Abstract Interpretation and Program Transformation: techniques running after each other

Agostino Cortesi
Ca' Foscari University, Venice, Italy

VPT 2021
Ninth International Workshop on
Verification and Program Transformation

March 28th, 2021

Motivation

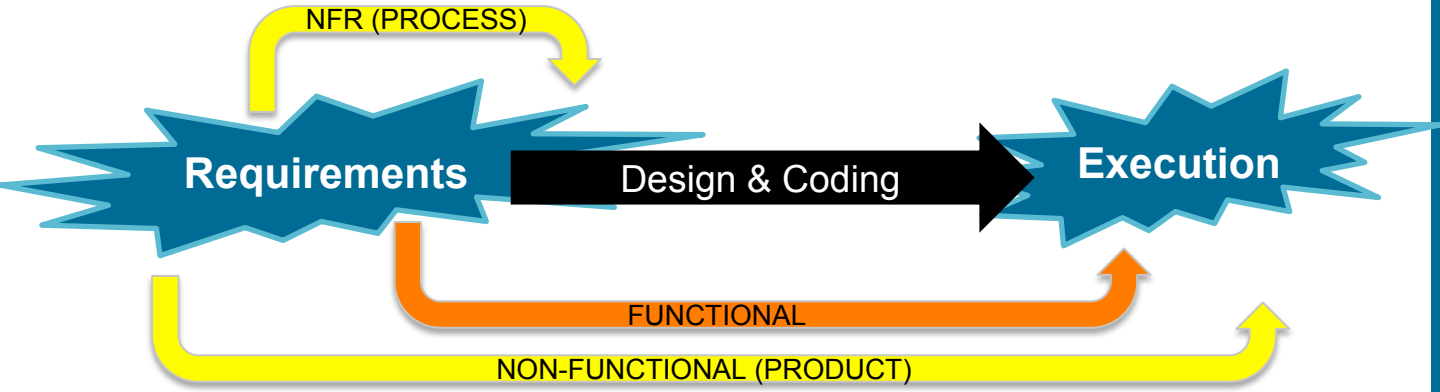
- An affectionate tribute to Alberto Pettorossi
  - The man who at dinner proves theorems on paper napkins
  - The man who never misses a Q&A session
- Share some general ideas on the different interrelationships between Abstract Interpretation and Program Transformation techniques
- Bring attention to the needs of software engineering in “real world”

Setting the scene - I

- In the software lifecycle everything starts from requirements
- Functional requirements
  - tell which services the system should provide (pre- & post-)
- Non-functional requirements
  - constraints on the services (product NFRs)
  - constraints on the way the services are realized (process NFRs)
- Product and Process NFRs are not independent, and conflicts arise often.
- The set of requirements represents the (possibly empty) space of acceptable software product solutions

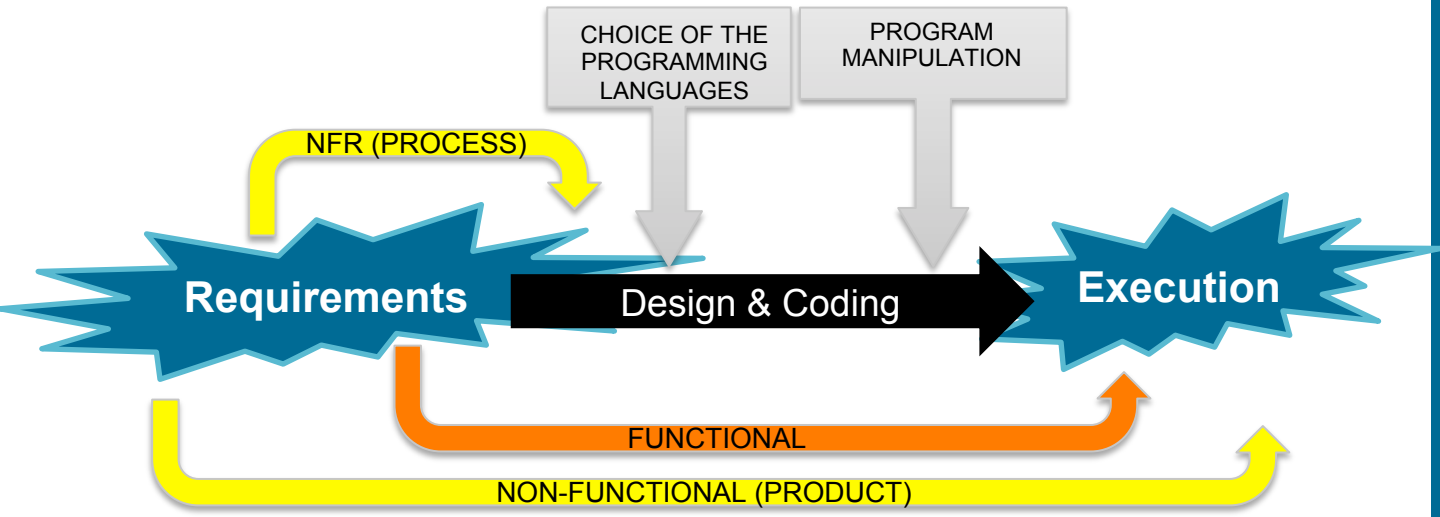
Setting the scene - I

- The starting point is the Requirement set R
- The final objective is an executable system that satisfies R
- Functional requirements
- Non functional product requirements
- Non functional process requirements



Setting the scene - II

- Program transformation and static analysis techniques apply to source code written in a given programming language
- The aims of these techniques are various: optimization, verification, etc.



The solution space of a requirements' set

- Let R= FR ∪ NFR be a set of requirements
- Let L be a set of programming languages
- The solution space S(R,L) of R in L is defined as the set of all the program codes P written in any language in L such that:

  P ∪ {t(I): t is an execution trace of c(P) with initial state in I} ∪ I

  satisfies all the requirements r in R, where c(P) is the compiled version of P.

Monotonicity

- The elements of S(R,L) are syntactic objects
- If R is the emptyset (no requirement) then S(R,L) is the set of all the programs that comply with the L syntax.
- Adding requirements reduces the solution space:
  if R and R' are requirement sets and R is a subset of R',
  then S(R',L) is a subset of S(R,L)
- If the set R is not consistent then S(R,L) is empty

Observe and constraint

- A requirement can be seen as the result of two actions: observe and constraint
  - E.g. observe only the input and the output, and make the constraint that the output is the square of the input
  - E.g. observe only the execution time, and make the constraint on the overall time efficiency
  - E.g. observe the modular structure, of the program code and make a constraint on the max size of each component

Observe and constraint (formally)

- At each requirement is associated a metric m_r and a threshold t_r
- P satisfies the requirement r if $m_r(P) \geq t_r$
  - E.g. observe only the input and the output and make the constraint that the output is the square of the input
    $m_r(P) = 1$ if the output is always correct, 0 otherwise
  - E.g. observe only the execution time and make the constraint on the overall time efficiency
    $m_r(P)$ = program execution time
  - E.g. observe the modular structure of the program code and make a constraint on the size of each component
  - $m_r(P) = \max\{LOC(p): p \text{ is a procedure in } P\}$

Abstract Interpretation

- Abstract interpretation formalizes the conservative approximation of the semantics of computer systems.
- Approximation: observation of the behavior of a computer system at some level of abstraction, ignoring irrelevant details;
- Conservative: the approximation cannot lead to any erroneous conclusion.
- Consider a set of requirements R= FR ∪ NFR and P be a program in the solution space S(R,L)

  Then, Abstract Interpretation of a program P just focuses on {t(I): t is an execution trace of c(P) with initial state in I}

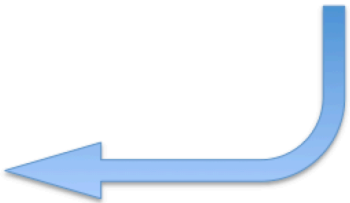
Abstract Interpretation

- By the abstract interpretation theory…
- If the abstract domain A is able to represent a subset S of the requirement set R, having as a target the execution traces,
- and $I^{\#}$ is an overapproximation of I in A,
- and the set of abstract traces $\{t(I^{\#})$: t is a trace of $P^{\#}$ with initial value $I^{\#}\}$ satisfies the requirements in S,
- then P satisfies the requirements in S as well.

Example

```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```

Can we get to an error here?



Example

- Let us consider the possible execution traces

```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```

x=0
x=0, y=?
x=0, y=?, z=1
x=0, y=0, z=1
return 0

a trace

Example

- Let us consider the possible execution traces

```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```

x=0
x=0, y=?
x=0, y=?, z=1
x=0, y=0, z=1
return 0

a trace

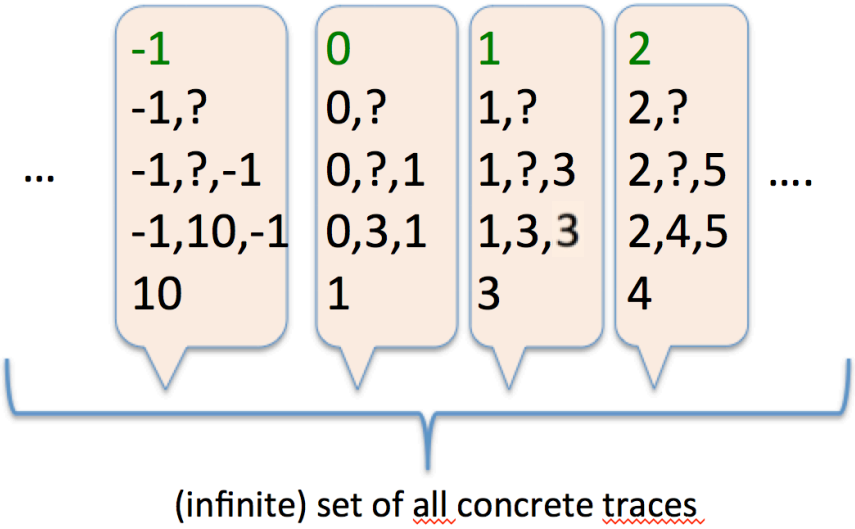
x=1
x=1, y=?
x=1, y=?, z=3
x=1, y=3, z=3
return 3

another trace

Example

- Let us consider the possible execution traces

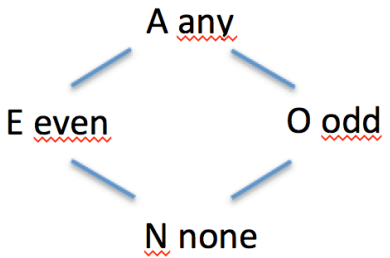
```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```



Example

```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```

Consider the integer parity instead of the concrete integer values

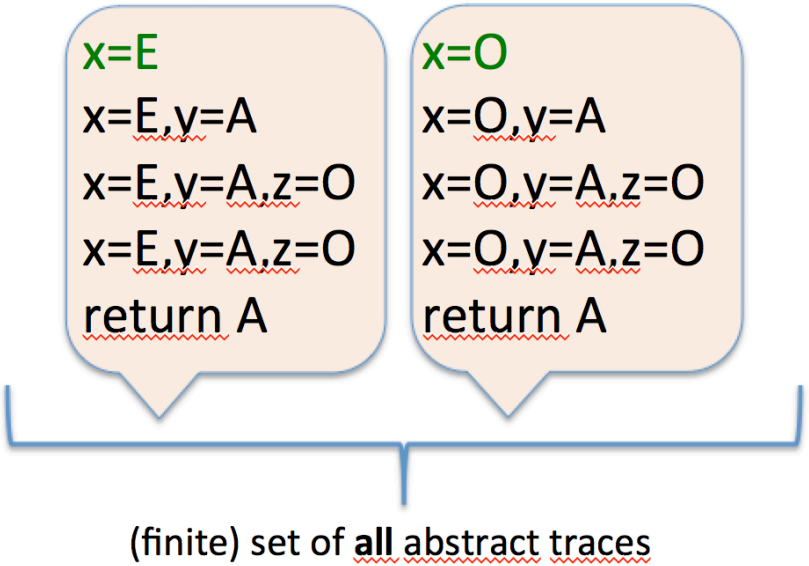


*	A	E	O	N
A	A	E	A	N
E	E	E	E	N
O	A	E	O	N
N	N	N	N	N

+	A	E	O	N
A	A	A	A	N
E	A	E	O	N
O	A	O	E	N
N	N	N	N	N

Example

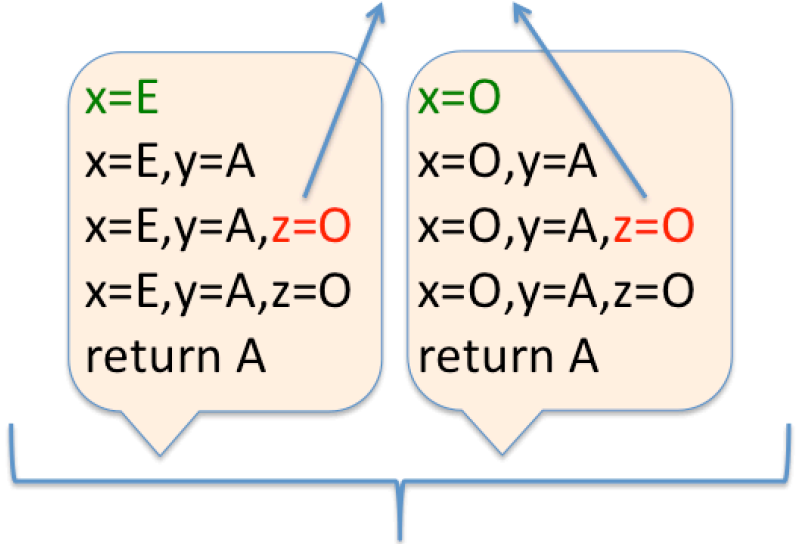
```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```



Example

```
int main(int x){
        int y;
        z = 2*x + 1;
        y= 10*x/z;
        return y;
}
```

variable z has always an odd value: it can never be equal to 0!

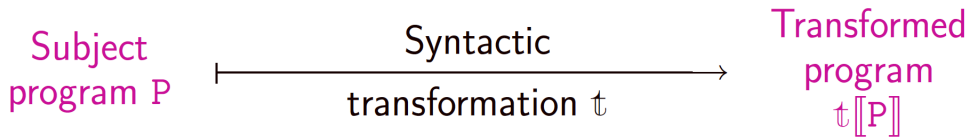


(finite) set of **all** abstract traces

Program Transformations

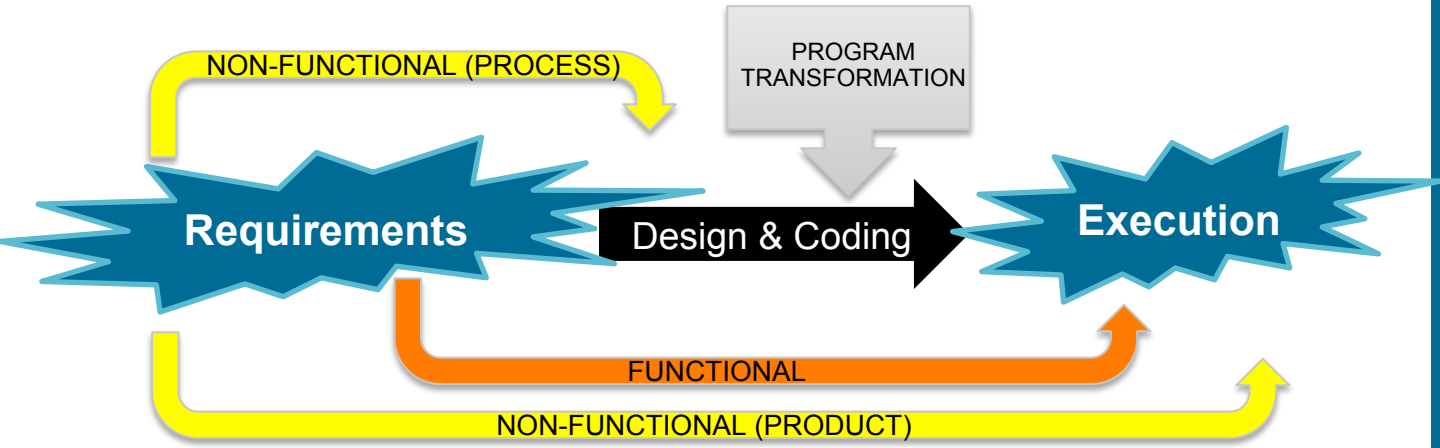
- Focus on syntactic modification of the program code
  - Within the same programming language
  - Translating into another programming language

- Based on transformation rules
- Correctness is usually proved with respect to an equivalence relation (e.g. bisimilarity)



Correctness

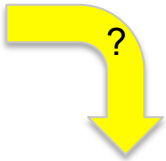
- We may say that a program transformation is **correct** with respect to a requirement set if the resulting process & product is still compliant with respect to the **entire** requirements' set
- Formally, a transformation rule z for L is R-compliant if P is an element of S(R,L) implies z(P) is in S(R,L) too.
- However, the process NFR may also provide constraints on the applyable program transformation techniques



What are the effects on the code?
What are the effects on the exec trace?
What are the effects on the user?

```
public static void main(String[] args) {
    …
    if( num1 >= num2 && num1 >= num3)
        System.out.println(num1+" is the largest Number");
    else if (num2 >= num1 && num2 >= num3)
        System.out.println(num2+" is the largest Number");
    else
        System.out.println(num3+" is the largest Number");
}
```

Syntactic equivalence?
Semantic equivalence?
User independence?

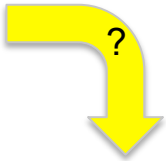



```
public static void main(String[] args) {
    …
    if( kow >= labaad && kow >= terco)
        System.out.println(kow+" waa tirada ugu badan");
    else if (labaad >= kow && labaad >= terco)
        System.out.println(labaad+" waa tirada ugu badan");
    else
        System.out.println(terco+" waa tirada ugu badan");
}
```

What are the effects on the code?
What are the effects on the exec trace?
What are the effects on the user?

```
public static void large(int num1, int num2, int num3) {
    …
    if( num1 >= num2 && num1 >= num3)
       System.out.println(num1+" is the largest Number");
    else if (num2 >= num1 && num2 >= num3)
       System.out.println(num2+" is the largest Number");
    else
       System.out.println(num3+" is the largest Number");
 }
```

247 characters
output in english
the largest number



251 characters
output in somali
the largest number

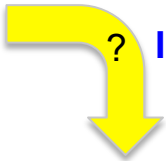
```
public static void weyn(int ow, int labaad, int terco) {
    …
    if( kow >= labaad && kow >= terco)
       System.out.println(kow+" waa tirada ugu badan");
    else if (labaad >= kow && labaad >= terco)
       System.out.println(labaad+" waa tirada ugu badan");
    else
       System.out.println(terco+" waa tirada ugu badan");
 }
```

What are the effects on the code?
What are the effects on the exec trace?
What are the effects on the user?

```
 int add (int x, int y)
{
  return x + y;
}

int sub (int x, int y)
{
  return add (x, -y);
}
```

Does inlining
preserve maintainability?

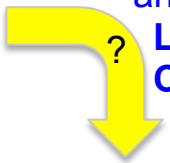



```
 int sub (int x, int y)
{
  return x - y;
}
```

What are the effects on the code?
What are the effects on the exec trace?
What are the effects on the user?

```
int a[100][300];

for (i = 0; i < 300; i++)
  for (j = 0; j < 100; j++)
    a[j][i] = 0;
```



Does loop collapsing keeps the program analysis feasible?

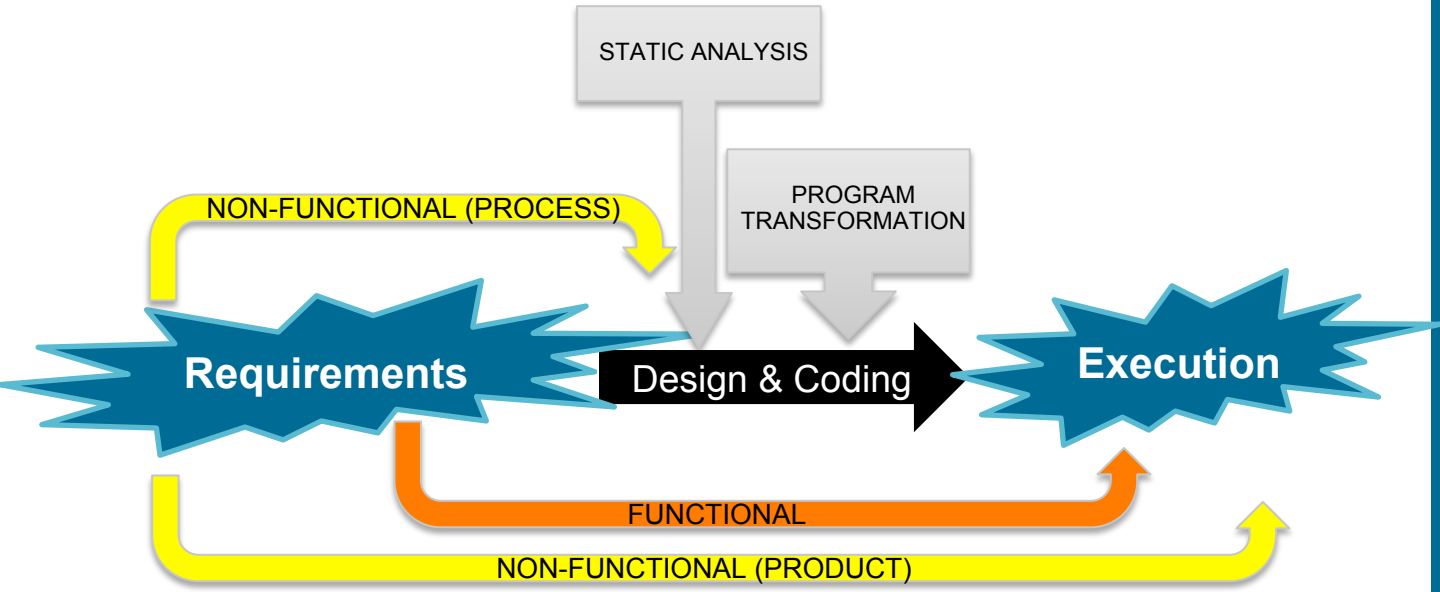
LOOP COLLAPSING

```
int a[100][300];
int *p = &a[0][0];

for (i = 0; i < 30000; i++)
  *p++ = 0;
```

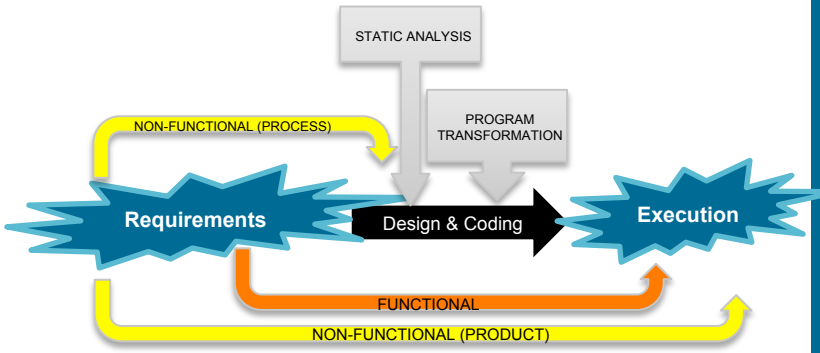
Static Analyses and Program Transformations

- The soundness of program transformation often relies on the result of a static analysis



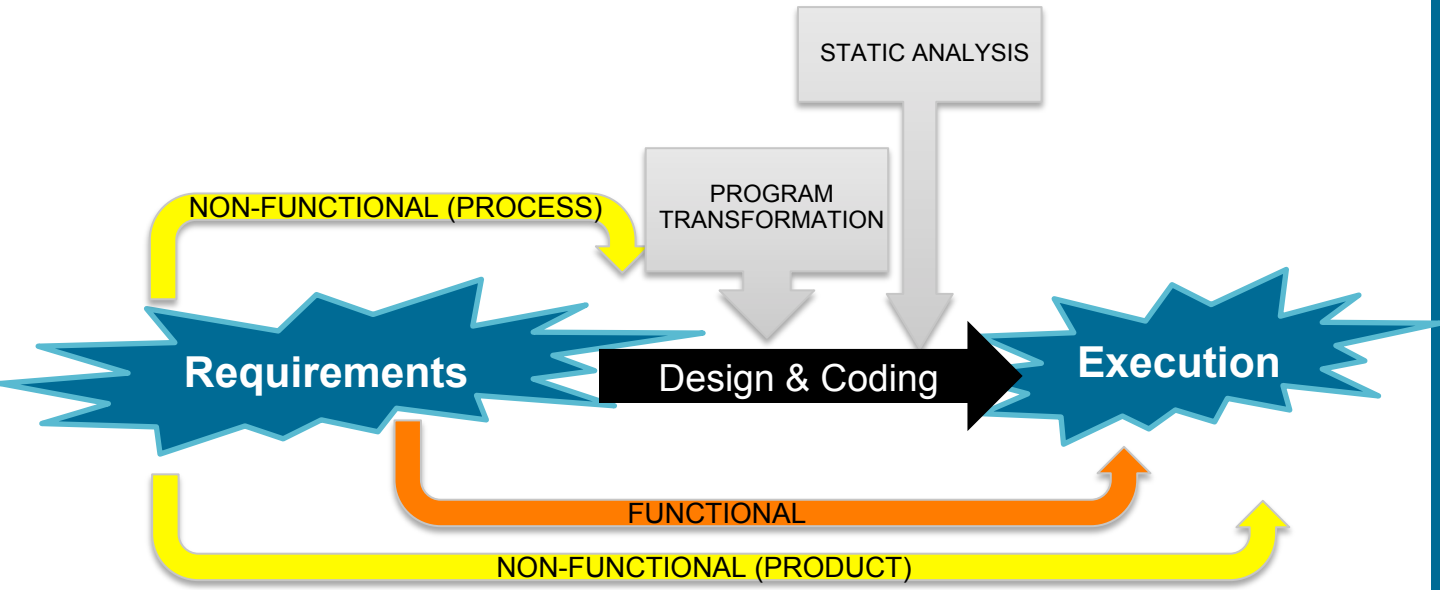
Examples

- Constant propagation relies on reaching definition analysis
- Dead code elimination relies on liveness analysis
- Hoisting relies on very busy expressions analysis
- Common subexpression elimination relies on available expressions analysis
- Unswitching (a loop containing a loop-invariant *if* statement can be transformed into an *if* statement containing two loops) relies on loop invariance analysis
- … the list is much longer…



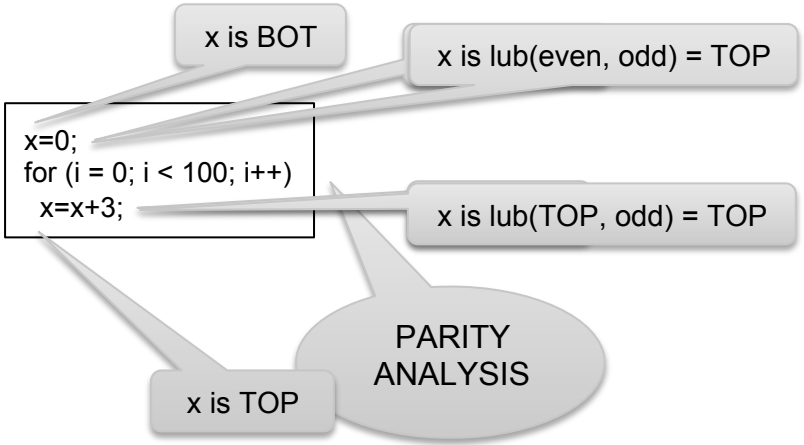
Static Analyses and Program Transformations

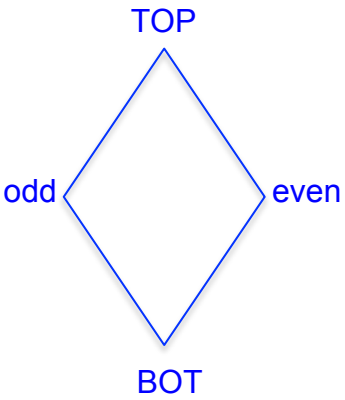
- And static analyses are performed on the resulting code
  - to find bugs & vulnerabilities
  - to support the assessment of requirement satisfaction



Program transformation may help static analysis too

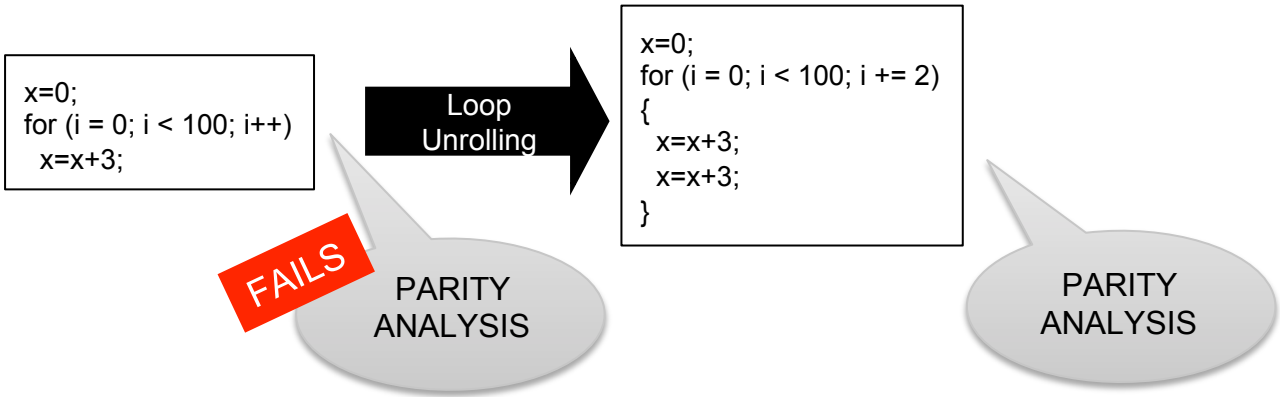
- Requirement: output x is even





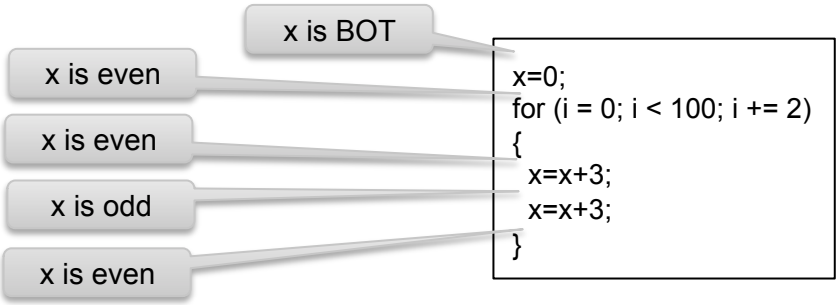
Program transformation may help static analysis too

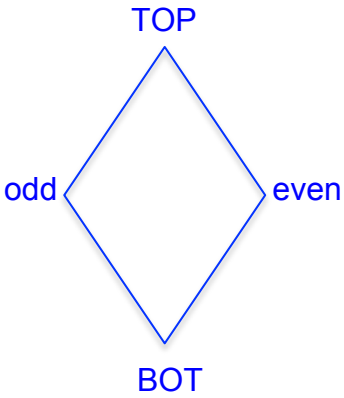
- Requirement: output x is even



Program transformation may help static analysis too

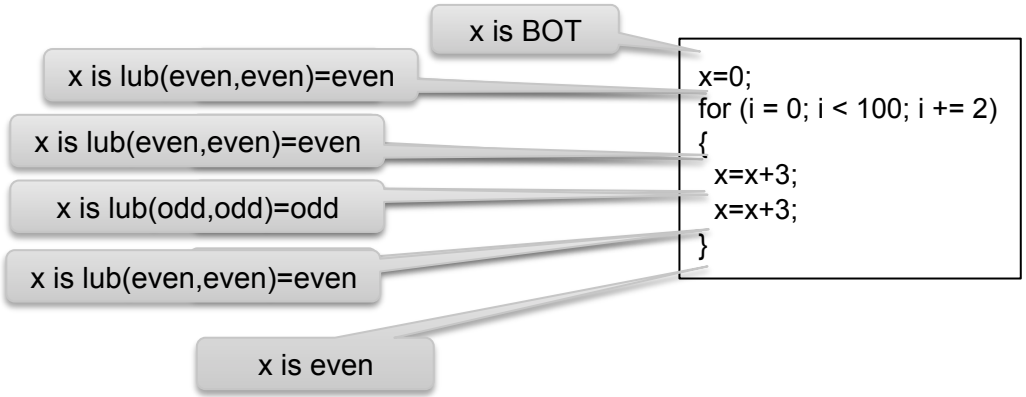
- Requirement: output x is even

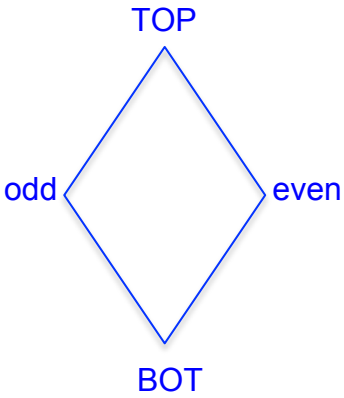




Program transformation may help static analysis too

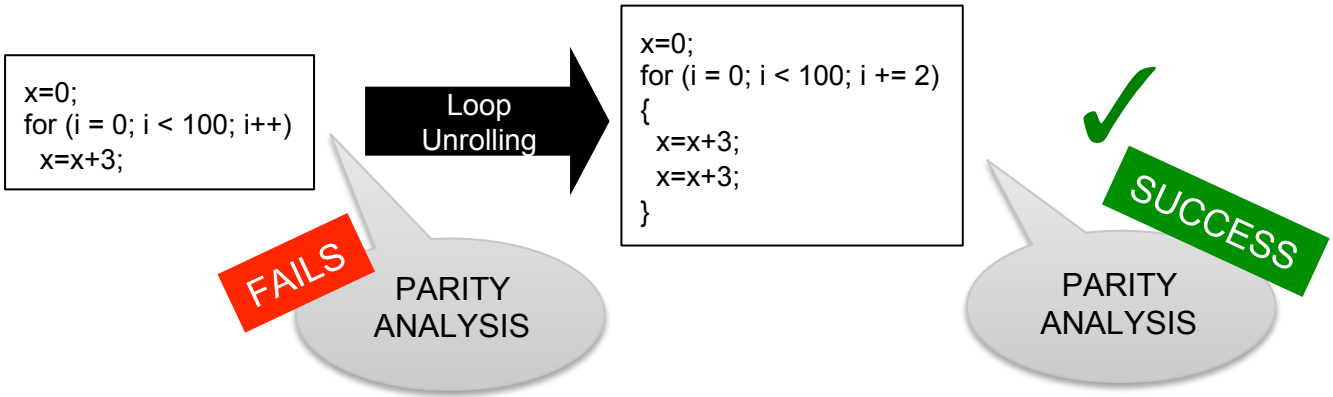
- Requirement: output x is even





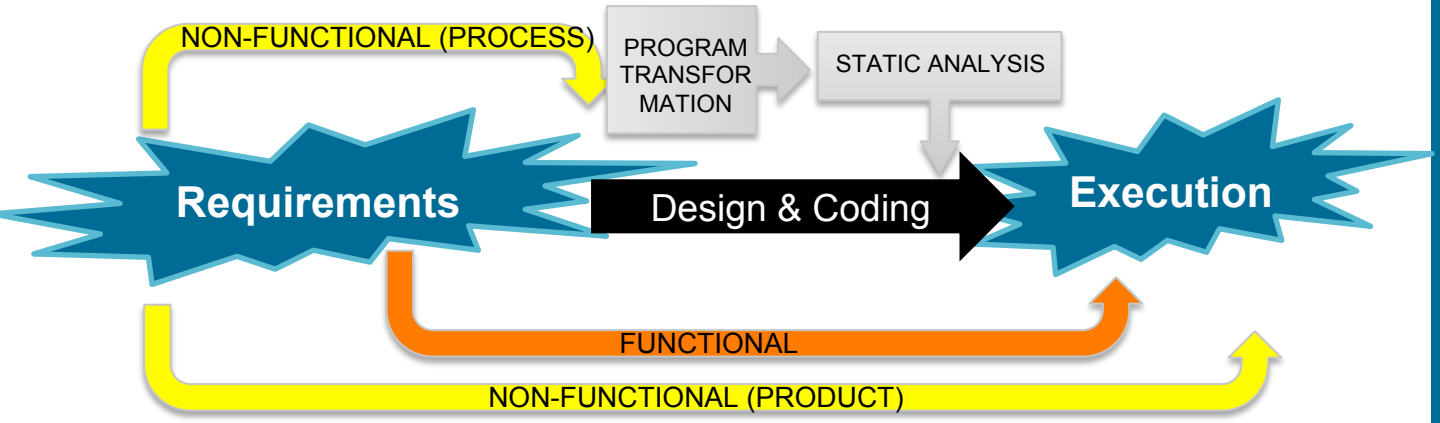
Program transformation may help static analysis too

- Requirement: output x is even



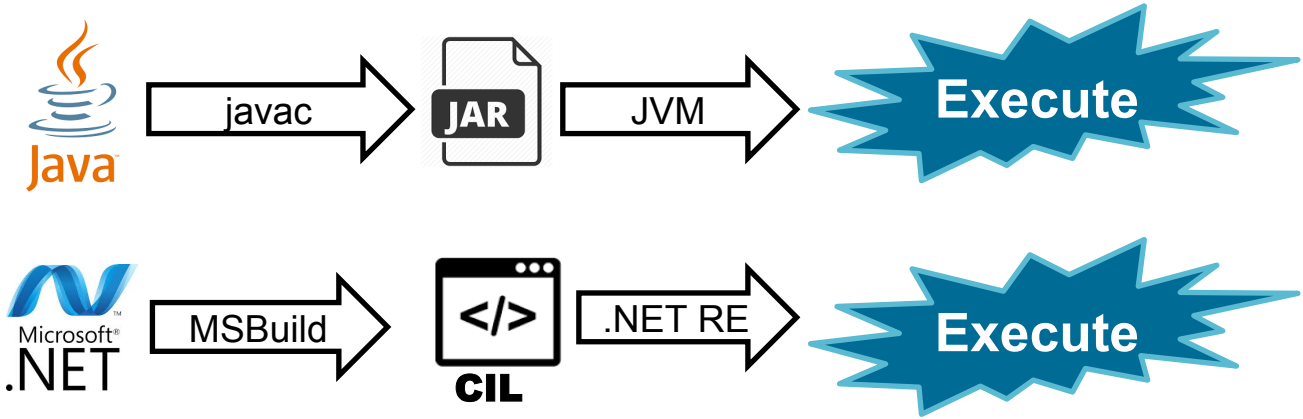
Static Analyses and Program Transformations: a third scenario

- Program transformation can be needed for static analysis purposes only (with no effect on the code to be delivered)

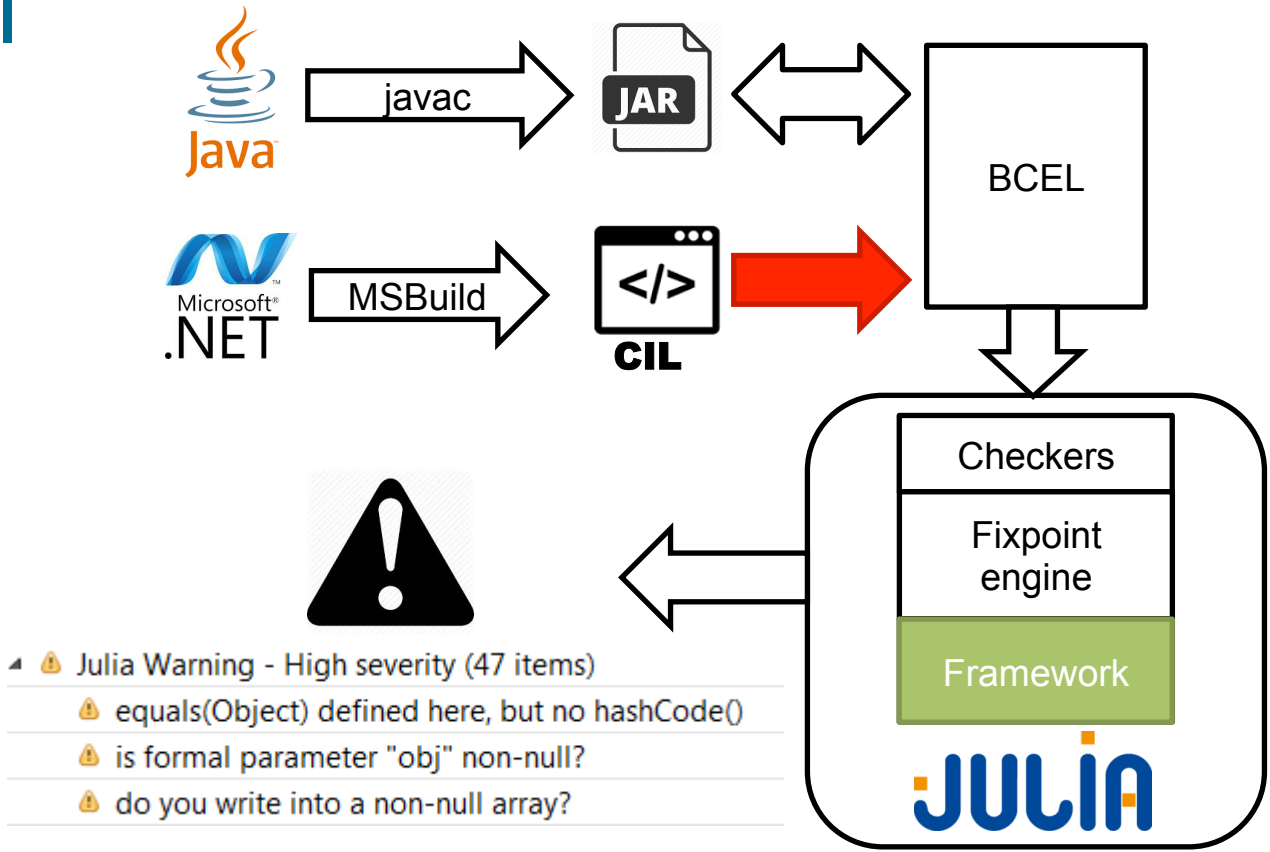


JB and CIL bytecode languages

- Machine-independent low-level languages
- Interpreted or compiled Just-In-Time
- Based on an array of local variables for source code variables, operand stack of temporary values, heap
- Object-oriented

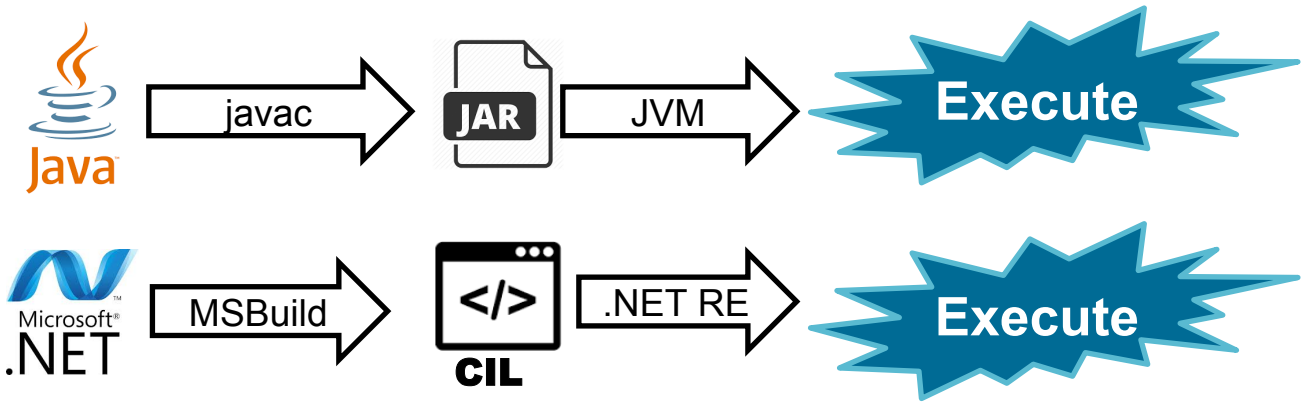


The Julia static analyzer



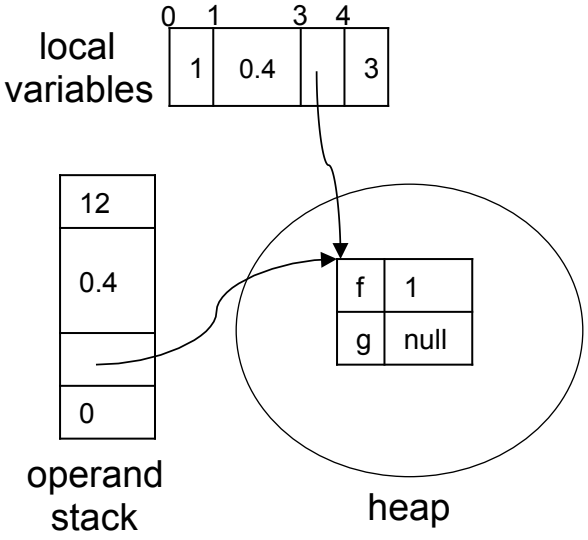
CIL and JB look similar but…

- CIL and JB differ
  - for the way of performing parameter passing
  - for the way they handle object creation
  - for the way the allocate memory slots
  - CIL uses pointers (also in type unsafe ways) while JB has no notion of pointers

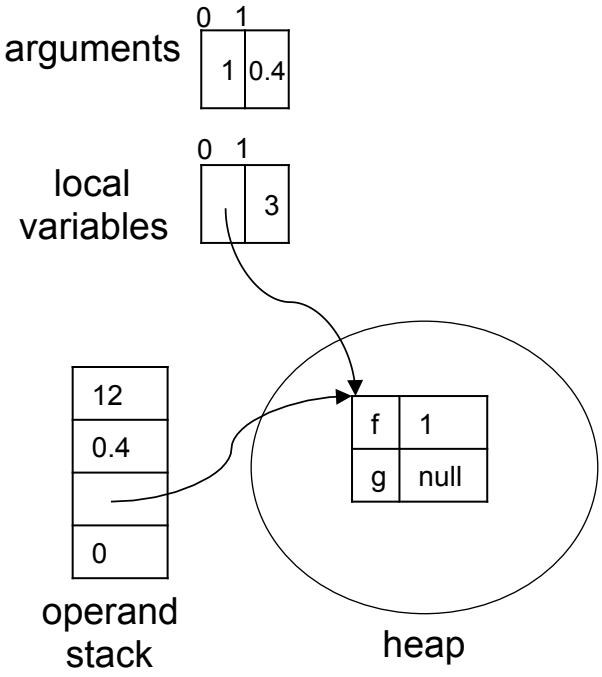


Concrete states

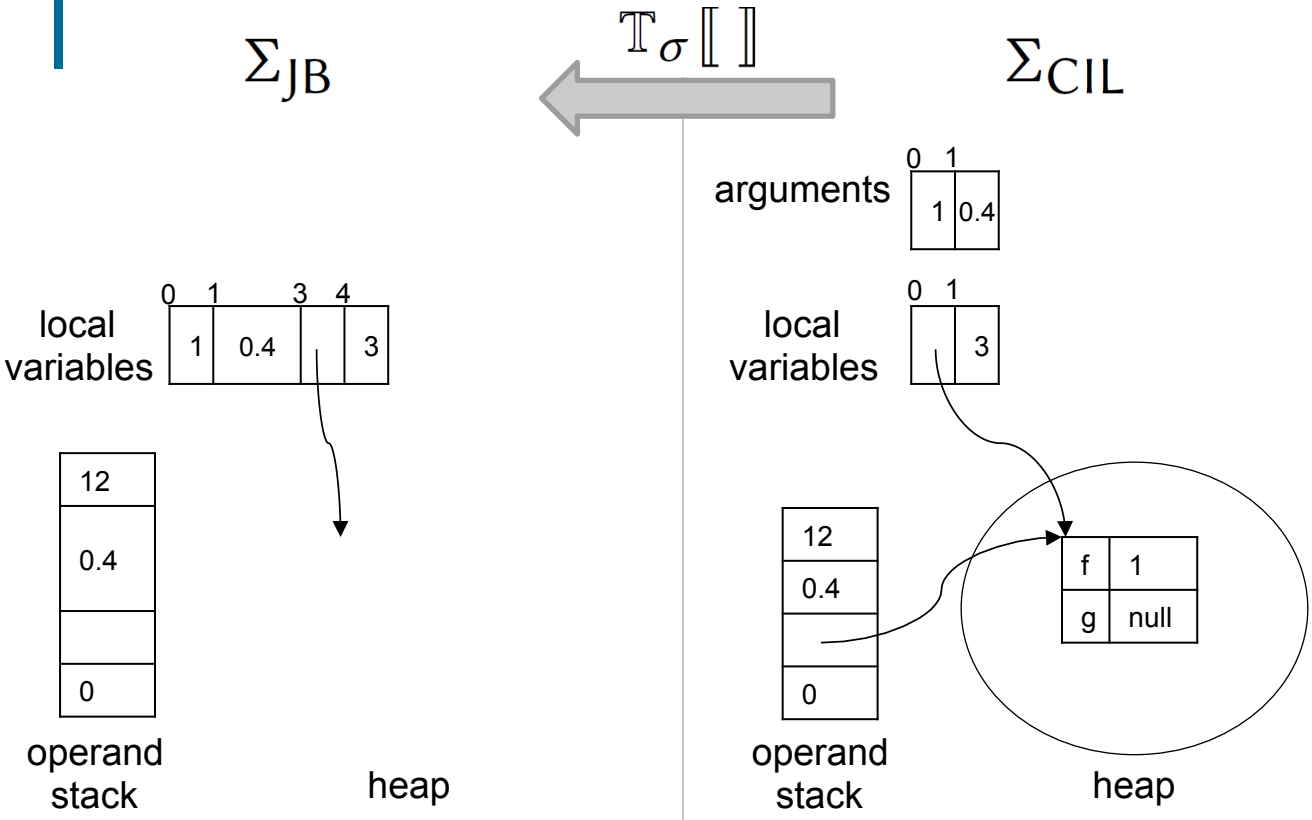
Java Bytecode



MS CIL



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



Java Bytecode (JB) vs. CIL

JB	CIL	
`iadd` `ladd`	`add`	*(arith. op.)*
`iload i` `lload i`	`ldloc i` `stloc i`	*ocal vars)*
		neth. call)
		bjects)
		ond. branch)
`dup` `dup2`	`dup`	*(stack)*

	`ldloca i`
	`stind`
	`ldind`
		(pointers)

- CIL more expressive than JB
- Direct references are simulated by constructing a wrapper object
- We focus on CIL derived from safe C# code only (where direct pointers apply only to ref and out method parameters)

JB: typed
CIL: untyped

JB: local vars
CIL: local vars + args

JB: static and dynamic
CIL: generic

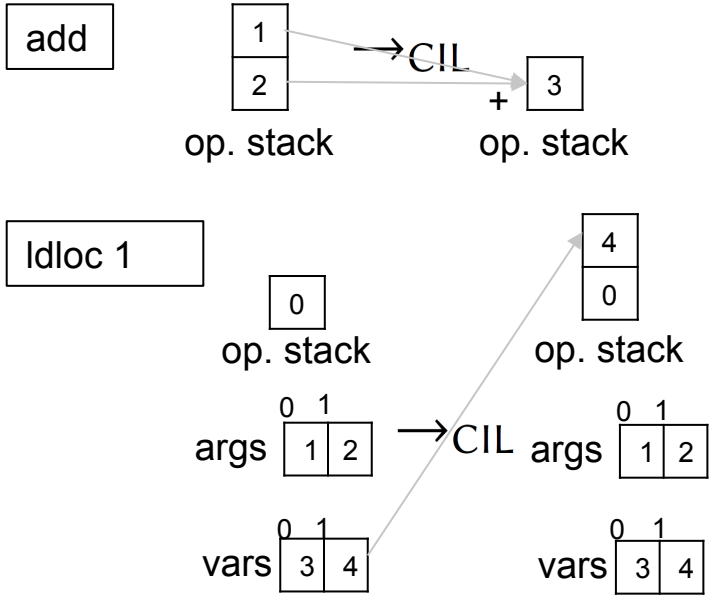
JB: typed
CIL: untyped

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

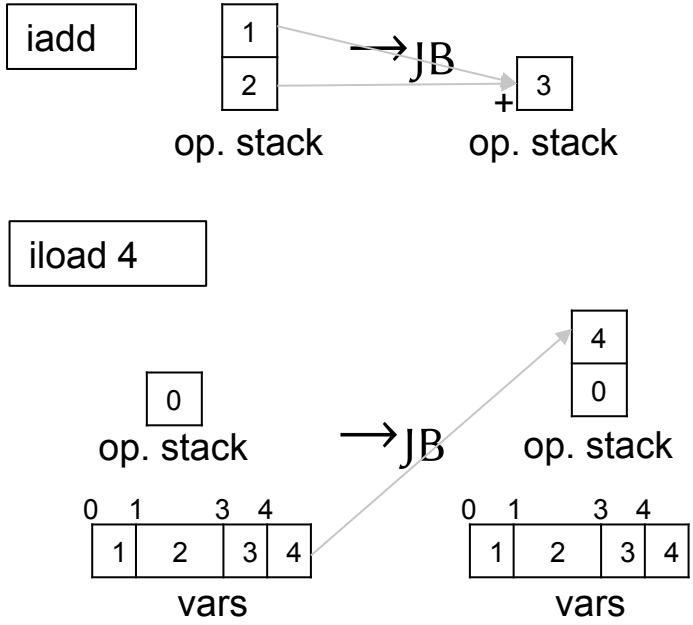
CIL: direct pointers

Concrete semantics

MS CIL



Java Bytecode



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Figure 5: Concrete CIL semantics.

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

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

$$\frac{typeOf(v_1) \neq \text{Long}}{\langle \text{dup2}, (s :: v_1 :: v_2, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: v_1 :: v_2 :: v_1 :: v_2, \text{l}, \text{h})}\ (\text{dup2 32})$$

$$\frac{typeOf(v) = \text{Long}}{\langle \text{dup2}, (s :: v, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: v :: v, \text{l}, \text{h})}\ (\text{dup2 64})$$

$$\frac{typeOf(v_1) = \text{Int} \wedge typeOf(v_2) = \text{Int}}{\langle \text{iadd}, (s :: v_1 :: v_2, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: (v_1 + v_2), \text{l}, \text{h})}\ (\text{iadd})$$

$$\frac{typeOf(v_1) = \text{Long} \wedge typeOf(v_2) = \text{Long}}{\langle \text{ladd}, (s :: v_1 :: v_2, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: (v_1 + v_2), \text{l}, \text{h})}\ (\text{ladd})$$

$$\frac{x = \mathcal{JVM}prefix(typeOf(\text{l}(\text{i})))}{\langle x\text{load i}, (s, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: \text{l}(\text{i}), \text{l}, \text{h})}\ (x\text{load})$$

$$\frac{x = \mathcal{JVM}prefix(typeOf(\text{v}))}{\langle x\text{store i}, (s :: \text{v}, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s, \text{l}[\text{i} \mapsto \text{v}], \text{h})}\ (x\text{store})$$

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

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

$$\frac{fresh(\text{T}, \text{h}) = (r, \text{h}')}{\langle \text{new T}, (s, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: r, \text{l}, \text{h}')}\ (\text{new})$$

$$\frac{o \neq \text{null}}{\langle \text{getfield f}, (s :: o, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s :: \text{h}(o)(\text{f}), \text{l}, \text{h})}\ (\text{getfield})$$

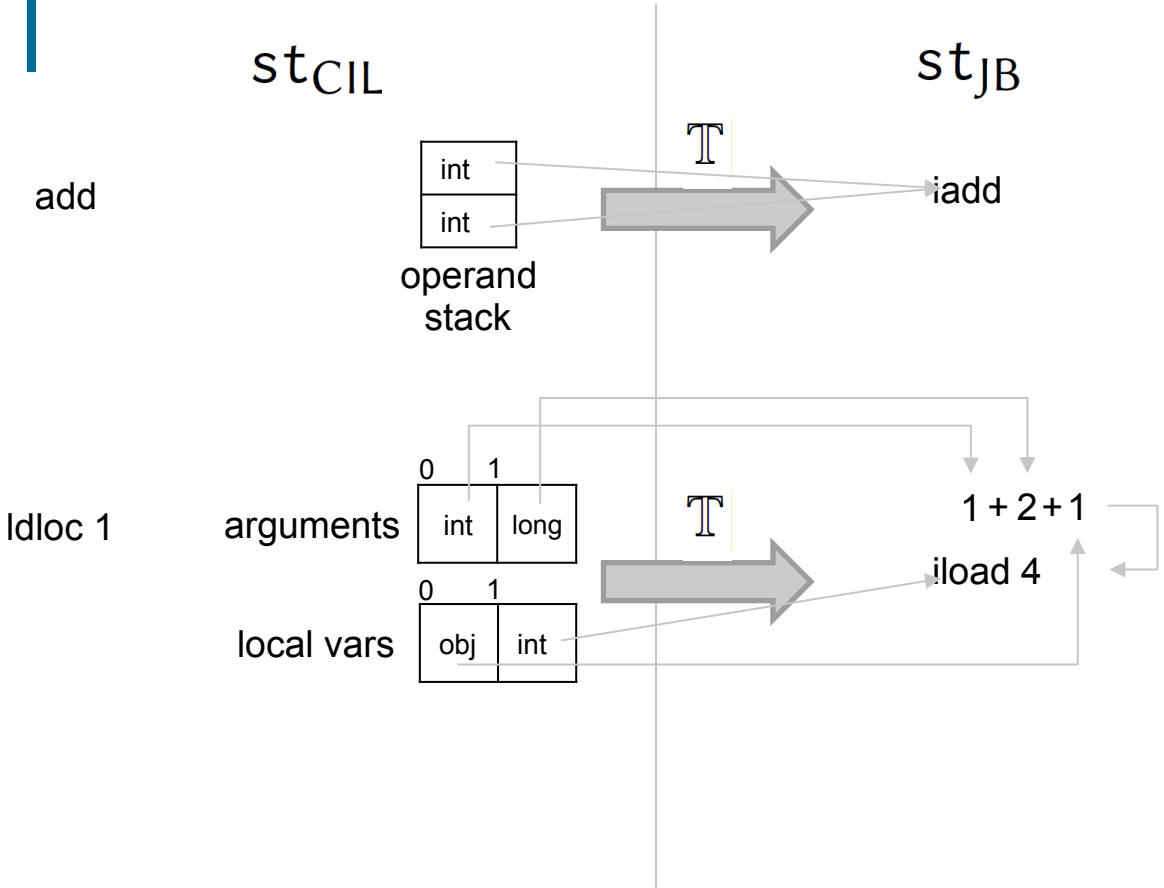
$$\frac{o \neq \text{null} \quad s' = \text{h}(o)[\text{f} \mapsto v]}{\langle \text{putfield f}, (s :: o :: v, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} (s, \text{l}, \text{h}[o \mapsto s'])}\ (\text{putfield})$$

$$\frac{typeOf(v_1) = \text{Int} \wedge typeOf(v_2) = \text{Int} \wedge v_1 > v_2}{\langle \text{if\_icmpgt l}, (s :: v_1 :: v_2, \text{l}, \text{h})\rangle \rightarrow_{\text{JB}} \langle \text{l}, (s, \text{l}, \text{h})\rangle}\ \begin{pmatrix}\text{if\_icmpgt} \\ \text{true}\end{pmatrix}$$

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

Figure 6: Concrete JB semantics.

Statement translation $\mathbb{T}[\![st_{CIL}, K]\!] = st_{JB}$



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

$$\mathbb{T}[\![\mathtt{dup}, \overline{\mathtt{s}} :: \mathtt{t}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \begin{cases} \mathtt{dup} & \text{if } t \neq \mathsf{Long} \\ \mathtt{dup2} & \text{if } t = \mathsf{Long} \end{cases}$$

$$\mathbb{T}[\![\mathtt{add}, \overline{\mathtt{s}} :: \mathtt{t}_1 :: \mathtt{t}_2, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \begin{cases} \mathtt{iadd} & \text{if } t_1 = t_2 = \mathsf{Int} \\ \mathtt{ladd} & \text{if } t_1 = t_2 = \mathsf{Long} \end{cases}$$

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

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

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

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

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

$$\mathbb{T}[\![\mathtt{ldfld\ f}, \overline{\mathtt{s}} :: \mathtt{t}_o, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \mathtt{getfield\ f}$$

$$\mathbb{T}[\![\mathtt{stfld\ f}, \overline{\mathtt{s}} :: \mathtt{t}_o :: \mathtt{t}_v, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \mathtt{putfield\ f}$$

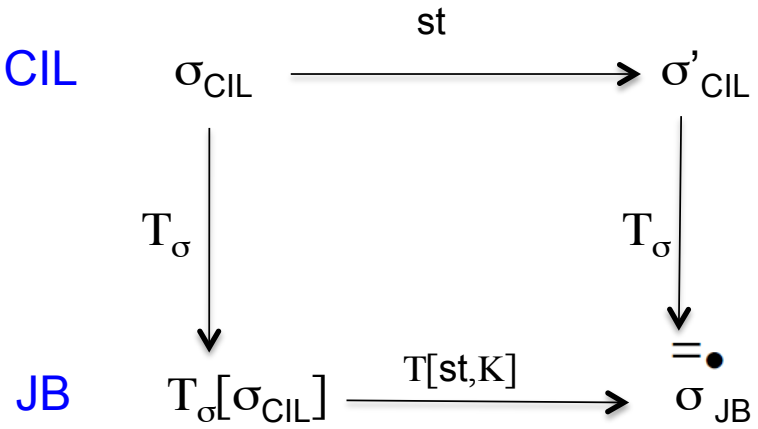
$$\mathbb{T}[\![\mathtt{bgt\ k}, \overline{\mathtt{s}} :: \mathtt{t}_1 :: \mathtt{t}_2, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \mathtt{if\_icmpgt\ k'} \text{ where } \mathtt{k'} = \mathit{statementIdx}(\mathit{getBody}(\mathtt{bgt\ k})(\mathtt{k})) \text{ if } t_1 =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

$$\mathbb{T}[\![\mathtt{ldloca\ i}, \overline{\mathtt{s}}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \mathbb{T}[\![\mathtt{newobj\ WrapRef()}\ ;\ \mathtt{dup2}\ ;\ \mathtt{stloc\ j}; \mathtt{ldloc\ i}; \mathtt{stfld\ value}, \overline{\mathtt{s}}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!]$$
$$\text{where } j = \mathit{freshIdx}(\mathtt{ldloca\ i}, 0)$$

$$\mathbb{T}[\![\mathtt{stind}, \overline{\mathtt{s}}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \mathbb{T}[\![\mathtt{stfld\ value}, \overline{\mathtt{s}}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!]$$

$$\mathbb{T}[\![\mathtt{ldind}, \overline{\mathtt{s}}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!] = \mathbb{T}[\![\mathtt{ldfld\ value}, \overline{\mathtt{s}}, \overline{\mathtt{l}}, \overline{\mathtt{a}}, \overline{\mathtt{w}}]\!]$$

Correctness of the translation



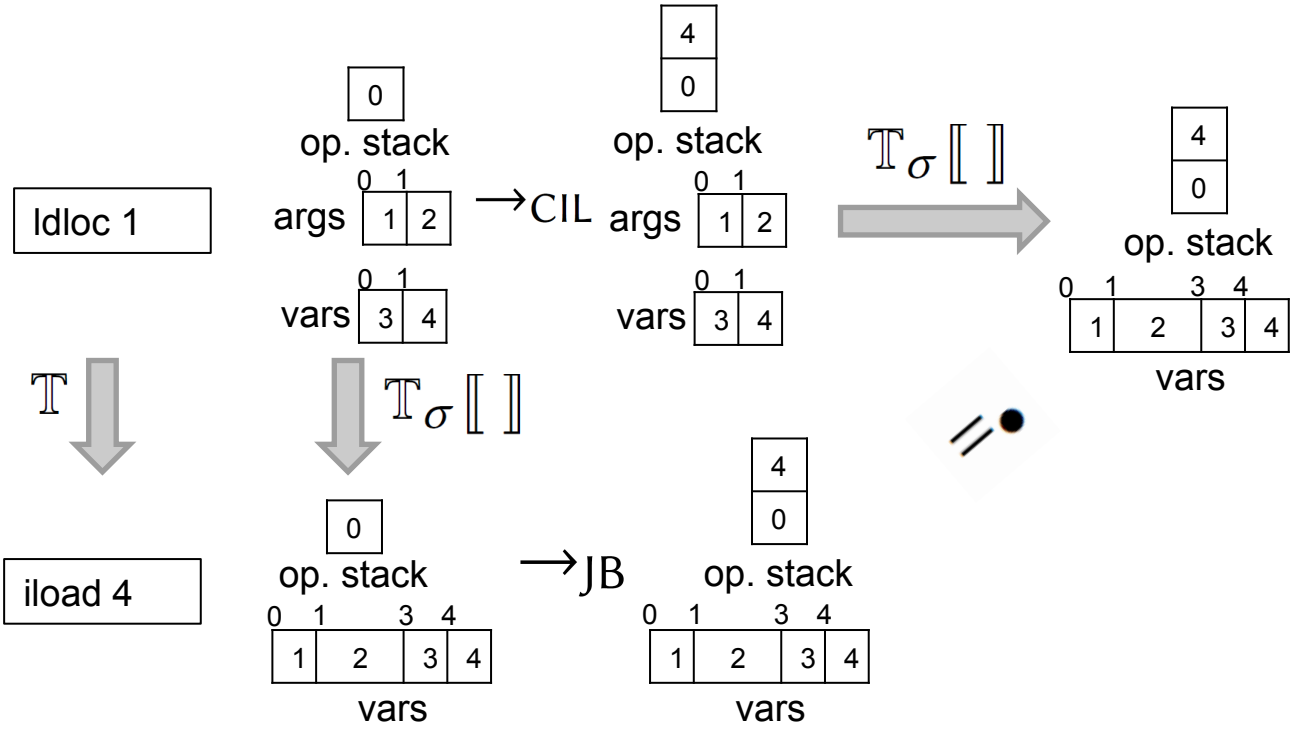
$=_\bullet$ means equal up to instrumentation variables introduced in the translation process

Correctness

$$\forall st \in St_{CIL}, \sigma_{CIL} \in \Sigma_{CIL}: \quad \langle st, \sigma_{CIL} \rangle \rightarrow_{CIL} \sigma'_{CIL} \text{ and } \langle \mathbb{T}[\![st, K]\!], \mathbb{T}_\sigma[\![\sigma_{CIL}]\!] \rangle \rightarrow_{JB} \sigma'_{JB}$$

$$\Downarrow$$

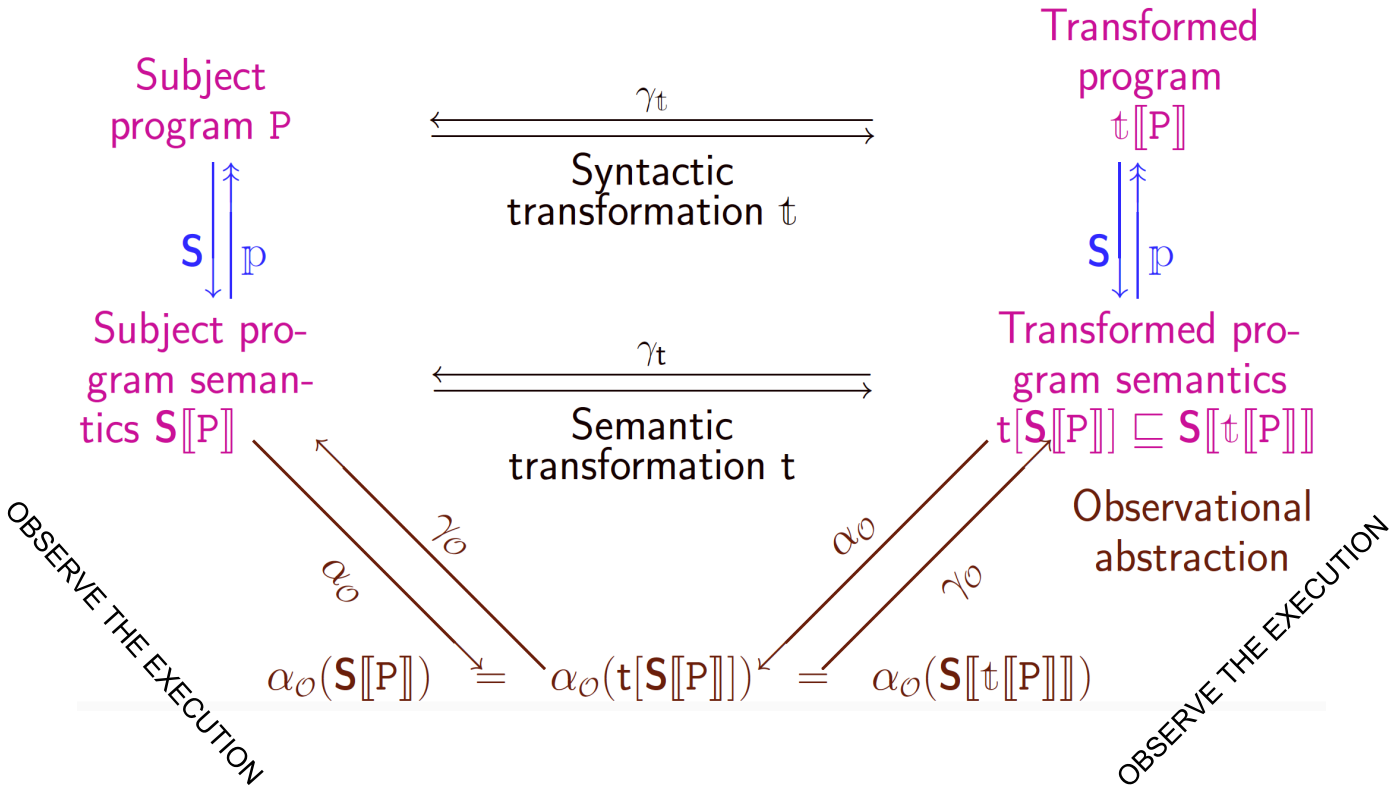
$$\mathbb{T}_\sigma[\![\sigma'_{CIL}]\!] =_\bullet \sigma'_{JB}$$



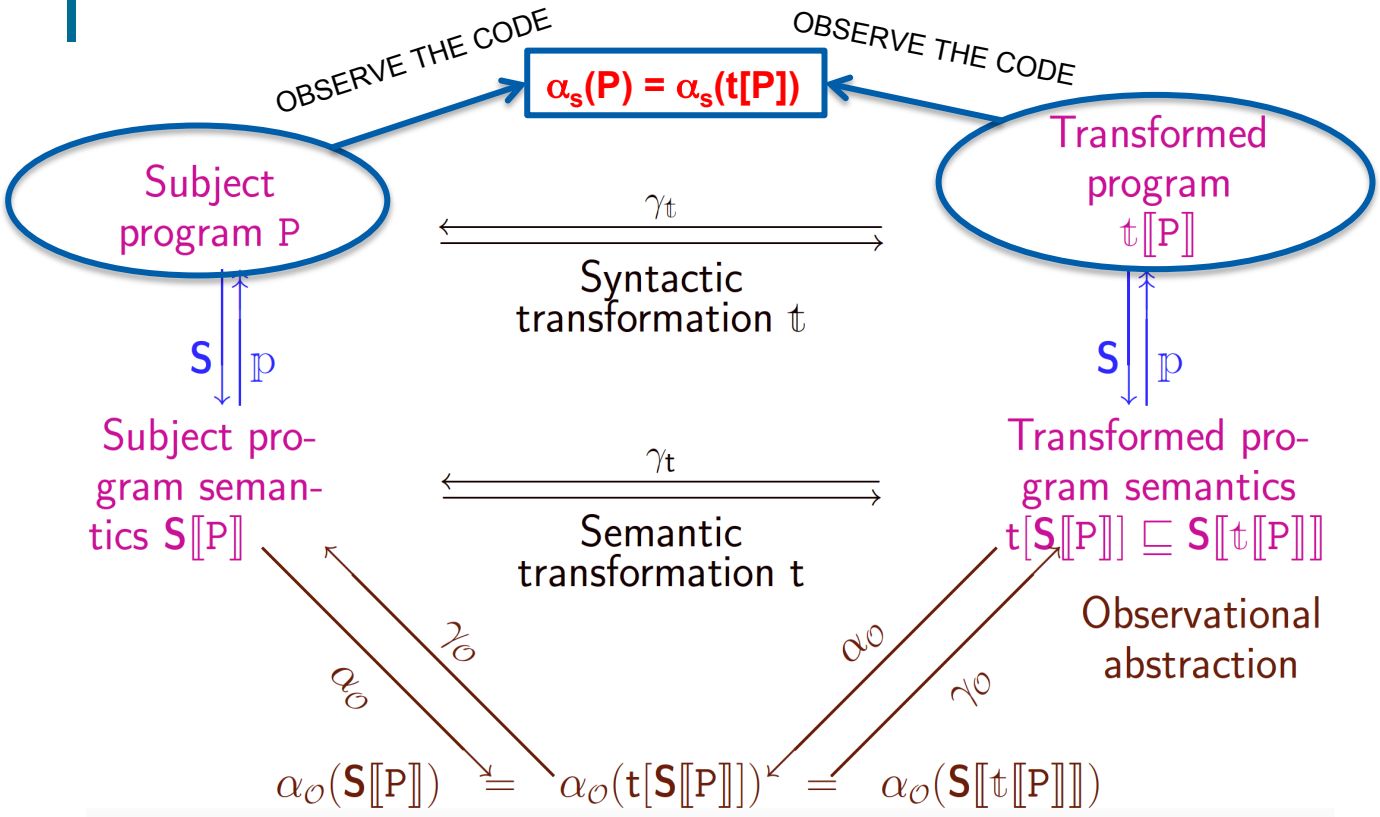
What we learned from the CIL to JB transformation…

- The translation from CIL to JN scales well:
  - Translating all the .NET libraries (500K methods, about 5MLOCs) took about 24 minute, i.e. 4 methods per milliseconds. It required at most 238 MB of RAM
  - On 5 large GitHub projects (more than 250 KLOCs) the analysis took 9 minutes, with only 40 sec. consumed for the translation from CIL to JB
- On the GitHub projects analysed the analysis reported about 2K critical or major warnings. Only 4% of them are false alarms due to the CIL to JB translation
- About libraries: the translation succeeds to translate 99,4% of their methods (only unsafe methods cannot be translated)
- The translation may also be applied to let Java and C# code interoperate, by compiling both of them in JB
- Which properties of the code are preserved by the transformation rules?

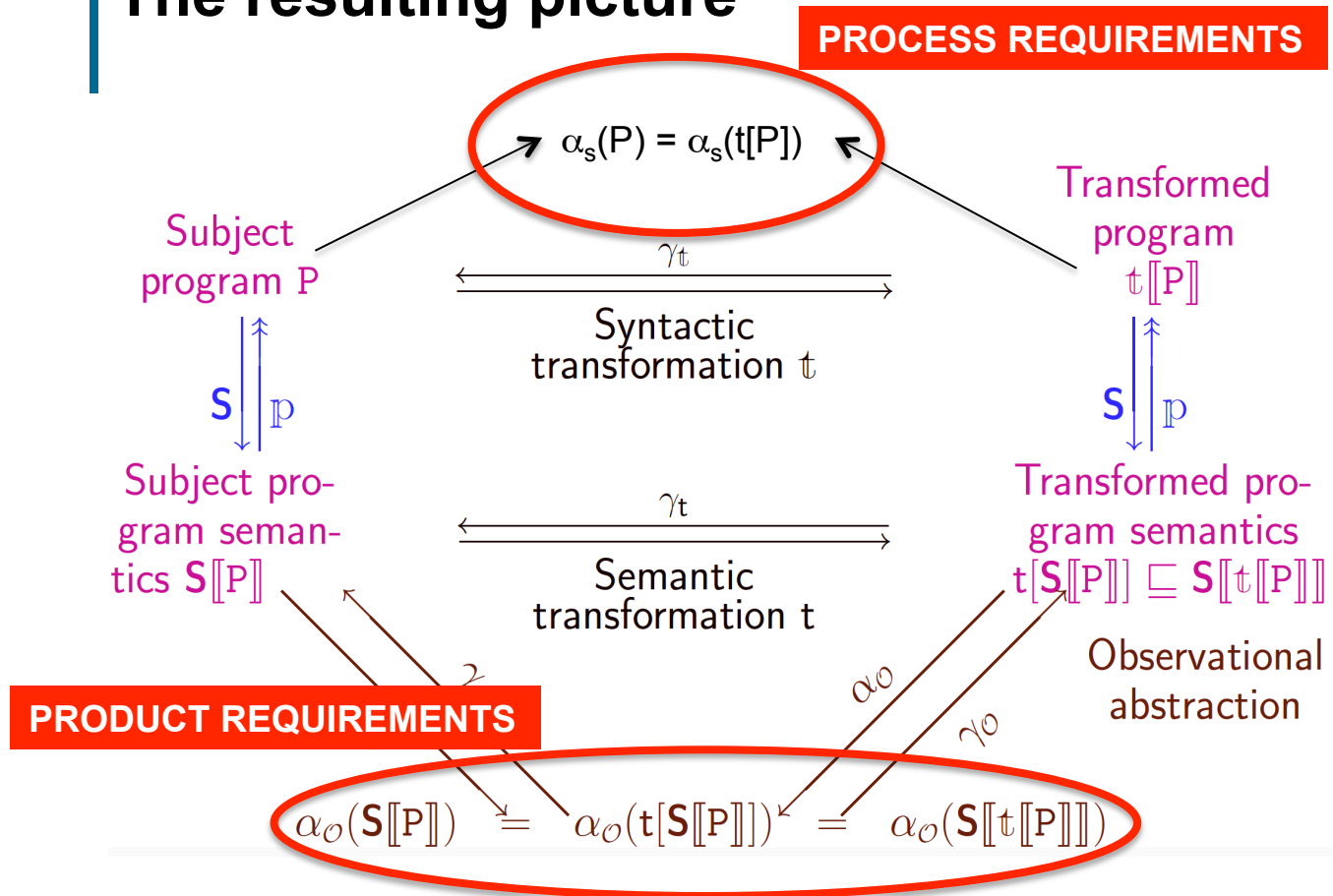
Trasformations as abstractions: the Cousot '02 overall vision



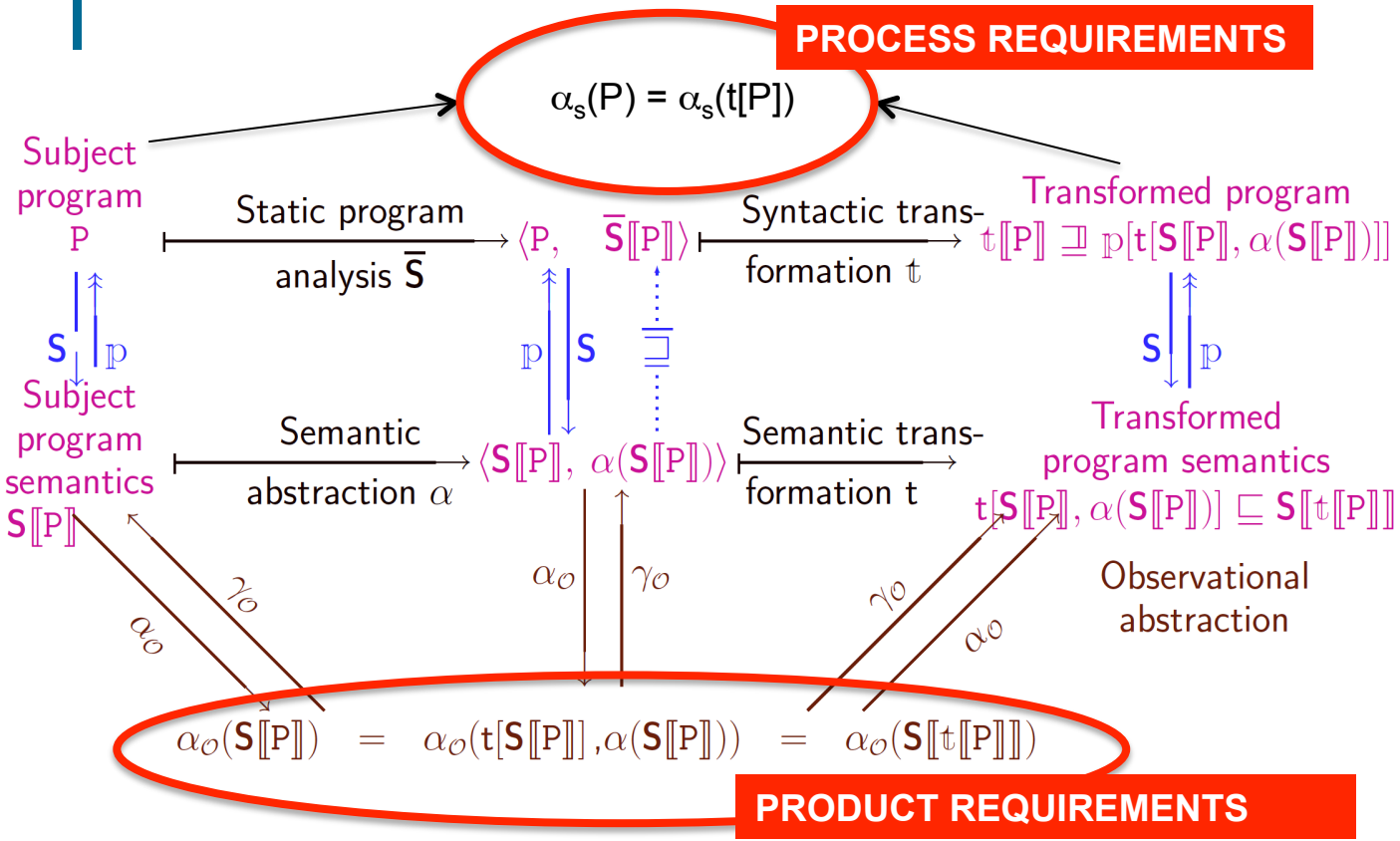
What was missing there?



The resulting picture



Offline program transformation



Conclusions

- Program Transformation and Abstract interpretation interact in different ways
- The key actions are just: observe, abstract and verify
- Correctness should refer to the whole Requirement set
- There are still many challenging issues that wait for us!

Thanks!

Tino Cortesi, cortesi@unive.it

Our last publications - Ca' Foscari SSV

- M.Roy, N.Deb, A.Cortesi, R.Chaki, N.Chaki. "NFR-Aware Prioritization of Software Requirements". Journal of Systems Engineering, Wiley, in print (2021)
- L.Negrini, V.Arceri, P.Ferrara and A.Cortesi: "Twinning automata and regular expressions for string static analysis". Proc. VMCAI'21 - 22nd Intern.Conf. on Verification, Model Checking, and Abstract Interpretation, Springer LNCS (2021)
- Ferrara, P., Mandal, A.K., Cortesi, A., Spoto, F: "Static analysis for discovering IoT vulnerabilities". International Journal on Software Tools for Technology Transfer, Springer (2021)
- P.Ferrara, A.Cortesi, F.Spoto: "From CIL to Java bytecode: Semantics-based translation for static analysis leveraging". Sci. Comput. Program. Vol. 191: 102392, Elsevier (2020)
- A.Jana, R.Halder, K.V.Abhishekh, S.Devi Ganni, A.Cortesi: "Extending Abstract Interpretation to Dependency Analysis of Database Applications". IEEE Transactions on Software Engineering. Vol. 46(5): 463-494 (2020)
- A.K. Mandal, F.Panarotto, A.Cortesi, P.Ferrara, F.Spoto: "Static analysis of Android Auto infotainment and on-board diagnostics II apps". Software Practice and Experience. Vol. 49(7): 1131-1161 (2019)