Difference Constraints: An adequate Abstraction for Complexity Analysis of Imperative Programs

Florian Zuleger
Technische Universität Wien
FMCAD, 28.09.2015

Joint work with Moritz Sinn, Helmut Veith
Procedure `foo(uint n)`

```plaintext
x = n;
y = n;
while (x > 0)
    t1: { x--; 
        y = y + 2;
    }
z = y;
while (z > 0)
    t2: z--; 
```

**Local Bound(t1): x**

Variable that decreases when t1 is executed.
Bounds and Complexity

foo(uint n)
x = n;
y = n;
while(x > 0)
t1:{
    x--;
    y = y + 2;
}
z = y;
while(z > 0)
t2: z--;

Local Bound(t1): x
Transition Bound(t1): n

# of visits to transition t1
Bounds and Complexity

```c
foo(uint n)
x = n;
y = n;
while(x > 0)
  t1: { x--; 
    y = y + 2;
  }
z = y;
while(z > 0)
  t2: z--;
```

Local Bound(t1): x
Transition Bound(t1): n
Local Bound(t2): z
Variable Bound(y): 3n

Invariant of shape
\[ y \leq 3n \]
Bounds and Complexity

```c
foo(uint n) {
    x = n;
    y = n;
    while (x > 0) {
        x--;
        y = y + 2;
    }
    z = y;
    while (z > 0) {
        z--;
    }
}
```

- Local Bound (t1): x
- Transition Bound (t1): n
- Local Bound (t2): z
- Variable Bound (y): 3n
- Transition Bound (t2): 3n
- Complexity: 4n
Bounds and Complexity

Bound Analysis:
• # of visits to a transition
• # of visits to multiple transitions
• # of iterations of a loop
• resource consumption of a program
• complexity of a program
• upper bound on the value of a variable

Intuition: All these bound analysis problems are related and can be reduced to each other.

Introduce a counter `c` and increment at places of interest.
Applications of Bound Analysis

**Verification:**
- Computing bounds on resource consumption (CPU time, memory, bandwidth, ...)
- Termination analysis with quantitative information on program progress

**Program understanding:**
- Static profiling
- Understanding program performance
Bound Analysis and the Halting Problem

For imperative programs:

Halting Problem = termination analysis of loops

→ Bound analysis is a hard problem!

while(n != 0)
  if (n % 2 == 0)
    n = n / 2;
  else
    n = 3n + 1;

Typical programs?

while(n > 0) {
  m := n--;
  while(m > 0 && ?)
    m--;
}

TU Wien

Florian Zuleger
Bound Analysis and the Halting Problem

**Collatz Conjecture**

while(n != 0)
  if (n % 2 == 0)
    n = n / 2;
  else
    n = 3n + 1;

**Real-life Programs**

while(n > 0) {
  m := n--;
  while(m > 0 && ?) 
    m--;
}
Methodological Approach

**Program** → **Abstract Program** → **Bounds**

**Program Abstraction**

**Bound Analysis**

**Desired properties of our abstract program model:**
- simple
- good computational properties
- motivates further theoretical analysis

**Design goals of our analysis:**
- no refinement loop
- fail fast
- captures most common loop patterns
Minimal Requirements for Abstract Program Model?

We need to model:
- counter variables
  - increments/decrements
  - resets
- finite control

```c
foo(uint n)
x = n;
y = n;
while(x > 0)
{
    x--;
    y = y + 2;
}
z = y;
while(z > 0)
    z--;
```
\[ u' \leq v + k \ (k \text{ in } \mathbb{Z}). \]

Variables take values over \( \mathbb{N} \).

\[
\begin{align*}
\text{foo}(\text{uint } n) \\
x &= n; \\
y &= n; \\
\text{while}(x > 0) \\
& \quad \{ x--; \\
& \quad \quad y = y + 2; \\
& \quad \} \\
z &= y; \\
\text{while}(z > 0) \\
& \quad z--; \\
\end{align*}
\]
Difference Constraint Programs (DCPs)

• Introduced by Ben-Amram (2008)
• Termination is undecidable in general, but decidable for deterministic DCPs
  (deterministic = at most one constraint $u' \leq v + k$ for every variable $u$, this is a natural subclass: every variable assignement is abstracted to one constraint)

• we show: DCPs can model interesting bound analysis problems
Invariant Analysis Problems

Variable Bound(y): 3n

Variable Bound(y): n + 2\max\{m1,m2\}

Variable Bound(y): 3n + 2n^2
Bound Analysis Algorithm: Intuition

**Local Bound**
- Transition Bound (t1): 3n
- Local Bound (t1): 3n
- Transition Bound (t2): 3n
- Local Bound (t2): z

**Variable Bound**
- Variable Bound (y): 3n

**Mutual recursion**
- Between Variable Bound and Transition Bound

```
x' \leq x - 1
y' \leq y + 2
z' \leq z - 1
```

```
y \text{ modified on two transitions}
```

```
Variable Bound(y) = n \cdot \text{Transition Bound(ta)} + 2 \cdot \text{Transition Bound(t1)}
```

TU Wien

Florian Zuleger
We assume a **local bound** for every transition, which **decreases** when the transition is executed:

- \( \text{LB}(t_1) = x \)
- \( \text{LB}(t_2) = z \)
- \( \text{LB}(ta) = 1 \)
- \( \text{LB}(tb) = 1 \)

We define **increments** and **resets**:

- \( \text{inc}(t_1,y) = 2 \)
- \( \text{reset}(ta,z) = y \)
- \( \text{reset}(tb,x) = n \)
- \( \text{reset}(tb,y) = n \)
Bound Analysis Algorithm

\[ TB(t_2) = \]
\[ = VB(\text{reset}(tb, LB(t_2))) \times TB(tb) = \]
\[ = VB(\text{reset}(tb, z)) \times 1 = \]
\[ = VB(y) = \]
\[ = VB(\text{reset}(ta, y)) \times TB(ta) + \]
\[ \text{inc}(t_1, y) \times TB(t_1) = \]
\[ = n \times 1 + \]
\[ 2 \times VB(\text{reset}(ta, LB(t_1))) \times TB(ta) = \]
\[ = n + 2 \times VB(x) \times 1 = 3n \]
Bound Analysis Algorithm

Let $P$ be a set of transitions, where each transition $t$ of $P$ has local bound $LB(t)$.

For all $t \in P$ we define

$$TB(t) = \sum_{s \in P} inc(s, LB(t)) \times TB(s) + \sum_{s \in P} VB(reset(s, LB(t))) \times TB(s)$$

$$VB(t) = \sum_{s \in P} inc(s, LB(t)) \times TB(s) + \max_{s \in P} VB(reset(s, LB(t))) \times TB(s)$$

Mutual recursion terminates, if there are no cycles. (is the case for reasonable programs).
Invariant Analysis Problems

Alternative Method for Invariant Computation:
- demand-driven
- compositional
- no fixed point computation needed

Complementary technique for invariant computation

Variable Bound(y):
- 3n
- n + 2max{m1,m2}
- 3n + 2n^2

Related Work:
- has also been observed in earlier work on bound analysis
  - SPEED
  - KoAT
  - Loopus 2014
Amoritzed Complexity Analysis

```c
foo(uint n)
x = n;
r = 0;
while(x > 0)
{
    t1: x--;
    r++;
    if(?) {
        p = r;
        while(p > 0)
            t2: p--;
        r = 0;
    }
}
```

- **Local Bound (t1):** x
- **Transition Bound (t1):** n
- **Local Bound (t2):** p
- **Variable Bound (p):** n
- **Transition Bound (t2):** n²
foo(uint n)
x = n;
r = 0;
while(x > 0)
{\textbf{t1}: } x--;
\textbf{r++};
if(?) {
\textbf{p = r};
while(p > 0)
\textbf{t2}:} p--;
\textbf{r = 0};
}

\textbf{We call } r = 0 \textbf{ a context for } p = r.

\textbf{r ist reset after the inner loop} → every increment \textbf{r++} leads to one loop iteration

\textbf{Complexity: 2n}

\textbf{Local Bound(t1): x}
\textbf{Transition Bound(t1): n}
\textbf{Local Bound(t2): p}
\textbf{Variable Bound(p): n}
\textbf{Transition Bound(t2): n}
foo(uint n)
x = n;
r = 0;
while(x > 0)
{
    x--;
    r++;
    if(?)
    {
        p = r;
        while(p > 0)
        {
            p--;
        }
        r = 0;
    }
}

Using contexts our algorithm can compute the linear complexity.

Complexity: 2n
Amortized Complexity in Real Code

Amortization due to

Dependencies between increments and resets

15 Examples found during our experiments

Examples: forsyte.at/software/loopus
Implementation

• **Tool:** „Loopus“ based on LLVM and Z3

• **Evaluation:** CBench („Collective Benchmark“)
  – C programs
  – 1027 files with > 200 kLoc, > 4000 loops
  – 1751 functions

• **Comparison to the tools:**
  – KoAT (TACAS 2014)
  – CoFloCo (APLAS 2014)
  – Loopus 2014 (CAV 2014)

• **First experimental comparison on real world code**
### Experimental Results: Function Complexity

<table>
<thead>
<tr>
<th>Succ</th>
<th>1</th>
<th>$n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$n^{&gt;3}$</th>
<th>$2^n$</th>
<th>Time</th>
<th>TimeOut</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loopus 15</td>
<td>806</td>
<td>205</td>
<td>489</td>
<td>97</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>15m</td>
</tr>
<tr>
<td>Loopus 14</td>
<td>431</td>
<td>200</td>
<td>188</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>40m</td>
</tr>
<tr>
<td>KoAT</td>
<td>430</td>
<td>253</td>
<td>138</td>
<td>35</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>5.6h</td>
</tr>
<tr>
<td>CoFloCo</td>
<td>386</td>
<td>200</td>
<td>148</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.7h</td>
</tr>
</tbody>
</table>

**Loopus 15**: - More Complexity Results  
- In Shorter Time

More details: forsyte.at/software/loopus
Summary

Contributions to bound/complexity analysis:

• notions of increment/decrement and reset
• first algorithm based on DCPs
• we demonstrate the scalability and applicability of our algorithm

• DCPs are an interesting model for further research