Handling non-linear operations in the value analysis of \textsc{Costa}

Diego Alonso, Puri Arenas, Samir Genaim

Departamento de Sistemas Informáticos y Computación (DSIC) Complutense University of Madrid (UCM)

March 27	extsuperscript{th} 2011
Costa in a nutshell

Costa is a COSSt and Termination Analyzer that
- analyzes a Java Bytecode (JBC) program
- with a cost model (termination, instructions, heap)
- and computes bounds on its execution cost
Example: logarithm with a fixed base $b = 2$

```c
// Pre: x > 0
int log2(int x) {
    int l = 0;
    int y = 1;
    while (x > y) {
        y = y * 2;
        l = l + 1;
    }
    return l;
}
```

- $y$ is initialized as $y = 1$
- each iteration duplicates $y$ until it reaches $x$
- therefore, an execution of $\log_2(x)$ terminates after $\log_2 x$ iterations
- **Costa** infers a cost in $O(\log(x))$
Example: logarithm with a **parametric** base $b$

Example

```c
// Pre: x>0, b>1
int logB(int b, int x) {
    int l = 0;
    int y = 1;
    while (x > y) {
        y = y * b;
        l = l + 1;
    }
    return l;
}
```

- $y$ is initialized as $y = 1$.
- each iteration multiplies $y$ until it reaches $x$
- therefore, an execution of $\log_B(b, x)$ terminates after approx $\log_b x$ iterations

**Costa** fails to prove termination or complexity
**Costa** first “compiles” the bytecode to abstract rules in which

- data is represented as size values
- each operation is replaced with its cost and
- its effect is modeled with **linear constraints**
Analysis example: logarithm with a **fixed** base $b = 2$

Example

```c
// Pre: x>0
int log2(int x){
    int l = 0;
    int y = 1;
    while (x>y){
        y = y * 2;
        l = l + 1 ;
    }
    return l;
}
```

```
log2(⟨x⟩, ⟨l⟩) ←
{ l0 = 0} ,
{ y0 = 1} ,
log2_w(⟨x, l0, y0⟩, ⟨l1⟩),
{l = l1} .
log2_w(⟨x, l1, y1⟩, ⟨l3⟩) ←
{x > y1} ,
{ y2 = y1 * 2} ,
{l2 = l1 + 1} ,
log2_w(⟨x, l2, y2⟩, ⟨l3⟩).
log2_w(⟨x, l, y1⟩, ⟨l⟩) ←
{x ≤ y1} .
```
The resolution phase obtains the desired results from the Cost-Size-Rules:

- **Entries** model the program’s starting state
- A **postcondition** models the size effect of a call
- A **transition** describes how variables change from the rule’s header to a recursive call
Resolution example: logarithm with a fixed base $b = 2$

Example

$$\log_2(\langle x \rangle, \langle l \rangle) \leftarrow$$
$$\{ l_0 = 0 \},$$
$$\{ y_0 = 1 \},$$
$$\log_2_w(\langle x, l_0, y_0 \rangle, \langle l_1 \rangle),$$
$$\{ l = l_1 \}.$$

$$\log_2_w(\langle x, l_1, y_1 \rangle, \langle l_3 \rangle) \leftarrow$$
$$\{ x > y_1 \},$$
$$\{ y_2 = y_1 \ast 2 \},$$
$$\{ l_2 = l_1 + 1 \},$$
$$\log_2_w(\langle x, l_2, y_2 \rangle, \langle l_3 \rangle).$$

$$\log_2_w(\langle x, l, y_1 \rangle, \langle l \rangle) \leftarrow$$
$$\{ x \leq y_1 \}.$$

- Preconditions:
  $$\log_2(\langle x_0 \rangle) \triangleright \{ x_0 \geq 0 \}$$
  $$\log_2_w(\langle x, l_1, y_1 \rangle) \triangleright$$
  $$\{ x > 0, l_1 \geq 0, y_1 > 0 \}$$

- Loop transition of $\log_2_w$:
  $$\langle x, l_1, y_1 \rangle \rightarrow \langle x, l_2, y_2 \rangle \triangleright$$
  $$\{ x > 0, l_1 \geq 0, y_1 > 0 \} \cap$$
  $$\{ x > y_1, y_2 = y_1 \ast 2, l_2 = l_1 + 1 \}$$

- This transition can be proven to have a logarithmic order
Analysis example: logarithm with a \textit{parametric} base $b$

Example

```
// Pre: x>0,b>1
int logB (int b, int x){
    int l = 0;
    int y = 1;
    while (x>y){
        y = y * b;
        l = l + 1;
    }
    return l;
}
```

$logB((b, x), (l)) \leftarrow$
\begin{align*}
    \{& l_0 = 0 \} , \\
    \{& y_0 = 1 \} , \\
    logB_w((b, x, l_0, y_0), (l)) .
\end{align*}

$logB_w((b, x, l_1, y_1), (l_2)) \leftarrow$
\begin{align*}
    \{& x > y_1 \} , \\
    \{& y_2 = \_ \} , \\
    \{& l_2 = l_1 + 1 \} , \\
    logB_w((b, x, l_2, y_2), (l_2)) .
\end{align*}

$logB_w((b, x, l, y_1), (l)) \leftarrow$
\begin{align*}
    \{& x \leq y_1 \} .
\end{align*}
Resolution example: logarithm with a parametric base $b$

Example

$$\log_B(\langle b, x \rangle, \langle l \rangle) \leftarrow$$

$\{l_0 = 0\},$

$\{y_0 = 1\},$

$$\log_{B_w}(\langle b, x, l_0, y_0 \rangle, \langle l \rangle).$$

$$\log_{B_w}(\langle b, x, l_1, y_1 \rangle, \langle l_2 \rangle) \leftarrow$$

$\{x > y_1\},$

$\{y_2 = -\},$

$\{l_2 = l_1 + 1\},$

$$\log_{B_w}(\langle b, x, l_2, y_2 \rangle, \langle l_2 \rangle).$$

$$\log_{B_w}(\langle b, x, l, y_1 \rangle, \langle l \rangle) \leftarrow$$

$\{x \leq y_1\}.$

- **Preconditions**
  $$\log_B(\langle b, x \rangle) \blacktriangleright \{b > 1, x \geq 0\}$$
  $$\log_{B_w}(\langle b, x, l_1, y_1 \rangle) \blacktriangleright$$
  $$\{b > 1, x \geq 1, l_1 \geq 0, y_1 = -\}$$

- **Loop transition of $\log_{2w}$:**
  $$\langle b, x, l_1, y_1 \rangle \rightarrow \langle b, x, l_2, y_2 \rangle \blacktriangleright$$
  $$\{x > y_1, y_2 = -, l_2 = l_1 + 1\}$$

- **This loop is non-terminating**
Linear or non-linear operations

A linear operation like $z = x + y$ can be modeled with a linear constraint.

A nonlinear operation like $z = x \cdot y$ can only be modeled as $\top$. 
Solution: disjunctive abstraction of $z = x \times y$

Example ($z = x \times y$)

$\varphi_1 \equiv \{x = 0\}, \{z = 0\}$
$\varphi_2 \equiv \{y = 0\}, \{z = 0\}$
$\varphi_3 \equiv \{x = 1\}, \{z = y\}$
$\varphi_4 \equiv \{y = 1\}, \{z = x\}$
$\varphi_5 \equiv \{x = -1\}, \{z = -y\}$
$\varphi_6 \equiv \{y = -1\}, \{z = -x\}$
$\varphi_7 \equiv \{x \geq 2, y \geq 2\}$
$\quad \{z \geq 2x, z \geq 2y\}$
$\varphi_8 \equiv \{x \geq 2, y \leq -2\}$
$\quad \{z \leq -2x, z \leq 2y\}$
$\varphi_9 \equiv \{x \leq -2, y \geq 2\}$
$\quad \{z \leq 2x, z \leq -2y\}$
$\varphi_{10} \equiv \{x \leq -2, y \leq -2\}$
$\quad \{z \geq -2x, z \geq -2y\}$
The solution: disjunctive abstractions

- Nonlinear operations can’t be modeled with linear constraints because no linear constraint holds for all input values (input space).
- But a constraint can hold for the inputs in a subset of the space.
- We abstract a non-linear operation $\star$ to a finite disjunction $\phi_1 \lor \phi_2 \lor \cdots \lor \phi_n$. 

\[
\begin{array}{c}
\phi_1 \\
\phi_2 \\
\phi_3 \\
\phi_4 \\
\phi_5 \\
\end{array}
\]
(a) We could employ a value analysis over a disjunctive domain, but using a disjunctive abstract domain doesn’t scale.

Example: When the value analysis reaches the operation $z = x \times y$ and it knows that $x \geq 1$ and $y \geq 2$, we want it to use only the satisfiable cases

$$\phi_3 \equiv \{ x = 1 \} \{ z = y \}$$

$$\phi_7 \equiv \{ x \geq 2, y \geq 2 \} \{ z \geq 2x, z \geq 2y \}$$

and ignore the (unsatisfiable) rest for computing the postcondition

$$\{ z = y \} \cup \{ z \geq 2x, z \geq 2y \} = \{ z \geq 2x, z \geq y \}$$
(a) We could employ a value analysis over a disjunctive domain but using a disjunctive abstract domain doesn’t scale

(b) Instead, we encode those disjunctions into the abstract program and use linear constraints in the value analysis

Example

When the value analysis reaches the operation $z = x \times y$ and it knows that $x \geq 1$ and $y \geq 2$, we want it to use only the satisfiable cases

$$\varphi_3 \equiv \{x = 1\} \quad \{z = y\}$$
$$\varphi_7 \equiv \{x \geq 2, y \geq 2\} \quad \{z \geq 2x, z \geq 2y\}$$

and ignore the (unsatisfiable) rest for computing the postcondition

$$\{z = y\} \sqcup \{z \geq 2x, z \geq 2y\} = \{z \geq 2x, z \geq y\}$$
Using disjunctive information

- Replace each code $b \equiv z = x \star y$ by a call to $\text{op}_{\star b}^\ast(\langle x, y \rangle, \langle z \rangle)$.
- $\text{op}_{\star b}^\ast(\langle x, y \rangle, \langle z \rangle)$ is defined with one rule per case.
Using disjunctive information

- Replace each code $b \equiv z = x \star y$ by a call to $op_{\star b}(\langle x, y \rangle, \langle z \rangle)$.
- $op_{\star b}(\langle x, y \rangle, \langle z \rangle)$ is defined with one rule per case.
- The value analysis computes the precondition $pre(op^b_{\star})$ and the postcondition $post(op^b_{\star})$ as for any other predicate.
Analyzing logB by abstract program transformation

Example (Program transformation of logB)

```c
// Pre: x > 0, b > 1
int logB(int b, int x){
    int l = 0;
    int y = 1;
    while (x > y){
        y = y * b;
        l = l + 1 ;
    }
    return l;
}
```

```
logB(⟨b, x⟩, ⟨l⟩) ←
   {l₀ = 0},
   {y₀ = 1},
   logBₜ(⟨b, x, l₀, y₀⟩, ⟨l⟩).

logBₜ(⟨b, x, l₁, y₁⟩, ⟨l₂⟩) ←
   {x > y₁},
   op*(⟨y₁, b⟩, ⟨y₂⟩),
   {l₂ = l₁ + 1},
   logBₜ(⟨b, x, l₂, y₂⟩, ⟨l₂⟩).

logBₜ(⟨b, x, l, y₁⟩, ⟨l⟩) ←
   {x ≤ y₁}.
```
The solution

Example (Value analysis of $op_*$ )

The precondition of $op_*(\langle y_1, b \rangle)$ is $op_*(\langle y_1, b \rangle) \blacktriangleleft \{ y_1 \geq 1, b \geq 2 \}$

\[
\begin{align*}
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ y_1 = 0 \}, \quad \ldots \quad \leftarrow \{ y_2 = b \}. \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ b = 0 \}, \quad \ldots \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ b = 1 \}, \quad \ldots \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ y_1 = 1 \}, \quad \leftarrow \{ y_2 = b \}. \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ b \geq 2, y_1 \geq 2 \}, \quad \leftarrow \{ y_2 \geq 2y_1, y_2 \geq 2b \}. \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ y_1 \geq 2, b \leq -2 \}, \quad \ldots \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ y_1 \leq -2, b \geq 2 \}, \quad \ldots \\
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \leftarrow \{ y_1 \leq -2, b \leq -2 \}, \quad \ldots
\end{align*}
\]

The value analysis computes the postcondition

\[
\begin{align*}
&\quad op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \blacktriangleleft \{ y_2 = b \} \sqcup \{ y_2 \geq 2y_1, y_2 \geq 2b \} = \{ y_2 \geq 2y_1, y_2 \geq b \}
\end{align*}
\]
Example (Bounding the iterations of $\log B$)

- The precondition of $op_*(\langle y_1, b \rangle)$ is $op_*(\langle y_1, b \rangle) \blacktriangleright \{y_1 \geq 1, b \geq 2\}$
- The postcondition is $op_*(\langle y_1, b \rangle, \langle y_2 \rangle) \blacktriangleright \{y_2 \geq 2y_1, y_2 \geq b\}$
- Using these, we can infer the transition

$$\langle b, x, l_1, y_1 \rangle \rightarrow \langle b, x, l_2, y_2 \rangle \blacktriangleright \{y_1 \geq 1, b \geq 2\} \sqcap \{x > y_1, l_2 = l_1 + 1\} \sqcap \{y_2 \geq 2y_1, y_2 \geq b\}$$

- This transition can be proven to have $O(\log(x - y))$ iterations
- **Costa** can now infer that $\log B$ has a logarithmic cost
Conclusions

- Nonlinear operations are difficult to analyze
- We propose to use a program transformation for handling nonlinear operations like $z = x \times y$ in static analysis
  - This technique **increases precision** by producing more accurate abstract information
  - and it’s **scalable** because it still uses linear constraints
- This solution is also applicable to other operations: integer quotient (\(/\)) and remainder (\(\%\)), and bitwise operations ($\&$, $\mid$, $<<$, $>>$, $>>>$)