parts:
   - $i > 0 \wedge i < 20 \wedge \neg(i > 50)$
   - $i > 50 \wedge \neg(i > 0 \wedge i < 20)$
   - $i > 0 \wedge i < 20 \wedge i > 50$
2. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

1. Split up the condition into disjunctive parts:
   - $i > 0 \wedge i < 20 \cancel{\wedge \neg(i > 50)}$
   - $i > 50 \cancel{\wedge \neg(i > 0 \wedge i < 20)}$
   - $\cancel{i > 0 \wedge i < 20 \wedge i > 50}$
2. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

1. Split up the condition into disjunctive parts:
   - $i > 0 \wedge i < 20$
   - $i > 50$
2. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

1. Split up the condition into disjunctive parts:
   - $i > 0 \wedge i < 20$
   - $i > 50$
2. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Example

```
while ((i>0 && i<20) || i>50) {
  if (i>50) i--;
  else i++;
}
```

$$\begin{cases} 20 - \mathtt{i} & \text{if } (\mathtt{i} > 0) \land (\mathtt{i} < 20) \\ \mathtt{i} - 50 & \text{if } \mathtt{i} > 50 \\ 0 & \text{else} \end{cases}$$

Extending the Basic Procedure: Generic

1. Put the condition in Disjunctive Normal Form
2. Split up the condition into its disjunctive pieces
3. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Generic

1. Put the condition in Disjunctive Normal Form
2. Split up the condition into its disjunctive pieces
3. Execute the basic procedure separately for each of the pieces

Extending the Basic Procedure: Generic

1. Put the condition in Disjunctive Normal Form
2. Split up the condition into its disjunctive pieces
3. Execute the basic procedure separately for each of the pieces

Presentation Outline

Example

```
while ((i>0 && i<20) || i>22) {
  if (i>22) i--;
  else i+=4;
}
```

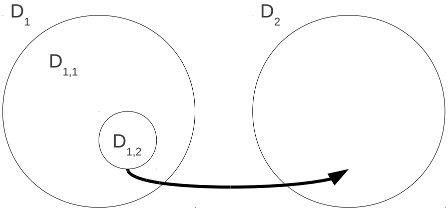
$$\begin{cases} \lceil (20 - \mathtt{i})/4 \rceil & \text{if } (\mathtt{i} > 0) \wedge (\mathtt{i} < 20) \\ \mathtt{i} - 22 & \text{if } \mathtt{i} > 22 \\ 0 & \text{else} \end{cases}$$

Example

```
while ((i>0 && i<20) || i>22) {
  if (i>22) i--;
  else i+=4;
}
```

$$\begin{cases} \lceil (20 - \mathtt{i})/4 \rceil + 1 & \text{if } (\mathtt{i} > 0) \wedge (\mathtt{i} < 20) \wedge i \bmod 4 = 3 \\ \lceil (20 - \mathtt{i})/4 \rceil & \text{if } (\mathtt{i} > 0) \wedge (\mathtt{i} < 20) \wedge i \bmod 4 \neq 3 \\ \mathtt{i} - 22 & \text{if } \mathtt{i} > 22 \\ 0 & \text{else} \end{cases}$$

What happens...



Detection of Condition Jumping: Example

```
while ((i>0 && i<20) || i>22) {
  if (i>22) i--;
  else i+=4;
}
```

$$next_i(i) = \begin{cases} \mathtt{i} - 1 & \text{if } \mathtt{i} > 22 \\ \mathtt{i} + 4 & \text{if } \neg(\mathtt{i} > 22) \end{cases}$$

```
(declare-fun i () Int)
(define-fun nexti ((x Int)) Int
    (ite (> x 22) (- x 1) (+ x 4)))
(assert (and (and (> i 0) (< i 20))
    (> (nexti i) 22)))
(check-sat)
```

Detection of Condition Jumping: Example

```
while ((i>0 && i<20) || i>22) {
  if (i>22) i--;
  else i+=4;
}
```

$$next_i(i) = \begin{cases} \mathtt{i} - 1 & \text{if } \mathtt{i} > 22 \\ \mathtt{i} + 4 & \text{if } \neg(\mathtt{i} > 22) \end{cases}$$

```
(declare-fun i () Int)
(define-fun nexti ((x Int)) Int
    (ite (> x 22) (- x 1) (+ x 4)))
(assert (and (and (> i 0) (< i 20))
    (> (nexti i) 22)))
(check-sat)
```

Detection of Condition Jumping: Example

```
while ((i>0 && i<20) || i>22) {
  if (i>22) i--;
  else i+=4;
}
```

$$next_i(i) = \begin{cases} \mathtt{i} - 1 & \text{if } \mathtt{i} > 22 \\ \mathtt{i} + 4 & \text{if } \neg(\mathtt{i} > 22) \end{cases}$$

```
(declare-fun i () Int)
(define-fun nexti ((x Int)) Int
    (ite (> x 22) (- x 1) (+ x 4)))
(assert (and (and (> i 0) (< i 20))
    (> (nexti i) 22)))
(check-sat)
```

Detection of Condition Jumping: Generic

- Symbolically execute the loop body to find a `next` function for each program variable
- Use SMT-solver to search for a model that satisfies one piece first and another after execution of the loop body

Finding Models for Condition Jumping: Example

Find all nodes that jump from the piece with condition $i > 0 \wedge i < 20$ into the piece with condition $i > 22$. Using an SMT-solver:

1. Find all nodes that jump directly into the other piece: $\{19\}$
2. Find all nodes that can jump to $\{19\}$, $\{3, 7, 11, 15\}$ and add them to the list of jumping nodes: $\{3, 7, 11, 15, 19\} = \{i \mid i \bmod 4 = 3 \wedge i > 0 \wedge i < 20\}$

Finding Models for Condition Jumping: Example

Find all nodes that jump from the piece with condition $i > 0 \wedge i < 20$ into the piece with condition $i > 22$. Using an SMT-solver:

1. Find all nodes that jump directly into the other piece: $\{19\}$
2. Find all nodes that can jump to $\{19\}$, $\{3, 7, 11, 15\}$ and add them to the list of jumping nodes: $\{3, 7, 11, 15, 19\} = \{i \mid i \bmod 4 = 3 \wedge i > 0 \wedge i < 20\}$

Finding Models for Condition Jumping: Example

Find all nodes that jump from the piece with condition $i > 0 \wedge i < 20$ into the piece with condition $i > 22$. Using an SMT-solver:

1. Find all nodes that jump directly into the other piece: $\{19\}$
2. Find all nodes that can jump to $\{19\}$, $\{3, 7, 11, 15\}$ and add them to the list of jumping nodes:
   $\{3, 7, 11, 15, 19\} = \{i \mid i \bmod 4 = 3 \wedge i > 0 \wedge i < 20\}$

Finding Models for Condition Jumping: Generic

J is the set of models of which it is known that condition jumping occurs, Q is a queue of models, find all models that jump from b_1 to b_2:

1. Is there a model $\bar{v}$ for which $b_1(\bar{v}) \wedge b_2(next(\bar{v})) \wedge \bar{v} \notin J$?
   - SAT $\rightarrow$ Add $\bar{v}$ to J and Q, goto 1.
   - UNSAT $\rightarrow$ Goto 2.
2. Q empty?
   - Yes $\rightarrow$ Done.
   - No $\rightarrow$ Goto 3.
3. Pop a model $\bar{q}$ off the queue Q. Is there a model $\bar{v}$ for which $b_1(\bar{v}) \wedge next(\bar{v}) = \bar{q} \wedge \bar{v} \notin J$?
   - SAT $\rightarrow$ Add $\bar{v}$ to J and Q, goto 3.
   - UNSAT $\rightarrow$ Goto 2.

Generating Ranking Functions: Example

- We now know the set $\{i \mid i \bmod 4 = 3 \wedge i > 0 \wedge i < 20\}$ for which jumping occurs
- So, we can split the condition $i > 0 \wedge i < 20$ into two: $i > 0 \wedge i < 20 \wedge i \,\texttt{mod} = 3$ and $i > 0 \wedge i < 20 \wedge i \,\texttt{mod} \neq 3$
- We can then apply the basic method separately to each of these disjunctive pieces

Generating Ranking Functions: Generic

- We now know the set $D_{1,2}$ for which jumping occurs
- So, we can split the condition b_1 into two: $b_1(\bar{v}) \wedge \bar{v} \in D_{1,2}$ and $b_1(\bar{v}) \wedge \bar{v} \notin D_{1,2}$
- We can then apply the basic method to each of these disjunctive pieces

Multi-Jumping

1. DNF-split into n conditions
2. For each i and j, $1 \leq i < j \leq n$, detect jumping from D_i to D_j. Build a list J of jumping pairs (D_x, D_y) for which condition jumping from D_x to D_y can occur.
3. If there are no more jumping pairs (D_x, D_y) for which D_x is unflagged, done! Else, goto 4.
4. Pop a jumping pair (D_x, D_y) off J, for which D_x is unflagged.
5. Find the set $D_{x,2}$ of all nodes in D_x from which jumping to D_y occurs and, dually, the set $D_{x,1}$ for which no jumping to D_y occurs. Replace any condition pair (D_x, D_z) in J by $(D_{x,1}, D_z)$. Add $(D_{x,2}, D_y)$ to J.
   - If $D_{x,1} = \emptyset$, flag $D_{x,2}$ as complete, goto 3.
   - Else, for any jumping pair (D_z, D_x) in J (i.e. for which jumping from D_z to D_x can occur), unflag D_z, detect jumping into $D_{x,1}$ and $D_{x,2}$ and update J accordingly. Goto 3.

Presentation Outline

Conclusions

- Extension to the method presented at PPPJ'10, which can infer *polynomial* ranking functions:
  - Definition of Condition Jumping
  - Detection of Condition Jumping
  - Infer ranking functions for loops in which condition jumping occurs
- Ranking functions for loops can be used in the creation of a *global* ranking function in order to prove termination
- If the body of a loop with ranking function $RF(\bar{v})$ consumes n resources, then we know that the whole loop consumes $RF(\bar{v}) \cdot n$ resources

Implementation: ResAna

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

- The basic procedure and DNF-splitting (minus removal of unsatisfiable pieces) have been implemented
- Future work: implement condition jumping solution