Etienne Kneuss, Viktor Kuncak and Philippe Suter
Laboratory of Automated Reasonning and Analysis
ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

INSANE: Combine, Reduce, and Conquer

Overview

Insane combines a **pointer analysis** with a **memory effect analysis** for the Scala programming language. The analysis is based on abstract interpretation, is interprocedural and flow sensitive and does not require manual annotations. It targets **higher order programs**, computing **compositional summaries** of methods using an **expressive representation** for heap effects. Generating reusable summaries of higher order functions is difficult because part of the function's effect is encapsulated in its arguments. The techniques generalises to functions taking objects as argument and calling their virtual methods.

Example

```
case class El(var left: Int, var right: Int);

@WillNotModify("lst.tail*.head.right")
def resetLeft(lst: List[El]) {
  lst.foreach { v => v.left = 0; }
}
@MayOnlyModify("l.head.right")
def resetRight(lst: List[El]) {
  lst.foreach { v => v.right = 0;  }
}
```

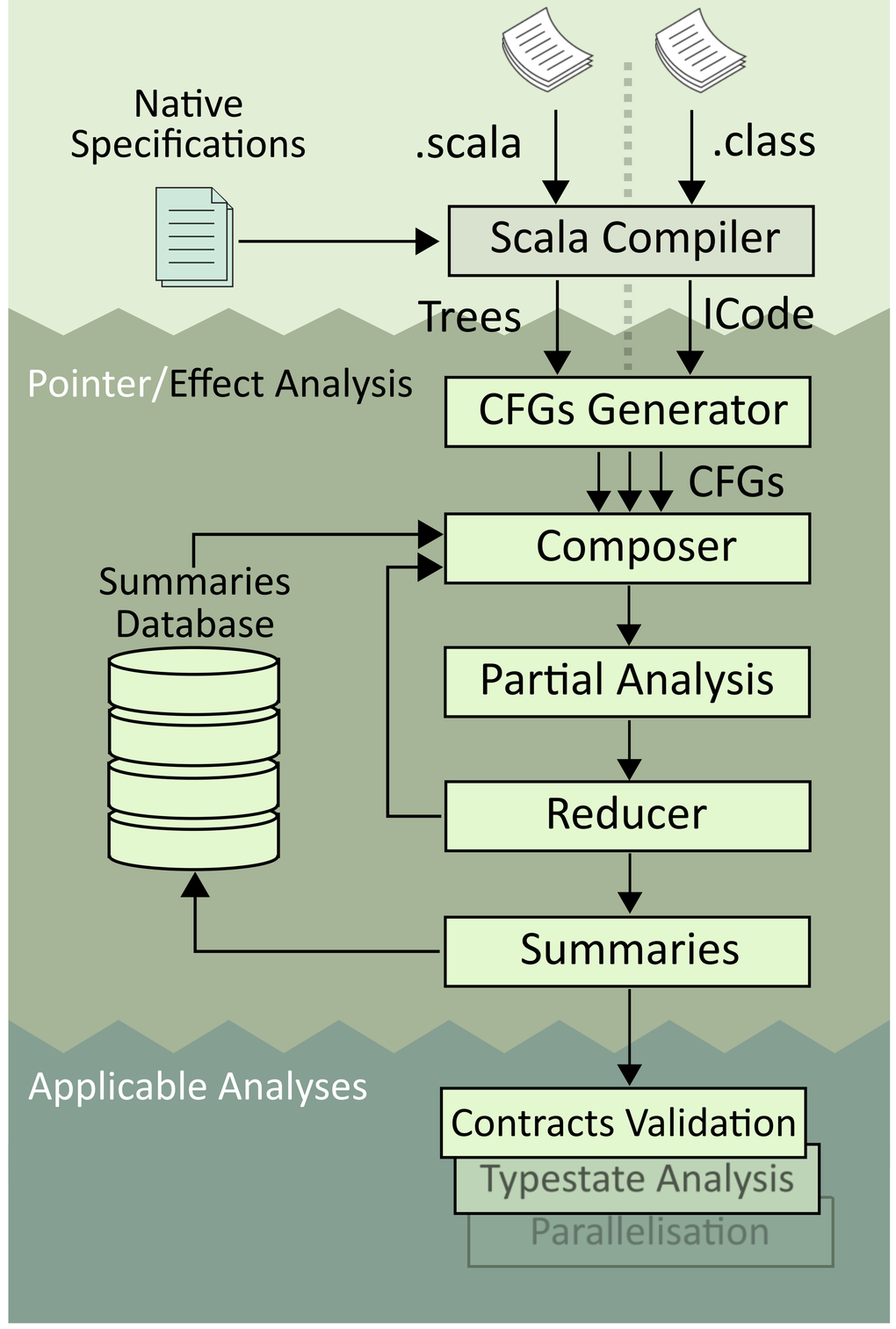
Verified

Not Verified

When assertions fail, Insane can provide a complete counter example:

Effect obtained: **l.tail*.head.right**
Counter example: **l.tail.tail*.head.right**

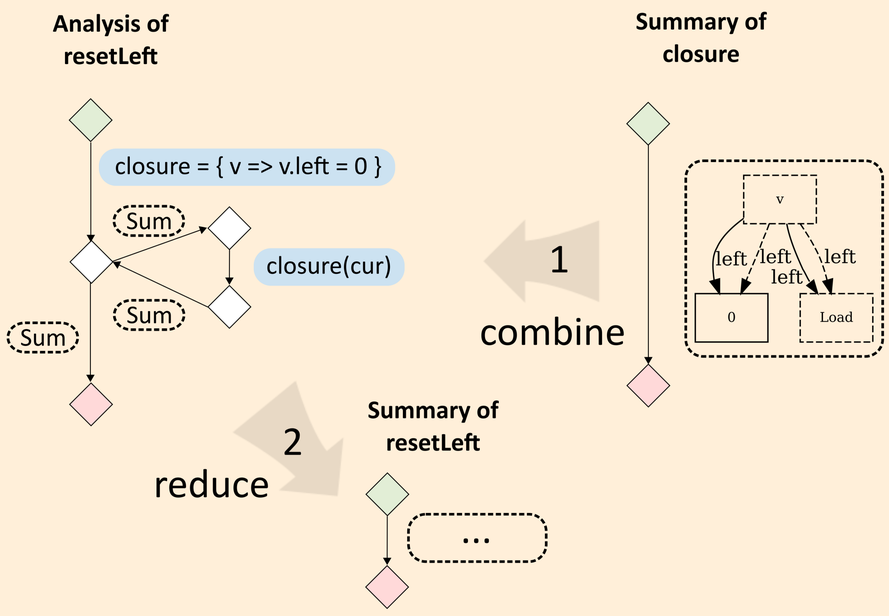
Architecture



Flexible Representation

We start with CFGs on statements from our language, extended with a **summary statement**. We also define a **composition operator** over summary statements, allowing us to combine two consecutive summary statements into one. For method calls, we replace the method call by the summaries of its potential targets. Calls to closures are **delayed** until sufficiently precise.

We use **partially reduced** CFGs as summaries, containing only summary statements or imprecise method calls.

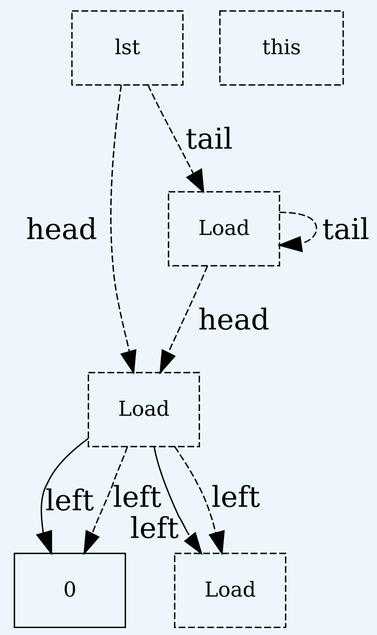


Pointer & Effect Analysis

The values of the abstract domain are represented as graphs that describe abstract heap transformers.

- **Nodes** represent **objects**
  - Dashed nodes represent unknown objects
  - Solid nodes represent allocated objects
- **Edges** are labelled with fields, and represent **effects**
  - Dashed edges represent reads
  - Full edges represent writes

The summary pictured beside is the computed effect of **resetLeft**. It represents a weak update of the field **a** in every element of the (unknown) list passed as argument.



Applications

Static program analysis techniques working on object-oriented languages require precise knowledge of the aliasing relation between variables. Being precise and modular, our analysis has many applications:

- Checking non-interference of effects for
  - program understanding
  - program verification
  - automated parallelization
- Inference of effect annotations for Scala
- Typestate analysis for stateful protocols