Handling non-linear operations in the value analysis of COSTA

Diego Alonso, Puri Arenas, Samir Genaim

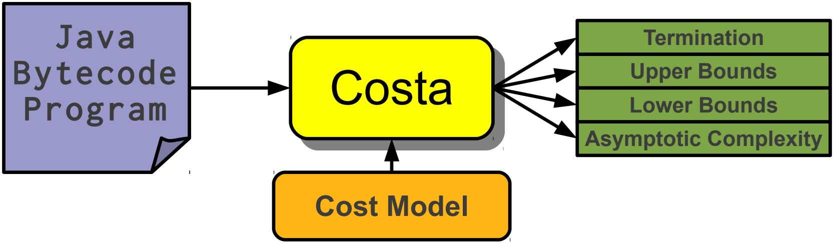
Departamento de Sistemas Informáticos y Computación (DSIC)
Complutense University of Madrid (UCM)

March 27^{th} 2011

COSTA in a nutshell

COSTA is a COSt 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 $b = 2$

Example

```
// 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 `log2(x)` terminates after $log_2 x$ iterations
- COSTA **infers a cost in** $\mathcal{O}(log(x))$

Example: logarithm with a **parametric** base b

Example

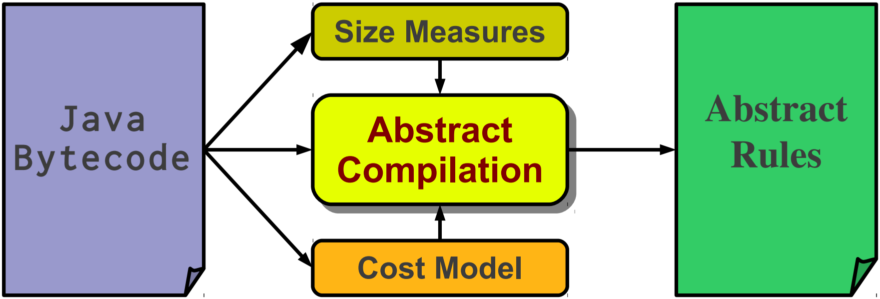
```
// Pre: x>0,b>1
int logB(int b,int x){
    int l = 0;
    int y = 1;
    while(x>y){
        y = y * b;
        l = l + 1 ;
    }
    return l;
}
```

- y is initialized as $y = 1$.
- each iteration multiplies y until it reaches x
- therefore, an execution of `logB(b, x)` terminates after approx $log_b x$ iterations
- COSTA **fails to prove termination or complexity**

Analysis overview

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

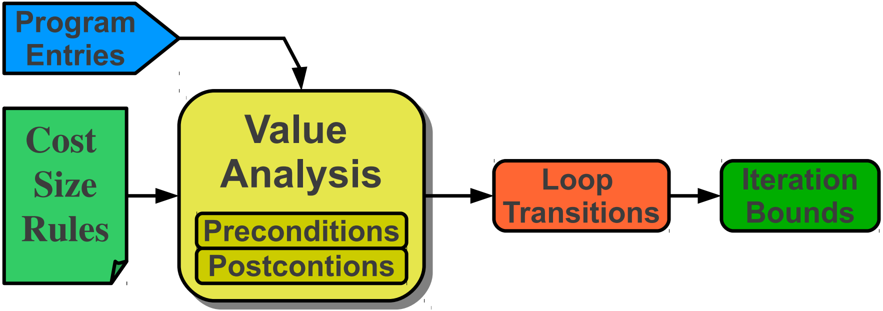
```
// Pre: x>0
int log2(int x){
    int l = 0;
    int y = 1;
    while(x>y){
        y = y * 2;
        l = l + 1 ;
    }
    return l;
}
```

$$
\begin{array}{l}
log2(\langle x\rangle, \langle l\rangle) \leftarrow \\
\quad \{l_0 = 0\}, \\
\quad \{y_0 = 1\}, \\
\quad log2_w(\langle x, l_0, y_0\rangle, \langle l_1\rangle), \\
\quad \{l = l_1\}. \\
log2_w(\langle x, l_1, y_1\rangle, \langle l_3\rangle) \leftarrow \\
\quad \{x > y_1\}, \\
\quad \{\mathbf{y_2 = y_1 * 2}\}, \\
\quad \{l_2 = l_1 + 1\}, \\
\quad log2_w(\langle x, l_2, y_2\rangle, \langle l_3\rangle). \\
log2_w(\langle x, l, y_1\rangle, \langle l\rangle) \leftarrow \\
\quad \{x \leq y_1\}.
\end{array}
$$

Resolution overview

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

$$
\begin{array}{l}
log2(\langle x\rangle, \langle l\rangle) \leftarrow \\
\quad \{l_0 = 0\}, \\
\quad \{y_0 = 1\}, \\
\quad log2_w(\langle x, l_0, y_0\rangle, \langle l_1\rangle), \\
\quad \{l = l_1\}. \\
log2_w(\langle x, l_1, y_1\rangle, \langle l_3\rangle) \leftarrow \\
\quad \{x > y_1\}, \\
\quad \{\mathbf{y_2 = y_1 * 2}\}, \\
\quad \{l_2 = l_1 + 1\}, \\
\quad log2_w(\langle x, l_2, y_2\rangle, \langle l_3\rangle). \\
log2_w(\langle x, l, y_1\rangle, \langle l\rangle) \leftarrow \\
\quad \{x \leq y_1\}.
\end{array}
$$

- Preconditions:

$$
\begin{array}{l}
log2(\langle x_0\rangle) \blacktriangleleft \{x_0 \geq 0\} \\
log2_w(\langle x, l_1, y_1\rangle) \blacktriangleleft \\
\quad \{x > 0, l_1 \geq 0, y_1 > 0\}
\end{array}
$$

- Loop transition of $log2_w$:

$$
\begin{array}{l}
\langle x, l_1, y_1\rangle \rightarrow \langle x, l_2, y_2\rangle \blacktriangleleft \\
\quad \{x > 0, l_1 \geq 0, \mathbf{y_1 > 0}\} \sqcap \\
\quad \{\mathbf{x > y_1}, \mathbf{y_2 = y_1 * 2}, l_2 = l_1 + 1\}
\end{array}
$$

- **This transition can be proven to have a logarithmic order**

Analysis example: logarithm with a 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;
}
```

$$
\begin{array}{l}
logB(\langle \mathbf{b}, x\rangle, \langle l\rangle) \leftarrow \\
\quad \{l_0 = 0\}, \\
\quad \{y_0 = 1\}, \\
\quad logB_w(\langle \mathbf{b}, x, l_0, y_0\rangle, \langle l\rangle). \\
logB_w(\langle \mathbf{b}, x, l_1, y_1\rangle, \langle l_2\rangle) \leftarrow \\
\quad \{x > y_1\}, \\
\quad \{\mathbf{y_2} = \_\}, \\
\quad \{l_2 = l_1 + 1\}, \\
\quad logB_w(\langle \mathbf{b}, x, l_2, y_2\rangle, \langle l_2\rangle). \\
logB_w(\langle \mathbf{b}, x, l, y_1\rangle, \langle l\rangle) \leftarrow \\
\quad \{x \leq y_1\}.
\end{array}
$$

Resolution example: logarithm with a **parametric** base b

Example

$$
\begin{aligned}
& logB(\langle \mathbf{b}, x\rangle, \langle l\rangle) \leftarrow \\
& \quad \{l_0 = 0\}, \\
& \quad \{y_0 = 1\}, \\
& \quad logB_w(\langle \mathbf{b}, x, l_0, y_0\rangle, \langle l\rangle). \\
& logB_w(\langle \mathbf{b}, x, l_1, y_1\rangle, \langle l_2\rangle) \leftarrow \\
& \quad \{x > y_1\}, \\
& \quad \{\mathbf{y_2} = \_\}, \\
& \quad \{l_2 = l_1 + 1\}, \\
& \quad logB_w(\langle \mathbf{b}, x, l_2, y_2\rangle, \langle l_2\rangle). \\
& logB_w(\langle \mathbf{b}, x, l, y_1\rangle, \langle l\rangle) \leftarrow \\
& \quad \{x \le y_1\}.
\end{aligned}
$$

- Preconditions

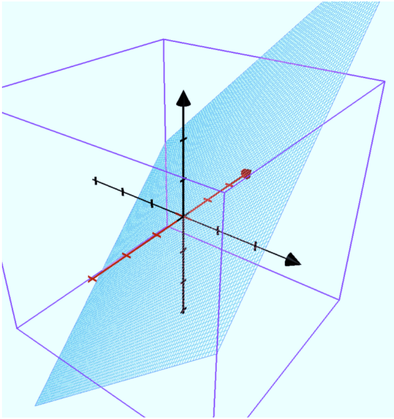
$$
\begin{aligned}
& logB(\langle \mathbf{b}, x\rangle) \blacktriangleleft \{b > 1, x \ge 0\} \\
& logB_w(\langle b, x, l_1, y_1\rangle) \blacktriangleleft \\
& \quad \{b > 1, x \ge 1, l_1 \ge 0, \mathbf{y_1} = \_\}
\end{aligned}
$$

- Loop transition of $log2_w$:

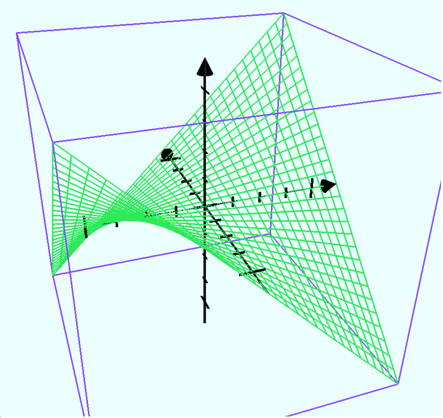
$$
\begin{aligned}
& \langle b, x, l_1, y_1\rangle \to \langle b, x, l_2, y_2\rangle \blacktriangleleft \\
& \quad \{x > \mathbf{y_1}, \mathbf{y_2} = \_, l_2 = l_1 + 1\}
\end{aligned}
$$

- **This loop is non-terminating**

Linear or non-linear operations



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



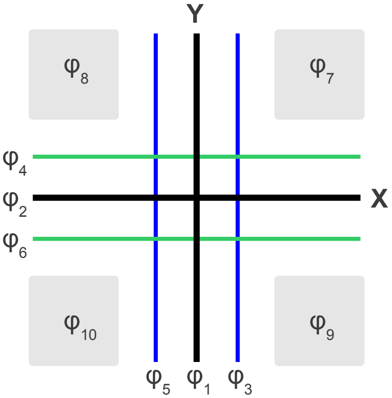
A nonlinear operation like $z = x * y$ can only be modeled as $\top$

Solution: disjunctive abstraction of $z = x * y$

Example ($z = x * y$)

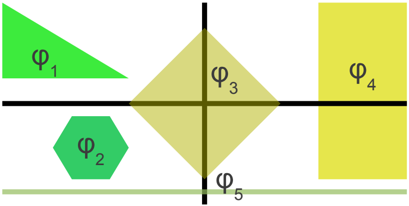
$$
\begin{array}{lll}
\varphi_1 & \equiv & \{x = 0\}, \{z = 0\} \\
\varphi_2 & \equiv & \{y = 0\}, \{z = 0\} \\
\varphi_3 & \equiv & \{x = 1\}, \{z = y\} \\
\varphi_4 & \equiv & \{y = 1\}, \{z = x\} \\
\varphi_5 & \equiv & \{x = -1\}, \{z = -y\} \\
\varphi_6 & \equiv & \{y = -1\}, \{z = -x\} \\
\varphi_7 & \equiv & \{x \geq 2, y \geq 2\} \\
 & & \{z \geq 2x, z \geq 2y\} \\
\varphi_8 & \equiv & \{x \geq 2, y \leq -2\} \\
 & & \{z \leq -2x, z \leq 2y\} \\
\varphi_9 & \equiv & \{x \leq -2, y \geq 2\} \\
 & & \{z \leq 2x, z \leq -2y\} \\
\varphi_{10} & \equiv & \{x \leq -2, y \leq -2\} \\
 & & \{z \geq -2x, z \geq -2y\}
\end{array}
$$

Input space of $z = x * y$



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 $\bigstar$ to a **finite** disjunction $\varphi_1 \vee \varphi_2 \vee \cdots \vee \varphi_n$.



Using disjunctive information

(a) We could employ a value analysis over over a disjunctive domain **but using a disjunctive abstract domain doesn't scale**

Using disjunctive information

(a) We could employ a value analysis over 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 $z = x * y$ and it knows that $x \geq 1$ and $y \geq 2$, we want it to use only the satisfiable cases

$$\begin{array}{lll} \varphi_3 \equiv & \{x = 1\} & \{z = y\} \\ \varphi_7 \equiv & \{x \geq 2, y \geq 2\} & \{z \geq 2x, z \geq 2y\} \end{array}$$

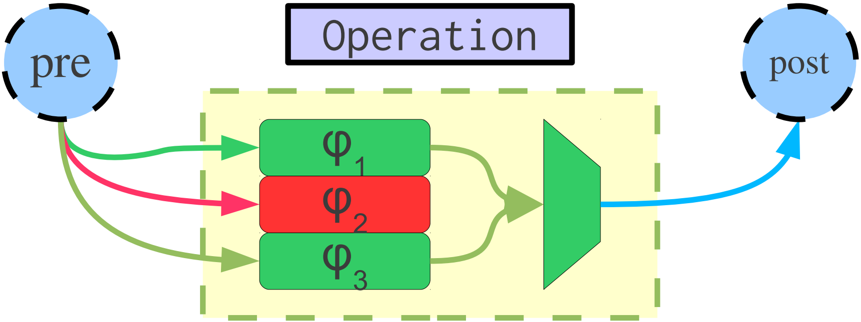
and ignore the (unsatisfiable) rest for computing the postcondition $\{z = y\} \sqcup \{z \geq 2x, z \geq 2y\} = \{z \geq 2x, z \geq y\}$

Using disjunctive information

- Replace each code $b \equiv z = x \bigstar y$ by a call to $op_{\bigstar^b}(\langle x, y \rangle, \langle z \rangle)$.
- $op_{\bigstar^b}(\langle x, y \rangle, \langle z \rangle)$ is defined with one rule per case.

Using disjunctive information

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



Analyzing logB by abstract program transformation

Example (Program transformation of `logB`)

```
// 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;
}
```

$$
\begin{array}{l}
logB(\langle \mathbf{b}, x\rangle, \langle l\rangle) \leftarrow \\
\quad \{l_0 = 0\}, \\
\quad \{y_0 = 1\}, \\
\quad logB_w(\langle \mathbf{b}, x, l_0, y_0\rangle, \langle l\rangle). \\
logB_w(\langle \mathbf{b}, x, l_1, y_1\rangle, \langle l_2\rangle) \leftarrow \\
\quad \{x > y_1\}, \\
\quad \mathbf{op}_*(\langle \mathbf{y_1}, \mathbf{b}\rangle, \langle \mathbf{y_2}\rangle), \\
\quad \{l_2 = l_1 + 1\}, \\
\quad logB_w(\langle \mathbf{b}, x, l_2, y_2\rangle, \langle l_2\rangle). \\
logB_w(\langle \mathbf{b}, x, l, y_1\rangle, \langle l\rangle) \leftarrow \\
\quad \{x \leq y_1\}.
\end{array}
$$

The solution

Example (Value analysis of op_*)

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

$$
\begin{array}{lll}
op_*(\langle y_1, b\rangle, \langle y_2\rangle) & \leftarrow \{y_1 = 0\}, & \ldots \\
op_*(\langle y_1, b\rangle, \langle y_2\rangle) & \leftarrow \{b = 0\}, & \ldots \\
op_*(\langle y_1, b\rangle, \langle y_2\rangle) & \leftarrow \{b = 1\}, & \ldots \\
\mathbf{op}_*(\langle \mathbf{y_1}, \mathbf{b}\rangle, \langle \mathbf{y_2}\rangle) & \leftarrow \{\mathbf{y_1 = 1}\}, & \leftarrow \{\mathbf{y_2 = b}\}. \\
\mathbf{op}_*(\langle \mathbf{y_1}, \mathbf{b}\rangle, \langle \mathbf{y_2}\rangle) & \leftarrow \{\mathbf{b \geq 2, y_1 \geq 2}\}, & \leftarrow \{\mathbf{y_2 \geq 2y_1, y_2 \geq 2b}\}. \\
op_*(\langle y_1, b\rangle, \langle y_2\rangle) & \leftarrow \{y_1 \geq 2, b \leq -2\}, & \ldots \\
op_*(\langle y_1, b\rangle, \langle y_2\rangle) & \leftarrow \{y_1 \leq -2, b \geq 2\}, & \ldots \\
op_*(\langle y_1, b\rangle, \langle y_2\rangle) & \leftarrow \{y_1 \leq -2, b \leq -2\}, & \ldots
\end{array}
$$

The value analysis computes the postcondition

$$
op_*(\langle y_1, b\rangle, \langle y_2\rangle) \blacktriangleleft \quad \{y_2 = b\} \sqcup \{y_2 \geq 2y_1, y_2 \geq 2b\} = \{\mathbf{y_2 \geq 2y_1, y_2 \geq b}\}
$$

The solution

Example (Bounding the iterations of `logB`)

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

$$\langle b, x, l_1, y_1\rangle \rightarrow \langle b, x, l_2, y_2\rangle \blacktriangleleft \begin{array}{l} \{\mathbf{y_1 \geq 1, b \geq 2}\} \sqcap \\ \{\mathbf{x > y_1}, l_2 = l_1 + 1\} \sqcap \\ \{\mathbf{y_2 \geq 2y_1}, y_2 \geq b\} \end{array}$$

- This transition can be proven to have $\mathcal{O}(log(x - y))$ iterations
- COSTA **can now infer that `logB` has a logarithmic cost**

Conclusions

- Nonlinear operations are difficult to analyze
- We propose to use a program transformation for handling nonlinear operations like $z = x * y$ 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 $(\&, |, <<, >>, >>>)$