JULIASOFT
code analysis reinvented

CIL to Java-bytecode Translation for Static Analysis Leveraging

Pietro Ferrara — JuliaSoft SRL
Agostino Cortesi — University of Venice
Fausto Spoto — University of Verona

June, 2nd 2018

Static Program Analysis

- Automatic analysis of the behavior of software
- Target some specific properties
- Static: without executing the program
  - Need a model of the semantic of the programming language
- Sound: detect all bugs w.r.t. the property of interest
  - False alarms: the analysis failes to prove correct a correct sw

Properties
- SQL Injection
- NullPointerExceptions
- ...

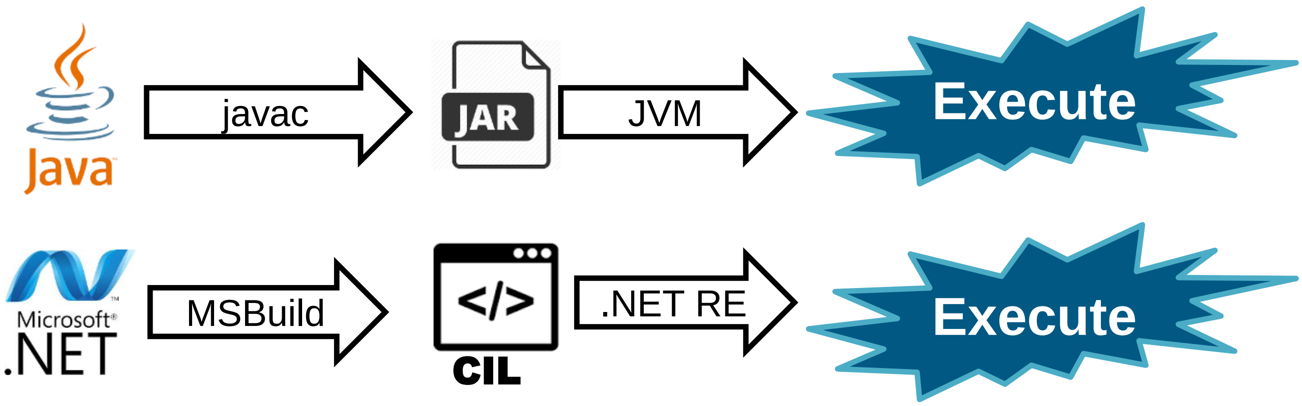
SPA

[Injection] possible SQL-injection

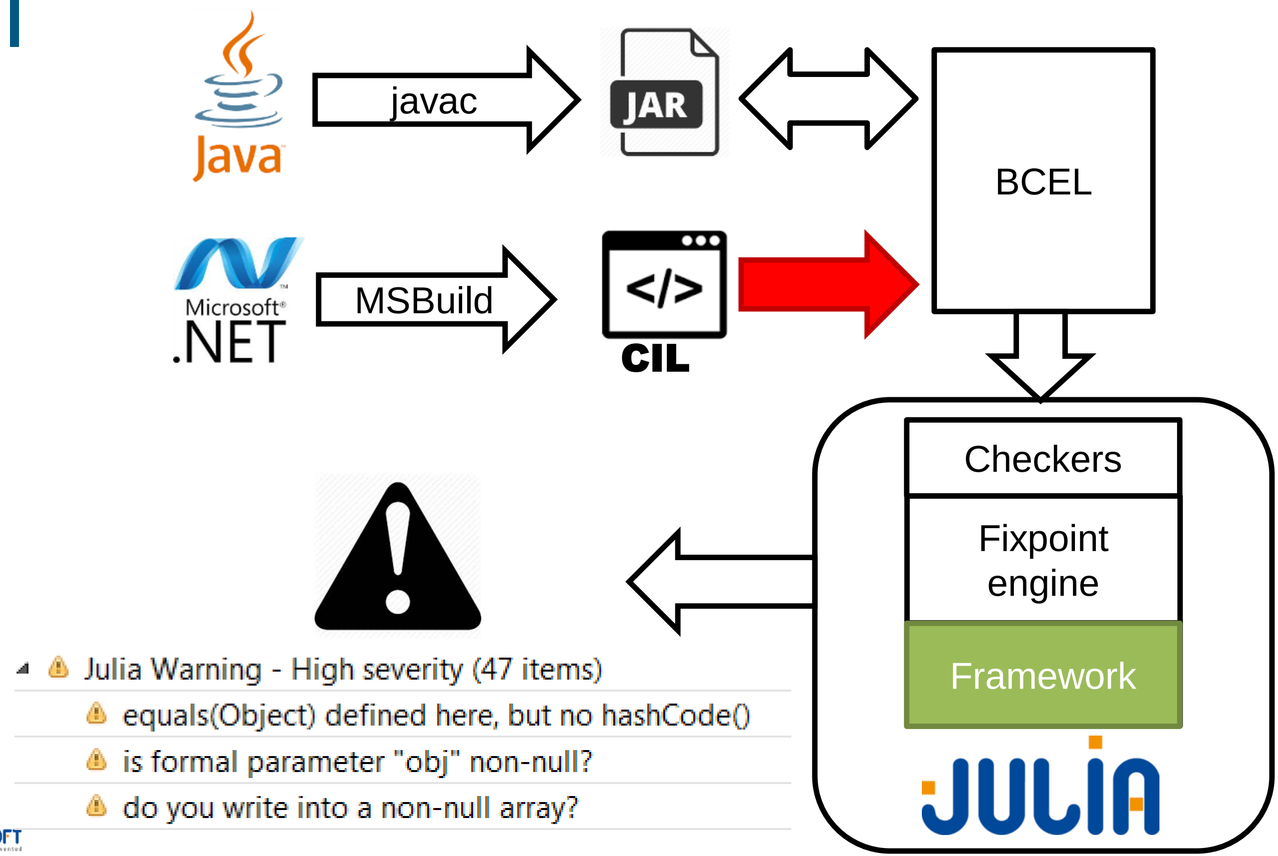
[BasicNullness] is the receiver of "getProperty" non-null?

Bytecode languages

- Machine-independent low-level languages
- Interpreted or Just-In-Time compiled
- Based on local variables, stack of values, heap
- Object-oriented (for the purposes of this work)

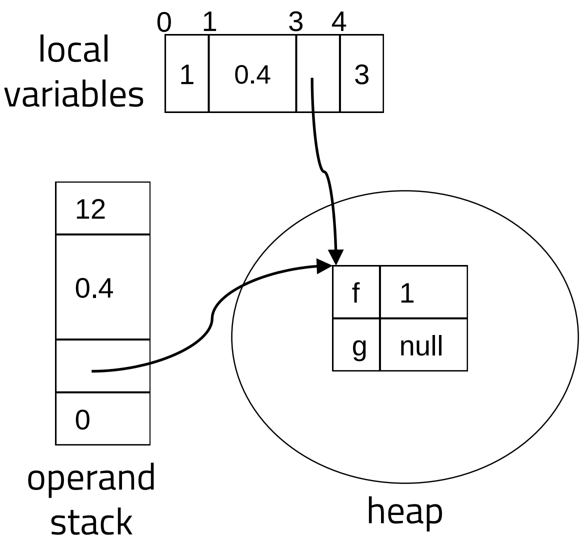


The Julia static analyzer

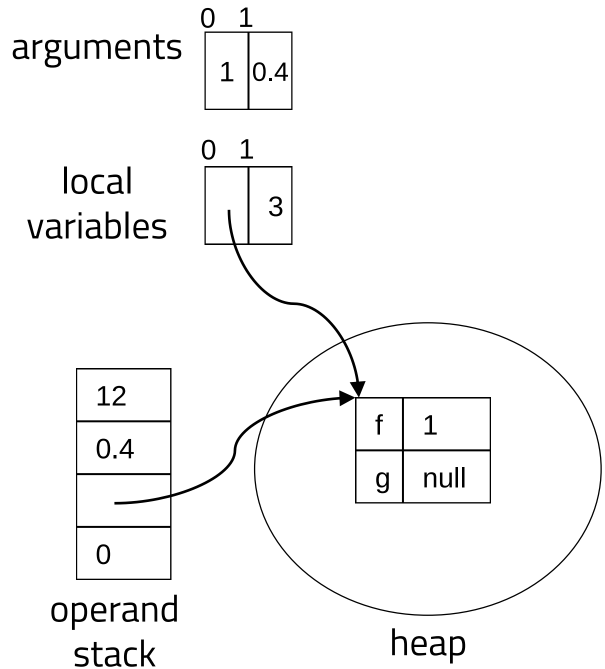


Concrete states

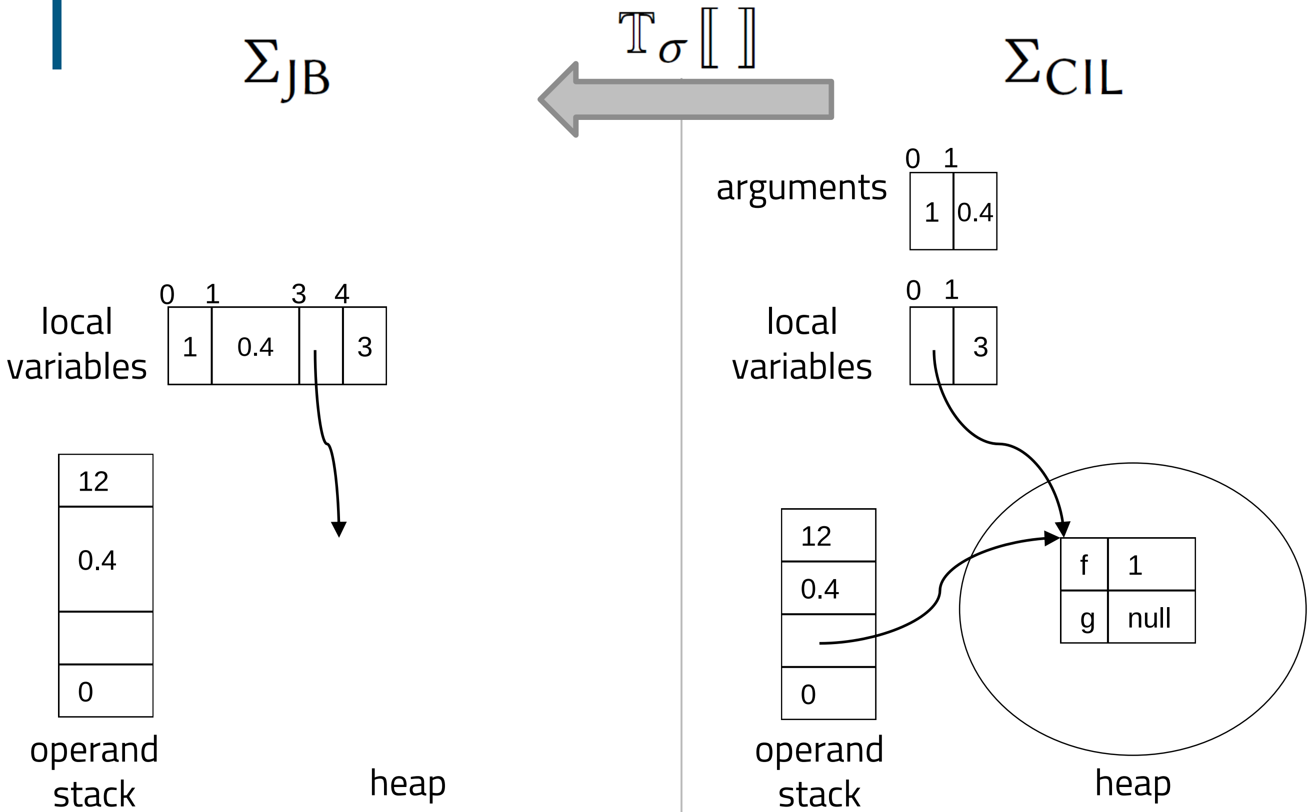
Java Bytecode



MS CIL



States translation $\mathbb{T}_\sigma[\![\ ]\!] : \Sigma_{\mathrm{CIL}} \to \Sigma_{\mathrm{JB}}$



Java Bytecode (JB) vs. CIL

JB	CIL	
iadd ladd	add	*(arith. op.)*
iload i lload i aload i istore i lstore i astore i	ldloc i stloc i ldarg i	*(local vars)*
invo invo		*ll)*
new getf putf		
if_i		*anch)*
dup dup2	p	*(stack)*
	ldloca i stind ldind	*(pointers)*

- CIL more expressive than JB
- Some uses of pointers do not have an equivalent in JB
- Focus on CIL derived from safe C# code

JB: typed
CIL: untyped

JB: local vars
CIL: local vars + args

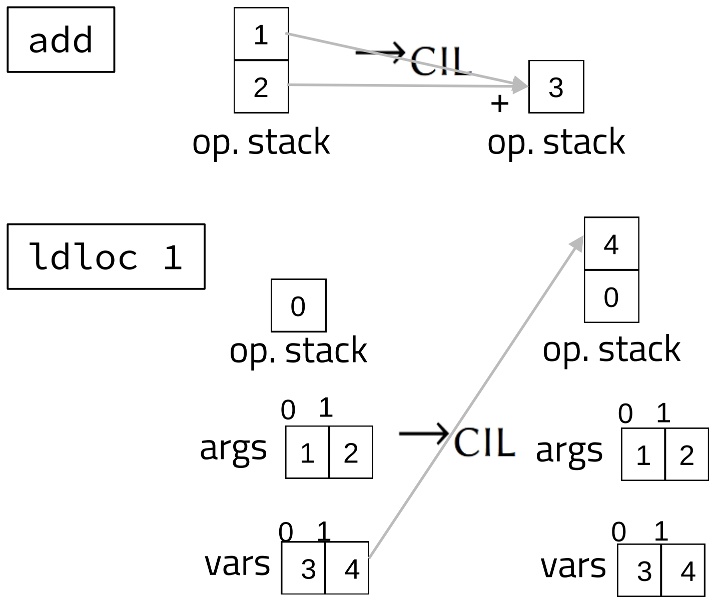
JB: static and dynamic
CIL: generic

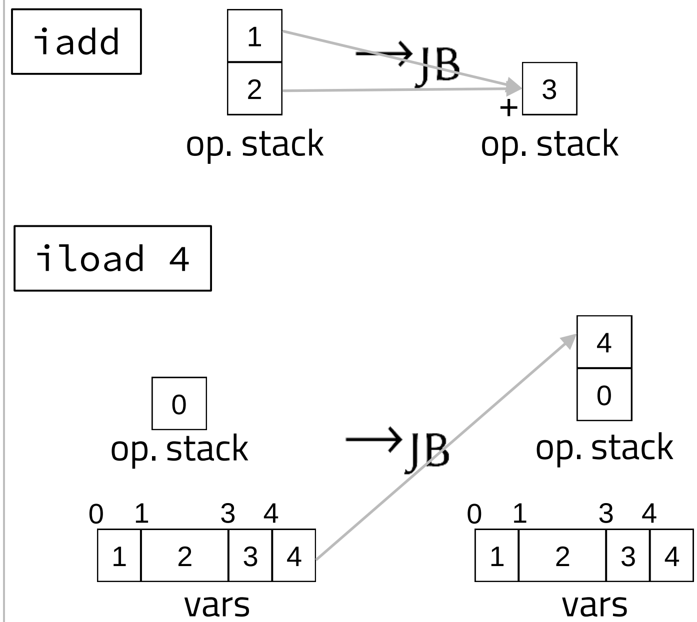
JB: typed
CIL: untyped

CIL: unique values
JB: 32- and 64-bit values

CIL: direct pointers

Concrete semantics





Concrete semantics $\rightarrow_{\mathrm{CIL}}$

$$\frac{typeOf(v_1) = typeOf(v_2)}{\langle \texttt{add}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: (v_1 + v_2), \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{add})$$

$$\frac{}{\langle \texttt{ldloc i}, (s, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: \mathsf{l}(\texttt{i}), \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{ldloc})$$

$$\frac{}{\langle \texttt{stloc i}, (s :: \mathsf{v}, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s, \mathsf{l}[\texttt{i} \mapsto \mathsf{v}], \mathsf{a}, \mathsf{h})}\ (\texttt{stloc})$$

$$\frac{}{\langle \texttt{ldarg i}, (s, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: \mathsf{a}(\texttt{i}), \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{ldarg})$$

$$\frac{\begin{array}{c} isStatic(\texttt{m}(\texttt{arg}_0, \cdots, \texttt{arg}_\texttt{i})) = \mathsf{false} \wedge t \neq \texttt{null} \wedge \\ \langle body(\texttt{m}(\texttt{arg}_0, \cdots, \texttt{arg}_\texttt{i}), (t, v_1, \cdots, v_i)), ([], \emptyset, [0 \mapsto t, j \mapsto v_j : j \in [1..i]], \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (\mathsf{s}', \mathsf{l}', \mathsf{a}', \mathsf{h}') \end{array}}{\langle \texttt{call m}(\texttt{arg}_1, \cdots, \texttt{arg}_\texttt{i}), (s :: t :: v_1 :: \cdots :: v_i, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (\mathsf{s}, \mathsf{l}, \mathsf{a}, \mathsf{h}')}\ (\texttt{call})$$

$$\frac{\begin{array}{c} isStatic(\texttt{m}(\texttt{arg}_0, \cdots, \texttt{arg}_\texttt{i})) = \mathsf{true} \wedge \\ \langle body(\texttt{m}(\texttt{arg}_0, \cdots, \texttt{arg}_\texttt{i}), (v_1, \cdots, v_i)), ([], \emptyset, [j-1 \mapsto v_j : j \in [1..i]], \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (\mathsf{s}', \mathsf{l}', \mathsf{a}', \mathsf{h}') \end{array}}{\langle \texttt{call m}(\texttt{arg}_1, \cdots, \texttt{arg}_\texttt{i}), (s :: v_1 :: \cdots :: v_i, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (\mathsf{s}, \mathsf{l}, \mathsf{a}, \mathsf{h}')}\ (\texttt{callstatic})$$

$$\frac{fresh(\mathsf{T}, \mathsf{h}) = (r, \mathsf{h}_1) \wedge \langle body(\texttt{ctor}(\texttt{arg}_1, \cdots, \texttt{arg}_\texttt{i}), (v_1, \cdots, v_i)), ([], \emptyset, [0 \mapsto r, j \mapsto v_j : j \in [1..i]], \mathsf{h}_1)\rangle \rightarrow_{\mathrm{CIL}} (\mathsf{s}', \mathsf{l}', \mathsf{a}', \mathsf{h}')}{\langle \texttt{newobj T}(\texttt{a}_1, \cdots, \texttt{a}_\texttt{i}), (s :: v_1 :: \cdots :: v_i, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: r, \mathsf{l}, \mathsf{a}, \mathsf{h}')}\ (\texttt{newobj})$$

$$\frac{o \neq \texttt{null}}{\langle \texttt{ldfld f}, (s :: o, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: \mathsf{h}(o)(\mathsf{f}), \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{ldfld})$$

$$\frac{o \neq \texttt{null} \quad s' = \mathsf{h}(o)[\mathsf{f} \mapsto v]}{\langle \texttt{stfld f}, (s :: o :: v, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s, \mathsf{l}, \mathsf{a}, \mathsf{h}[o \mapsto s'])}\ (\texttt{stfld})$$

$$\frac{typeOf(v_1) = typeOf(v_2) \wedge v_1 > v_2}{\langle \texttt{bgt l}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} \langle \texttt{l}, (s, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle}\ (\texttt{bgt true})$$

$$\frac{typeOf(v_1) = typeOf(v_2) \wedge v_1 \leq v_2}{\langle \texttt{bgt l}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s, \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{bgt false})$$

$$\frac{}{\langle \texttt{ldloca i}, (s, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: r_i, \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{ldloca})$$

$$\frac{}{\langle \texttt{stind}, (s :: r_i :: v, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s, \mathsf{l}, \mathsf{a}, \mathsf{h}[r_i \mapsto v])}\ (\texttt{stind})$$

$$\frac{}{\langle \texttt{dup}, (s :: v, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: v :: v, \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{dup})$$

$$\frac{}{\langle \texttt{ldind}, (s :: r_i, \mathsf{l}, \mathsf{a}, \mathsf{h})\rangle \rightarrow_{\mathrm{CIL}} (s :: \mathsf{h}(r_i), \mathsf{l}, \mathsf{a}, \mathsf{h})}\ (\texttt{ldind})$$

Figure 5: Concrete CIL semantics.

Concrete semantics $\rightarrow_{\mathsf{JB}}$

$$\frac{\mathit{typeOf}(v) \neq \mathsf{Long}}{\langle \mathsf{dup}, (s :: v, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: v :: v, \mathsf{l}, \mathsf{h})}\ (\mathsf{dup})$$

$$\frac{\mathit{typeOf}(v_1) \neq \mathsf{Long}}{\langle \mathsf{dup2}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: v_1 :: v_2 :: v_1 :: v_2, \mathsf{l}, \mathsf{h})}\ (\mathsf{dup2}\ 32) \qquad \frac{\mathit{typeOf}(v) = \mathsf{Long}}{\langle \mathsf{dup2}, (s :: v, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: v :: v, \mathsf{l}, \mathsf{h})}\ (\mathsf{dup2}\ 64)$$

$$\frac{\mathit{typeOf}(v_1) = \mathsf{Int} \wedge \mathit{typeOf}(v_2) = \mathsf{Int}}{\langle \mathtt{iadd}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: (v_1 + v_2), \mathsf{l}, \mathsf{h})}\ (\mathtt{iadd}) \qquad \frac{\mathit{typeOf}(v_1) = \mathsf{Long} \wedge \mathit{typeOf}(v_2) = \mathsf{Long}}{\langle \mathsf{ladd}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: (v_1 + v_2), \mathsf{l}, \mathsf{h})}\ (\mathtt{ladd})$$

$$\frac{x = \mathit{JVMprefix}(\mathit{typeOf}(\mathsf{l}(\mathtt{i})))}{\langle x\mathtt{load}\ \mathtt{i}, (s, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: \mathsf{l}(\mathtt{i}), \mathsf{l}, \mathsf{h})}\ (x\mathtt{load}) \qquad \frac{x = \mathit{JVMprefix}(\mathit{typeOf}(\mathtt{v}))}{\langle x\mathtt{store}\ \mathtt{i}, (s :: \mathtt{v}, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s, \mathsf{l}[\mathtt{i} \mapsto \mathtt{v}], \mathsf{h})}\ (x\mathtt{store})$$

$$\frac{\begin{array}{c}\mathit{isStatic}(\mathsf{m}(\mathtt{arg}_0, \cdots, \mathtt{arg}_i)) = \mathsf{false} \wedge t \neq \mathtt{null} \wedge \\ \langle \mathit{body}(\mathsf{m}(\mathtt{arg}_0, \cdots, \mathtt{arg}_i), (t, v_1, \cdots, v_i)), ([], [0 \mapsto t, j \mapsto v_j : j \in [1..i]], \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (\mathsf{s}', \mathsf{l}', \mathsf{h}')\end{array}}{\langle \mathtt{invokevirtual}\ \mathsf{m}(\mathtt{arg}_1, \cdots, \mathtt{arg}_i), (s :: t :: v_1 :: \cdots :: v_i, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s, \mathsf{l}, \mathsf{h}')}\ (\mathtt{invokevirtual})$$

$$\frac{\begin{array}{c}\mathit{isStatic}(\mathsf{m}(\mathtt{arg}_0, \cdots, \mathtt{arg}_i)) = \mathsf{true} \wedge \\ \langle \mathit{body}(\mathsf{m}(\mathtt{arg}_0, \cdots, \mathtt{arg}_i), (v_1, \cdots, v_i)), ([], [j - 1 \mapsto v_j : j \in [1..i]], \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (\mathsf{s}', \mathsf{l}', \mathsf{h}')\end{array}}{\langle \mathtt{invokestatic}\ \mathsf{m}(\mathtt{arg}_1, \cdots, \mathtt{arg}_i), (s :: v_1 :: \cdots :: v_i, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s, \mathsf{l}, \mathsf{h}')}\ (\mathtt{invokestatic})$$

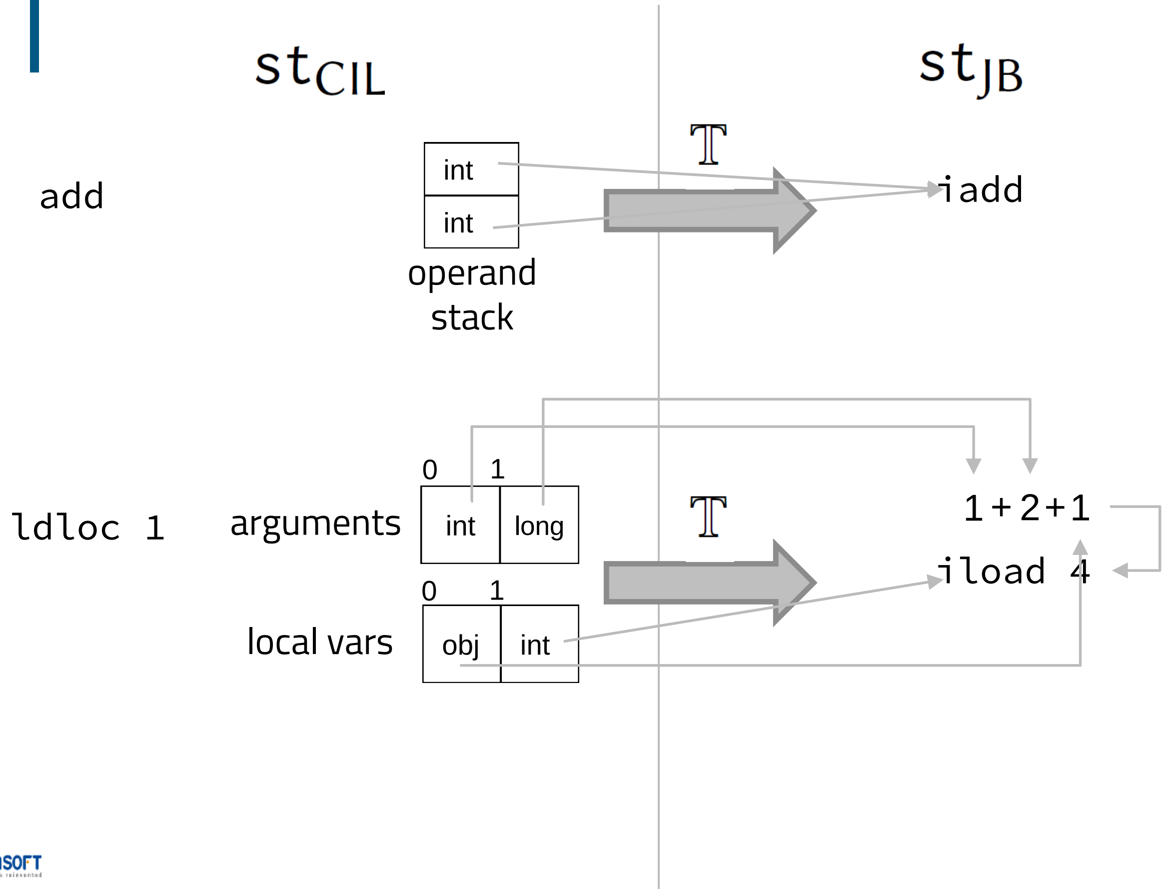
$$\frac{\mathit{fresh}(\mathsf{T}, \mathsf{h}) = (r, \mathsf{h}')}{\langle \mathsf{new}\ \mathsf{T}, (s, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: r, \mathsf{l}, \mathsf{h}')}\ (\mathsf{new}) \qquad \frac{o \neq \mathtt{null}}{\langle \mathtt{getfield}\ \mathsf{f}, (s :: o, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s :: \mathsf{h}(o)(\mathsf{f}), \mathsf{l}, \mathsf{h})}\ (\mathtt{getfield})$$

$$\frac{o \neq \mathtt{null} \quad s' = \mathsf{h}(o)[\mathsf{f} \mapsto v]}{\langle \mathtt{putfield}\ \mathsf{f}, (s :: o :: v, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s, \mathsf{l}, \mathsf{h}[o \mapsto s'])}\ (\mathtt{putfield}) \qquad \frac{\mathit{typeOf}(v_1) = \mathsf{Int} \wedge \mathit{typeOf}(v_2) = \mathsf{Int} \wedge v_1 > v_2}{\langle \mathtt{if\_icmpgt}\ \mathtt{l}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} \langle \mathtt{l}, (s, \mathsf{l}, \mathsf{h})\rangle}\ \begin{pmatrix}\mathtt{if\_icmpgt} \\ \mathtt{true}\end{pmatrix}$$

$$\frac{\mathit{typeOf}(v_1) = \mathsf{Int} \wedge \mathit{typeOf}(v_2) = \mathsf{Int} \wedge v_1 \leq v_2}{\langle \mathtt{if\_icmpgt}\ \mathtt{l}, (s :: v_1 :: v_2, \mathsf{l}, \mathsf{h})\rangle \rightarrow_{\mathsf{JB}} (s, \mathsf{l}, \mathsf{h})}\ \begin{pmatrix}\mathtt{if\_icmpgt} \\ \mathtt{false}\end{pmatrix}$$

Figure 6: Concrete JB semantics.

Statement translation $\mathbb{T}[\![\mathtt{st}_{\mathrm{CIL}}, \mathrm{K}]\!] = \mathtt{st}_{\mathrm{JB}}$



Statement translation $\mathbb{T}\llbracket \mathtt{st}_{\mathrm{CIL}}, \mathrm{K} \rrbracket = \mathtt{st}_{\mathrm{JB}}$

$$\mathbb{T}\llbracket \mathrm{dup}, \overline{\mathrm{s}} :: \mathrm{t}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \begin{cases} \mathrm{dup} & \text{if } t \neq \mathrm{Long} \\ \mathrm{dup2} & \text{if } t = \mathrm{Long} \end{cases}$$

$$\mathbb{T}\llbracket \mathrm{add}, \overline{\mathrm{s}} :: \mathrm{t}_1 :: \mathrm{t}_2, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \begin{cases} \mathtt{iadd} & \text{if } t_1 = t_2 = \mathrm{Int} \\ \mathtt{ladd} & \text{if } t_1 = t_2 = \mathrm{Long} \end{cases}$$

$$\mathbb{T}\llbracket \mathtt{ldloc\ i}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = x\mathtt{load}\ j \text{ where } j = |\overline{\mathrm{a}}| + 64^{|\overline{\mathrm{a}}|}_{\overline{\mathrm{a}}} + i + 64^{i}_{\bar{\mathrm{l}}} \wedge x = \mathit{JVMprefix}(\mathit{typeOf}(\bar{\mathrm{l}}(\mathrm{i})))$$

$$\mathbb{T}\llbracket \mathtt{stloc\ i}, \overline{\mathrm{s}} :: \mathrm{t}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = x\mathtt{store}\ j \text{ where } j = |\overline{\mathrm{a}}| + 64^{|\overline{\mathrm{a}}|}_{\overline{\mathrm{a}}} + i + 64^{i}_{\bar{\mathrm{l}}} \wedge x = \mathit{JVMprefix}(\mathit{typeOf}(\bar{\mathrm{l}}(\mathrm{i})))$$

$$\mathbb{T}\llbracket \mathtt{ldarg\ i}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = x\mathtt{load}\ j \text{ where } j = i + 64^{i}_{a} \wedge x = \mathit{JVMprefix}(\mathit{typeOf}(\overline{\mathrm{a}}(\mathrm{i})))$$

$$\begin{aligned} &\mathbb{T}\llbracket \mathtt{call}\ \mathrm{m}(\mathrm{arg}_1, \cdots, \mathrm{arg}_\mathrm{i}), \\ &\quad \overline{\mathrm{s}} :: t_1 :: \cdots :: t_i, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} :: p_1 :: \cdots :: p_i \rrbracket \\ &= \mathtt{invoke}\ ;\ \mathtt{aload}\ p^1_{idx_1}\ ;\ \mathtt{getfield\ value}\ ;\ x_{idx_1}\mathtt{store}\ p^2_{idx_1}\ ;\ \cdots \\ &\qquad \cdots \mathtt{aload}\ p^1_{idx_j}\ ;\ \mathtt{getfield\ value}\ ;\ x_{idx_j}\mathtt{store}\ p^2_{idx_j}\ ; \\ &\quad \text{where } \mathtt{invoke} = \begin{cases} \mathtt{invokestatic}\ \mathrm{m}(\mathrm{arg}_1, \cdots, \mathrm{arg}_\mathrm{i}) & \text{if } \mathit{isStatic}(\mathrm{m}(\mathrm{arg}_1, \cdots, \mathrm{arg}_\mathrm{i})) \\ \mathtt{invokevirtual}\ \mathrm{m}(\mathrm{arg}_1, \cdots, \mathrm{arg}_\mathrm{i}) & \text{otherwise} \end{cases} \\ &\quad \{idx_1, \cdots, idx_j\} = \{k : \mathrm{arg}_\mathrm{k} \in \mathrm{Ref}_{\mathrm{Loc}}\} \\ &\quad \forall k \in [1..j] : x_{idx_k} = \mathit{JVMprefix}(\mathit{typeOf}(\bar{\mathrm{l}}(p^2_{idx_k}))) \wedge \forall r \in [1..i] : p_i = (p^1_i, p^2_j) \end{aligned}$$

$$\begin{aligned} &\mathbb{T}\llbracket \mathtt{newobj}\ \mathrm{T}(\mathrm{a}_1, \cdots, \mathrm{a}_\mathrm{i}), \\ &\quad \overline{\mathrm{s}} :: t_1 :: \cdots :: t_i, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket \\ &= x_i\mathtt{store}\ idx_i\ ;\ \cdots\ ;\ x_1\mathtt{store}\ idx_1\ ;\ \mathtt{new\ T}\ ;\ \mathtt{dup}\ ; \\ &\qquad x_1\mathtt{load}\ idx_1\ ;\ \cdots\ ;\ x_i\mathtt{load}\ idx_i\ ;\ \mathtt{invokevirtual}\ <\mathtt{init}>(\mathrm{arg}_1, \cdots, \mathrm{arg}_\mathrm{i}) \\ &\quad \text{where } \forall j \in [1..i] : x_j = \mathit{JVMprefix}(a_j) \wedge idx_j = \mathit{freshIdx}(\mathtt{newobj}\ \mathrm{T}(\mathrm{a}_1, \cdots, \mathrm{a}_\mathrm{i}), j) \end{aligned}$$

$$\mathbb{T}\llbracket \mathtt{ldfld\ f}, \overline{\mathrm{s}} :: \mathrm{t}_\mathrm{o}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \mathtt{getfield\ f}$$

$$\mathbb{T}\llbracket \mathtt{stfld\ f}, \overline{\mathrm{s}} :: \mathrm{t}_\mathrm{o} :: \mathrm{t}_\mathrm{v}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \mathtt{putfield\ f}$$

$$\mathbb{T}\llbracket \mathrm{bgt\ k}, \overline{\mathrm{s}} :: \mathrm{t}_1 :: \mathrm{t}_2, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \mathtt{if\_icmpgt}\ \mathrm{k}' \text{ where } \mathrm{k}' = \mathit{statementIdx}(\mathit{getBody}(\mathrm{bgt\ k})(\mathrm{k})) \text{ if } t_1 = t_2 = \mathrm{Int}$$

$$\begin{aligned} \mathbb{T}\llbracket \mathtt{ldloca\ i}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket &= \mathbb{T}\llbracket \mathtt{newobj\ WrapRef()}\ ;\ \mathtt{dup2}\ ;\ \mathtt{stloc\ j};\ \mathtt{ldloc\ i};\ \mathtt{stfld\ value}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket \\ &\quad \text{where } j = \mathit{freshIdx}(\mathtt{ldloca\ i}, 0) \end{aligned}$$

$$\mathbb{T}\llbracket \mathtt{stind}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \mathbb{T}\llbracket \mathtt{stfld\ value}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket$$

$$\mathbb{T}\llbracket \mathtt{ldind}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket = \mathbb{T}\llbracket \mathtt{ldfld\ value}, \overline{\mathrm{s}}, \bar{\mathrm{l}}, \overline{\mathrm{a}}, \overline{\mathrm{w}} \rrbracket$$

JULIASOFT
code analysis reinvented

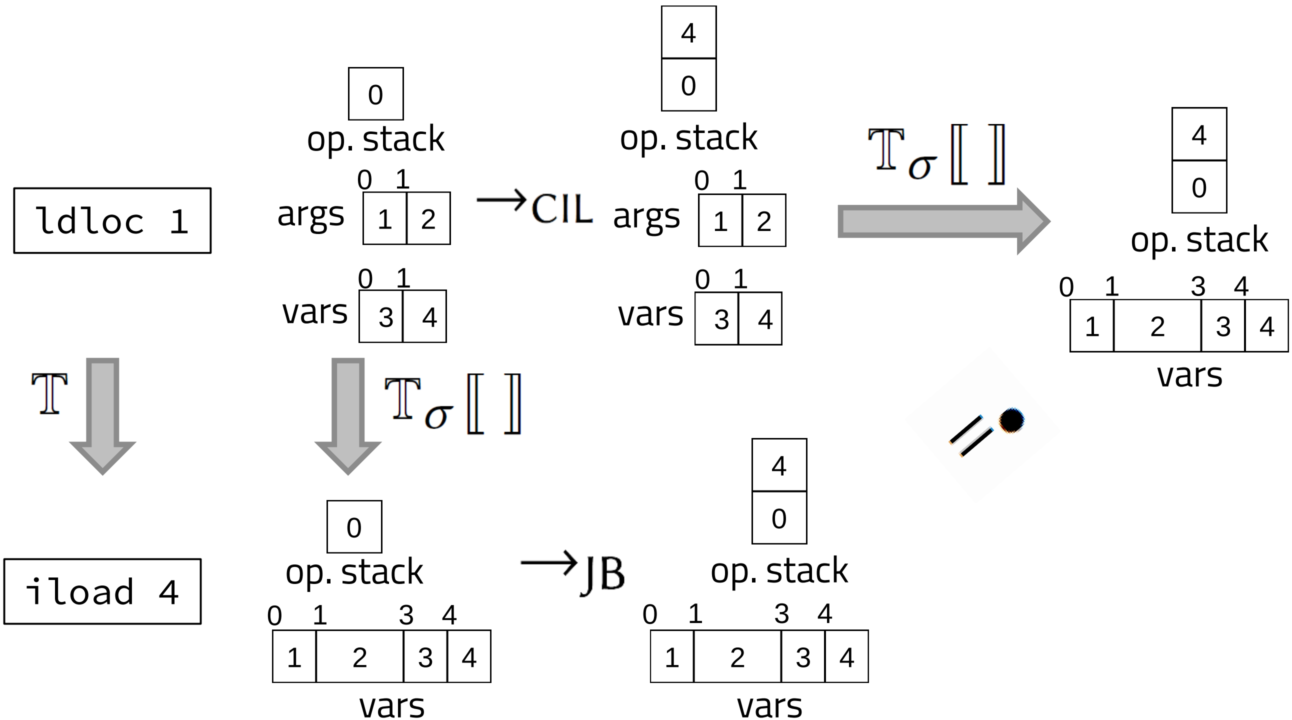
Correctness

$$\forall \mathtt{st} \in \mathrm{St}_{\mathrm{CIL}}, \sigma_{\mathrm{CIL}} \in \Sigma_{\mathrm{CIL}} :$$

$$\boxed{\langle \mathtt{st}, \sigma_{\mathrm{CIL}} \rangle \to_{\mathrm{CIL}}} \; \sigma'_{\mathrm{CIL}} \text{ and } \boxed{\langle \mathbb{T}[\![\mathtt{st}, \mathrm{K}]\!]}, \boxed{\mathbb{T}_\sigma[\![\sigma_{\mathrm{CIL}}]\!]} \rangle \to_{\mathrm{JB}} \sigma'_{\mathrm{JB}}$$

$$\Downarrow$$

$$\boxed{\mathbb{T}_\sigma[\![\sigma'_{\mathrm{CIL}}]\!]} \boxed{=_\bullet} \; \sigma'_{\mathrm{JB}}$$



Experimental results

Efficiency and precision

Table 1: Experimental results on the 5 most starred Github C# projects.

Program	LOC	met.	fail	Tr. t.	An.t.	Al	F	Prec.
CodeHub	32,510	4,887	0	0'07"	0'43"	9	1	89%
SignalR	71,207	6,610	3	0'07"	0'50"	8	1	88%
Dapper	22,513	1,058	0	0'07"	0'29"	13	3	77%
ShareX	171,580	11,568	14	0'58"	2'08"	57	0	100%
Nancy	109,139	8,817	0	0'07"	1'25"	18	1	94%
Total	406,949	32,940	17	1'26"	4'35"	105	6	94%

- 5 most popular GitHub repositories written in C#
- Efficiency:
  - Analyze industrial-size software in a few minutes
  - Translation time comparable to the analysis time.
- Precision:
  - 6/105 alarms (6%) are false because the translation

Experimental results
Libraries

Table 2: Experimental results on libraries.

Library	# met.	# fail	% fail	Tr. t.	Mem.
mscorlib	28,344	870	3.07%	23"	158
Sys.Core	6,988	47	0.68%	4"	96
Sys.Design	13,509	4	0.03%	20"	180
Sys	17,851	242	1.36%	21"	142
Sys.Runtime.Serial	5,624	74	1.32%	5"	86
Sys.ServiceModel	34,603	80	0.23%	34"	156
Sys.Web	28,249	38	0.13%	37"	216
Sys.Web.Extensions	4,245	0	0.00%	4"	109
Sys.Windows.Forms	28,319	53	0.19%	42"	189
Sys.XML	12,727	171	1.34%	23"	146
Total	180,460	1,579	0.87%	3'33"	

- 10 largest system libraries of .NET framework
- Study how much code we can translate
  - 99.13% of the methods, with a worst case of 96.93%

Limitations and future work

- Native [and unsafe] code
  - Code written in languages other than bytecode
  - Linked to and executed by the virtual machine/runtime env.
  - Static analysis requires a (manually written) model
  - Cannot be translated by our approach
    - We cannot execute the translation of a CIL program with the JVM
- "Naming conventions" introduced by the compiler
  - Need some polishing to link warnings back to the source code
- Support .NET frameworks
  - ASP.NET, etc..

JULIASOFT
code analysis reinvented

Thank you. Questions?

Corvallis Group
Management and Coordination by Corvallis Holding SpA

Lungadige Galtarossa 21
37133 Verona, Italy
Tel. + 39 045.2081901
info@juliasoft.com
www.juliasoft.com

Administrative Offices
Via Savelli, 56 - 35129 Padova, Italy
T +39 049.8434511 | F +39 049.8434555

CONTACTS:

pietro.ferrara@juliasoft.com