ABC: Algebraic Bound Computation for Loops

Régis Blanc  Thomas A. Henzinger  Thibaud Hottelier  Laura Kovács
EPFL  IST Austria  UC Berkeley  TU Vienna
What Is ABC?

- **Automatically** computes an algebraic relation among iteration variables of a loop
What Is ABC?

- **Automatically** computes an algebraic relation among iteration variables of a loop
- Can then derive *iteration bound* of the loop
What Is ABC?

- **Automatically** computes an algebraic relation among iteration variables of a loop
- Can then derive **iteration bound** of the loop
- Works for **any** level of nested loops
What Is ABC?

- **Automatically** computes an algebraic relation among iteration variables of a loop
- Can then derive **iteration bound** of the loop
- Works for **any** level of nested loops
- Only works on **specific** shape of loops
Some real-time systems need to prove an upper bound time of their computation

Can be used to prove termination
Application: Matrix Linearization

for $i = 1$ to $n$ do
  for $j = 1$ to $n$ do
    $M[i,j] = i + j$
  end for
end for

$\rightarrow$

for $i = 1$ to $n$ do
  for $j = 1$ to $n$ do
    $A[(i - 1)n + j] = i + j$
  end for
end for
An Example

\[
\begin{align*}
z &= 1 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
&\quad \text{for } j = 1 \text{ to } n \text{ do} \\
&\quad \quad z = z + 1 \\
&\quad \text{end for} \\
\text{end for}
\end{align*}
\]
An Example

\[ z = 1 \]
\[
\text{for } i = 1 \text{ to } n \text{ do }
\]
\[
\text{for } j = 1 \text{ to } n \text{ do }
\]
\[ z = z + 1 \]
\[
\text{end for}
\]
\[
\text{end for}
\]

▶ Z-relation:

\[ z = (i - 1)n + j \]
An Example

\[ z = 1 \]
\[ \text{for } i = 1 \text{ to } n \text{ do} \]
\[ \quad \text{for } j = 1 \text{ to } n \text{ do} \]
\[ \quad z = z + 1 \]
\[ \quad \text{end for} \]
\[ \text{end for} \]

- Z-relation:
  \[ z = (i - 1)n + j \]

- Iteration bound:
  \[ n^2 \]
Another Example

\[
z = 1 \\
\text{for } i = 0; i \leq n; i = i + 1 \text{ do} \\
\quad \text{for } j = 0; j \leq m; j = j + 2 \text{ do} \\
\quad \quad z = z + 1 \\
\quad \text{end for} \\
\text{end for}
\]
Another Example

\[ z = 1 \]
\[ \text{for } i = 0; i \leq n; i = i + 1 \text{ do} \]
\[ \quad \text{for } j = 0; j \leq m; j = j + 2 \text{ do} \]
\[ \quad \quad z = z + 1 \]
\[ \quad \text{end for} \]
\[ \text{end for} \]

▶ Z-relation:

\[ z = 1 + \left\lfloor \frac{j}{2} \right\rfloor + i \left( \left\lfloor \frac{m}{2} \right\rfloor + 1 \right) \]
Another Example

\[ z = 1 \]
\[ \text{for } i = 0; \ i \leq n; \ i = i + 1 \ \text{do} \]
\[ \text{for } j = 0; \ j \leq m; \ j = j + 2 \ \text{do} \]
\[ z = z + 1 \]
\[ \text{end for} \]
\[ \text{end for} \]

- Z-relation:
  \[ z = 1 + \left\lfloor \frac{j}{2} \right\rfloor + i \left( \left\lfloor \frac{m}{2} \right\rfloor + 1 \right) \]

- Iteration bound:
  \[ 1 + (1 + n) \left\lfloor \frac{m}{2} \right\rfloor + n \]
ABC Structure

ABC relies on three modules:

- Bound Computer
- Loop Converter
- Symbolic Solver
ABC Overall Workflow
Bound Computer

- Works on a special class of loop: ABC-loops
- Computes a polynomial relation over loop variables: z-relation
- Core part of ABC
Bound Computer On An Example

\[ z = 1 \]
\[ \text{for } i = 1 \text{ to } n \text{ do } \]
\[ \quad \text{for } j = 1 \text{ to } m \text{ do } \]
\[ \quad \quad z = z + 1 \]
\[ \quad \text{end for} \]
\[ \text{end for}\]

\[ z = 1 \]
\[ \text{for } i = 1 \text{ to } n \text{ do } \]
\[ \quad z = z + \sum_{k=1}^{m} 1 \]
\[ \text{end for}\]
Bound Computer On An Example

Solving the sum:

\[
z = 1 \\
\text{for } i = 1 \text{ to } n \text{ do } \\
\quad \text{for } j = 1 \text{ to } m \text{ do } \\
\quad \quad z = z + 1 \\
\quad \text{end for} \\
\text{end for}
\]

\[
\rightarrow z = 1 \\
\text{for } i = 1 \text{ to } n \text{ do } \\
\quad z = z + m \\
\text{end for}
\]

Recurrence relation for the value \(z_i\) of \(z\) at iteration \(i\):

\[
z_i = z_{i-1} + m
\]

With \(z_1 = 1\), we get:

\[
z_i = 1 + \sum_{k=1}^{i-1} m = 1 + (i - 1)m
\]
Apply recursively the method on the loop:

\[
\begin{align*}
  z &= 1 \\
  \text{for } &i = 1 \text{ to } n \text{ do} \\
  &\quad \text{for } j = 1 \text{ to } m \text{ do} \\
  &\quad \quad z = z + 1 \\
  &\quad \text{end for} \\
  &\text{end for}
\end{align*}
\]

\[
\begin{align*}
  z &= 1 + (i - 1)m \\
  \text{for } &j = 1 \text{ to } m \text{ do} \\
  &\quad z = z + 1 \\
  &\text{end for}
\end{align*}
\]

Recurrence relation of \(z_j\):

\[
\begin{align*}
  z_j &= z_{j-1} + 1
\end{align*}
\]

With \(z_1 = 1 + (i - 1)m\), we get:

\[
\begin{align*}
  z_j &= 1 + (i - 1)m + \sum_{k=1}^{j-1} 1 = (i - 1)m + j
\end{align*}
\]
Bound Computer On An Example

\[
z = 1 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad \text{for } j = 1 \text{ to } m \text{ do} \\
\quad \quad z = z + 1 \\
\quad \text{end for} \\
\text{end for}
\]

Thus we obtain the \textit{z-relation}: 

\[
z = (i - 1)m + j
\]

Substituting \(i\) by \(n\) and \(j\) by \(m\), we get the iteration bound: 

\[
nm
\]
Why A Loop Converter?

▶ The preceding algorithm only works on the ABC-loops.
Why A Loop Converter?

- The preceding algorithm only works on the ABC-loops.
- Transforms, whenever possible, a loop into its equivalent ABC-loop.
Why A Loop Converter?

- The preceding algorithm only works on the ABC-loops.
- Transforms, whenever possible, a loop into its equivalent ABC-loop.
  - Loop that starts from an arbitrary expression
Why A Loop Converter?

- The preceding algorithm only works on the ABC-loops.
- Transforms, whenever possible, a loop into its equivalent ABC-loop.
  - Loop that starts from an arbitrary expression
  - Loop with each of its iteration variables incremented by one
Yet Another Example

\[z = 1\]

\[\text{for } i = 0; \ i \leq n; \ i = i + 1 \text{ do} \]
\[\quad \text{for } j = 0; \ j \leq m; \ j = j + 2 \text{ do} \]
\[\quad \quad z = z + 1 \]
\[\quad \text{end for} \]
\[\text{end for} \]
Yet Another Example

\[ z = 1 \]

\[ \text{for } i = 0; \ i \leq n; \ i = i + 1 \ \text{do} \]
\[ \quad \text{for } j' = 0; \ j' \leq \left\lfloor \frac{m}{2} \right\rfloor; \ j' = j' + 1 \ \text{do} \]
\[ \quad \quad z = z + 1 \]
\[ \quad \text{end for} \]
\[ \text{end for} \]

With:

\[ j = 2j' \]
Yet Another Example

\[ z = 1 \]
\[ \text{for } i' = 1; i' \leq n + 1; i' = i' + 1 \text{ do} \]
\[ \quad \text{for } j'' = 1; j'' \leq \left\lfloor \frac{m}{2} \right\rfloor + 1; j'' = j'' + 1 \text{ do} \]
\[ \\
\quad \quad z = z + 1 \]
\[ \quad \text{end for} \]
\[ \text{end for} \]

With:

\[ j = 2j' \quad \land \quad j' = j'' - 1 \quad \land \quad i = i' - 1 \]
Symbolic Solver

- We built our own symbolic solver:
  - Simplifies symbolic expressions
  - Derives closed forms for symbolic sums
- We rely on a simplified version of the Gosper Algorithm
The ABC-Loops

\[
\begin{aligned}
\text{for } (i_1 = 1; i_1 \leq f_1; i_1 = i_1 + 1) & \text{ do} \\
\text{for } (i_2 = 1; i_2 \leq f_2(i_1); i_2 = i_2 + 1) & \text{ do} \\
\ldots & \\
\text{for } (i_n = 1; i_n \leq f_n(i_1, \ldots, i_{n-1}); i_n = i_n + 1) & \text{ do} \\
\text{skip} \\
\text{end for} \\
\ldots & \\
\text{end for} & \\
\end{aligned}
\]

Where
\[
\begin{aligned}
i_1, i_2, \ldots, i_n & \text{ are iteration variables} \\
f_1, f_2, \ldots, f_n & \text{ are polynomials functions with symbolic constants}
\end{aligned}
\]
Bound Computer Algorithm

Require: ABC-loop $F$, initial value $z_0$ of $z$
Ensure: $z$-relation $z_{rel}$

\[
\begin{align*}
inner & := \text{loop\_body}(F) \\
incr & := \text{z\_reduce\_loop}(inner) \\
\langle o\text{var},\ o\text{bound} \rangle & := \\
\langle \text{outer\_iteration\_variable}(F),\ \text{outer\_iteration\_upperbound}(F) \rangle \\
n\text{var} & := \text{fresh\_variable}() \\
z_i & := z_0 + \text{solve\_sum}(n\text{var}, 1, o\text{var} - 1, incr[\ o\text{var} \mapsto \ n\text{var}]) \\
\text{if } \text{isloop}(inner) \text{ then} \\
\quad z_{rel} & := z = \text{Bound Computer}(inner, z_i) \\
\text{else} \\
\quad z_{rel} & := z = z_i \\
\text{end if}
\end{align*}
\]
Loop Converter Algorithm

Require: For-loop $F$ and $conversion\_list = {}$

Ensure: ABC-loop $F'$ and $conversion\_list$

\[
\langle ovartext, oincretext \rangle := \langle outer\_iteration\_variable(F), outer\_iteration\_increment(F) \rangle
\]

\[
\langle olboundtext, ouboundtext \rangle := \langle outer\_iteration\_lowerbound(F),
outer\_iteration\_upperbound(F) \rangle
\]

$nvar := fresh\_variable()$

$F_0 := loop\_body(F)[ovar \mapsto oincretext \cdot (nvar + olboundtext - 1)]$

$conversion\_list := conversion\_list \cup \{ovar = oincretext \cdot (nvar + olboundtext - 1)\}$

if isloop($F_0$) then

\[
F' := for\_loop(nvar, 1, \left\lfloor \frac{ouboundtext - olboundtext}{oincretext} \right\rfloor + 1, 1, Loop\_Converter(F_0))
\]

else

\[
F' := for\_loop(nvar, 1, \left\lfloor \frac{ouboundtext - olboundtext}{oincretext} \right\rfloor + 1, 1, F_0)
\]

end if
More General Loops

\[
\begin{align*}
\text{for } & (i_1 = g_1; i_1 \diamond f_1; i_1 = i_1 + r_1) \text{ do } \\
\text{for } & (i_2 = g_2(i_1); i_2 \diamond f_2(i_1); i_2 = i_2 + r_2) \text{ do } \\
\& \quad \ldots \\
\text{for } & (i_n = g_n(i_1, \ldots, i_{n-1}); i_n \diamond f_n(i_1, \ldots, i_{n-1}); i_n = i_n + r_n) \text{ do } \\
\& \quad \quad \text{skip } \\
\& \quad \text{end for } \\
\& \quad \ldots \\
\text{end for } \\
\text{end for }
\end{align*}
\]

Where

\(i_1, \ldots, i_n\) are iteration variables

\(r_1, \ldots, r_n\) are symbolics constants

\(f_1, \ldots, f_n, g_1, \ldots, g_n\) are polynomials functions with symbolic constants

\(\diamond \in \{<, \leq, >, \geq\}\)
Symbolic Solver Capabilities

The handled sums are of the following form:

$$\sum_{x=e_1}^{e_2} c_1 \cdot n_1^x \cdot x^{d_1} + \ldots + c_r \cdot n_r^x \cdot x^{d_r}$$

where $e_1, e_2$ are integer valued symbolic constants, $n_i, d_i \in \mathbb{N}$ and $c_i \in \mathbb{Q}$.
We extracted 90 loops from the JAMA package
JAMA Package

- We extracted 90 loops from the JAMA package
- ABC derived the \textit{z-relation} for 87 of them
- ABC was able to compute complexity for \textit{all of them}
We extracted 90 loops from the JAMA package. ABC derived the z-relation for 87 of them. ABC was able to compute complexity for all of them. All in less than one second.
## Some Results

<table>
<thead>
<tr>
<th>Loop</th>
<th>z-relation</th>
<th>Iteration bound</th>
<th>Time [s]</th>
</tr>
</thead>
</table>
| **for** \((i = 1; i \leq n; i = i + 1)\)  
**for** \((j = 1; j \leq i; j = j + 1)\)  
skip  
end do  
end do | \(z = \frac{i^2 - i}{2} + j\)                                                      | \(\frac{n^2 + n}{2}\)                           | 0.203    |
| **for** \((i = 1; i \leq m; i = i + 1)\)  
**for** \((j = 1; j \leq i; j = j + 1)\)  
**for** \((k = i + 1; k \leq m; k = k + 1)\)  
**for** \((l = 1; l \leq k; l = l + 1)\)  
skip  
end do  
end do  
end do  
end do | \(z = \frac{i^2 m^2 - im^2 + i^2 m - im}{4} + \frac{i^2 - i^4}{8} + \frac{i^3 - i}{12} + \frac{jm + jm^2 + k^2}{2} - \frac{m^2 + ji^2 + ji + m + k}{2} + 1\) | \(
\frac{3m^4 + 2m^3 - 3m^2 - 2m}{24}\)                           | 1.281    |
| **for** \((i = 0; i \leq \left(\frac{n^2 n}{2} - 1\right); i = i + 1)\)  
**for** \((j = 0; j \leq n - 1; j = j + 1)\)  
**for** \((k = 0; k \leq j - 1; k = k + 1)\)  
skip  
end do  
end do  
end do | \(z = 1 + k + \frac{in^2 - in + j^2 - j}{2}\)                              | \(
\frac{n^5 - n^4}{4}\)                           | 0.234    |
### Some Other Results

<table>
<thead>
<tr>
<th>Loop</th>
<th>z-relation</th>
<th>Iteration bound</th>
<th>Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>for ($i = n; i \geq 1; i = i - 1$) for ($j = m; j \geq 1; j = j - 1$) skip end do</td>
<td>$z = 1 - j + (n - i + 1)m$</td>
<td>$nm$</td>
<td>0.187</td>
</tr>
<tr>
<td>for ($i = a; i \leq b; i = i + 1$) for ($j = c; j \leq d; j = j + 1$) for ($k = i - j; k \leq i + j; k = k + 1$) skip end do</td>
<td>$z =$</td>
<td>$\left(c^2 - (d+1)^2\right)(a-b-1)$</td>
<td>0.328</td>
</tr>
<tr>
<td>for ($i = n; i \geq 1; i = i - 1$) for ($j = 1; j \leq m; j = j + 1$) for ($k = i; k \leq i + j; k = k + 1$) for ($l = 1; l \leq k; l = l + 1$) skip end do</td>
<td>$z =$</td>
<td>$2m^2 + 3mn + 9m + 9n + 13$ $mn$</td>
<td>0.625</td>
</tr>
</tbody>
</table>
Related Work

- **User-defined invariant templates** [Seidl04]
  - invariants: constraint solving over template coefficients;

- **User-defined atomic predicates and loop patterns** [Gulwani10]
  - bounds: control-flow refinement and abstract interpretation;

- **Recurrence Solving** [van Egelen00, Albert08, Valigator08]
  - bounds: unfolding loops with simple but non-deterministic recurrences;
  - bounds: pattern matching simple class of recurrences;
  - bounds: quantifier elimination for unnested loops and non-initializing assignments;

- **WCET** [aiT04, TUBound09]
  - bounds: interval-based abstract interpretation with unrollings of simple-loops;
  - bounds: solving constraints over variables from linear loop tests.
Conclusion And Future Work

- ABC automatically computes algebraic loop bounds
- ABC is available at http://mtc.epfl.ch/software-tools/ABC
Conclusion And Future Work

▶ ABC automatically computes algebraic loop bounds

▶ ABC is available at http://mtc.epfl.ch/software-tools/ABC

▶ Extend ABC to handle more complex loops and symbolic sums