Applied Static Analysis 2016

Dr. Michael Eichberg (Organizer)
Johannes Lerch, Ben Hermann, Sebastian Proksch, Karim Ali Ph.D.

Software Technology Group

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Exam Like Questions

Static Analysis

- Is an analysis for Java code that does not analyze the called native methods subject to (a) false positives and/or (b) false negatives w.r.t. the Java code?

  In both case shortly explain your answer.

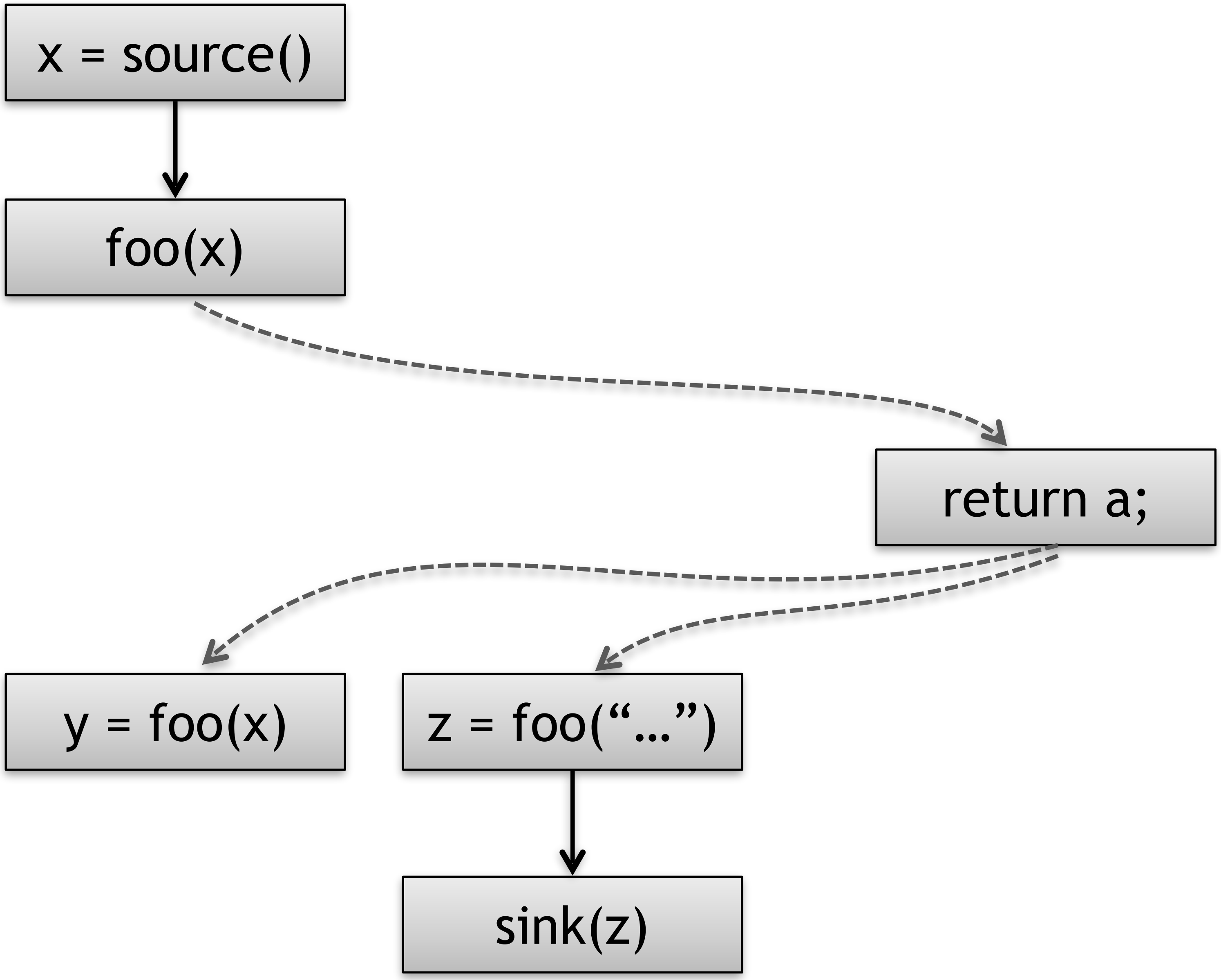
Static Analysis - Transfer Question

- Do you think that it is possible to design a programming language in such a way that it is no longer necessary to have additional static analyses to check for specific issues?
  (E.g., using Java it is no longer possible to produce buffer overflows, using other language it is possible to avoid injection bugs…)

Assume we want to define a context-sensitive taint analysis using the context-free language reachability problem. We already computed the data-flow graph shown below for the given code example.

```
main() {
	x = source();
	if(unknown()) {
		y = foo(x);
	}
	else {
		z = foo(“const”);
	}
	sink(z);
}

foo(a) {
	return a;
}
```



x = source()

foo(x)

return a;

y = foo(x)

z = foo(“…”)

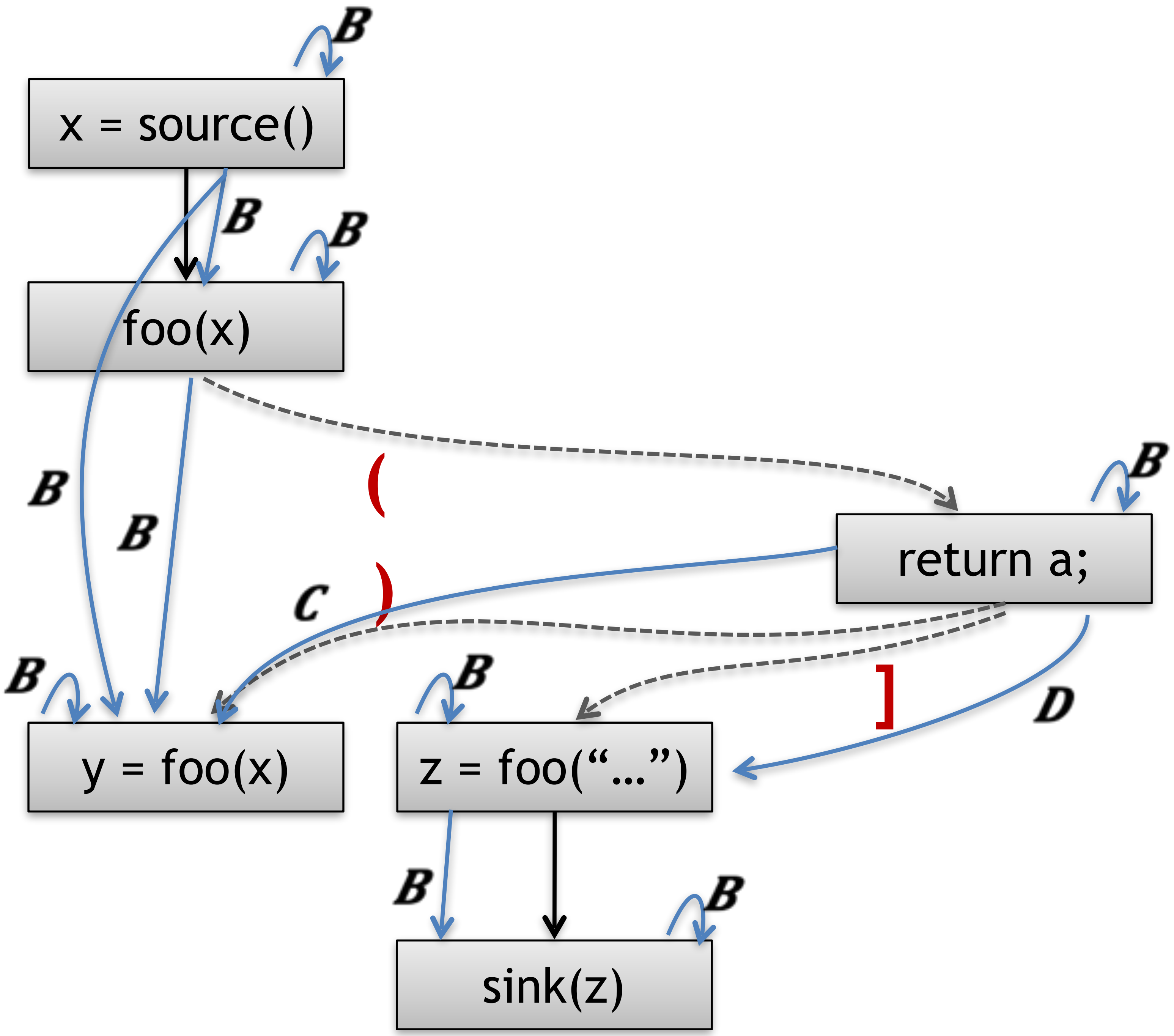
sink(z)

- Task 1)
  Define a context-free language that describes paths that are valid w.r.t. to calling contexts, i.e., the results of the analysis should be context sensitive.
- Task 2)
  Place labels on the edges of the given data-flow graph using terminals of the previously defined context-free language.
- Task 3)
  Solve the context-free language reachability problem by performing the steps of the respective algorithm. Moreover, normalize the previously defined context-free language and draw edges in the given graph that will be computed by the algorithm. Include labels for each computed edge in your illustration.

$B \to (B) \mid [B] \mid BB \mid \epsilon$ ➔ $B \to (C \mid [D \mid BB \mid \epsilon$

$C \to B)$

$D \to B]$



x = source()

foo(x)

return a;

y = foo(x)

z = foo("…")

sink(z)

State for each assumption (CPA, OPA) if for the following code snippet an interface-based call-by-signature edge has to be introduced when *Expression.eval()* is called somewhere in the library.

Justify your answer.

```
public interface Expression { int eval(); }

abstract class ExprNode {
    ExprNode left, right;
    public ExprNode(ExprNode left, ExprNode right){
        this.left = left;
        this.right = right;
    }
    public abstract int eval();
}

class AddNode extends ExprNode {
    public AddNode(ExprNode left, ExprNode right){ super(left, rig
}
    public int eval() { return left.eval() + right.eval(); }
}
```

Answer

- OPA:
- Yes, a CBS-Edge has to be introduced. A developer could create a new Subclass that inherits from the AddNode class and implements the Expression interface. If the subclass does not override the public eval method AddNode.eval() would be called.
- CPA:
- No, a CBS-Edge don’t has to be introduced. AddNode belongs to the library private implementation, hence, there is no way for an application to create a subtype such that eval() a CBS edge has to be introduced.

Name at least 3 differences between Java Bytecode and LLVM IR

- LLVM IR has an alloc operation
- LLVM IR is more flexibly typed (e.g., i8, i32)
- Java Bytecode is not optimized by the Java compiler but by the runtime, LLVM IR is optimized during compilation from C code

- **Describe two typical challenges when writing a static analysis in the context of RSSE that are usually non-existent in traditional application of static analyses, like security or bug detection.**