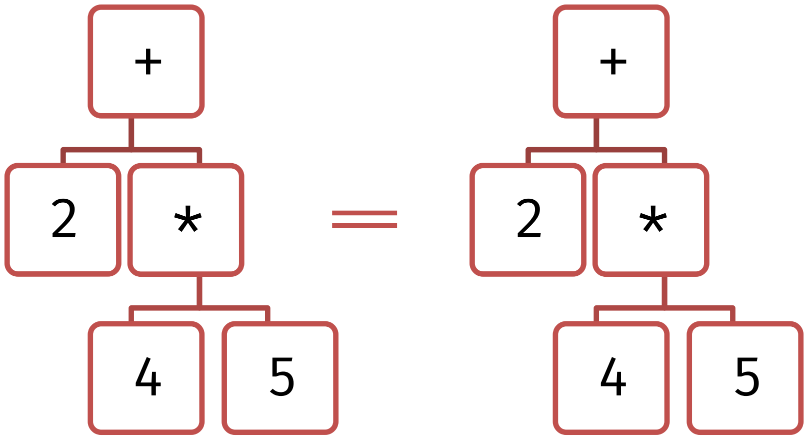
FLYWEIGHT ASTs:
A Study in Applied Lazyness

Fabian Bösiger
Supervised by Dr. Malte Schwerhoff

AST EQUALITY CHECKS



- Are these subtrees structurally equal?
- Also have to check children for equaltiy
- Structural equality checks happen recursively

AST EQUALITY CHECKS

- Can‘t avoid equality checks
- ... but we can make them faster!

```
relevantChunks.sortWith((ch1, ch2
) ⇒ {
    // args is of type Seq[Term]
    // ... &&
    ch1.args == args
})
```

https://github.com/viperproject/silicon/blob/78ff67514b907a2ceb31427fa457ad4ceeac175b/src/main/scala/rules/MoreCompleteExhaleSupporter.scala

“

The verifier should enable an IDE-like experience: it should be sufficiently fast such that users can con tinuously work on verifying programs [...]

Malte Schwerhoff,
Advancing Automated, Permission-Based Program Verification Using Symbolic Execution

”

FLYWEIGHT PATTERN

- If a term (subtree) is created, first check if a structurally equal object already exists
- Only create new instance if no structurally equal instance exists
- Else, return reference to existing object
- Introduces boilerplate code for all of the nearly 100 terms

```
class Plus private (left: Term, right: Term) ext
ends Term {
    // ...
}

object Plus {
    var pool = new HashMap[(Term, Term), Plus];
    def apply(left: Term, right: Term): Plus = {
        pool.get((e0, e1)) match {
            case Some(term) ⇒ term
            case None ⇒
                val term = new Plus(e0, e1)
                pool.addOne((e0, e1), term)
                term
        }
    }
}
```

MACRO ANNOTATIONS

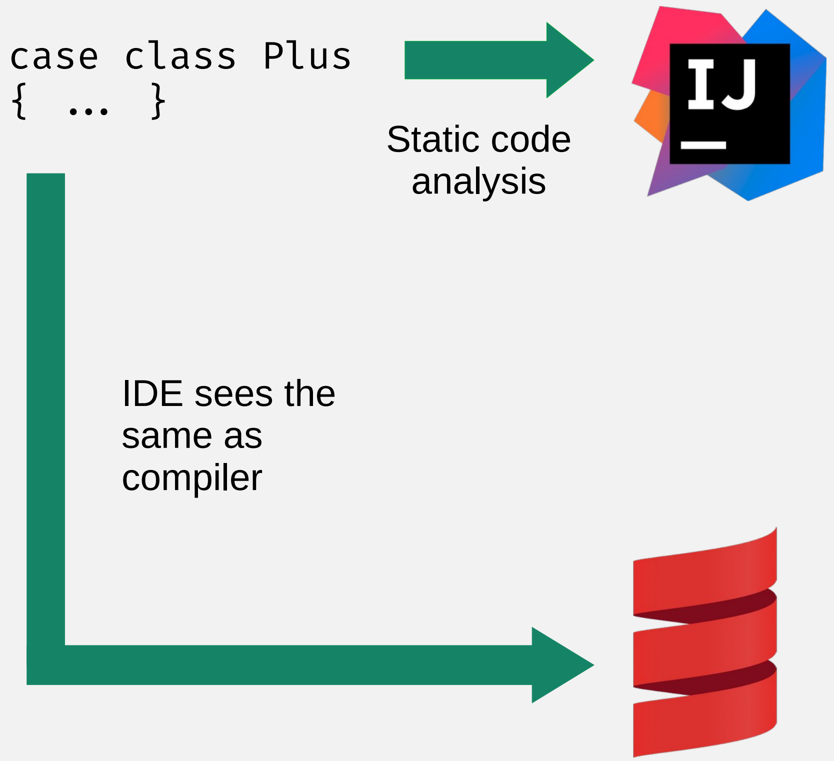
```
@memoizing
case class Plus private (left: Term, right: Term)
extends Term {
    // ...
}
```

Use macro annotations to genererate boilerplate code

```
class Plus private (left: Term, right: Term) ext
ends Term {
    // ...
}

object Plus {
    var pool = new HashMap[(Term, Term), Plus];
    def apply(left: Term, right: Term): Plus = {
        pool.get((e0, e1)) match {
            case Some(term) ⇒ term
            case None ⇒
                val term = new Plus(e0, e1)
                pool.addOne((e0, e1), term)
                term
        }
    }
}
```

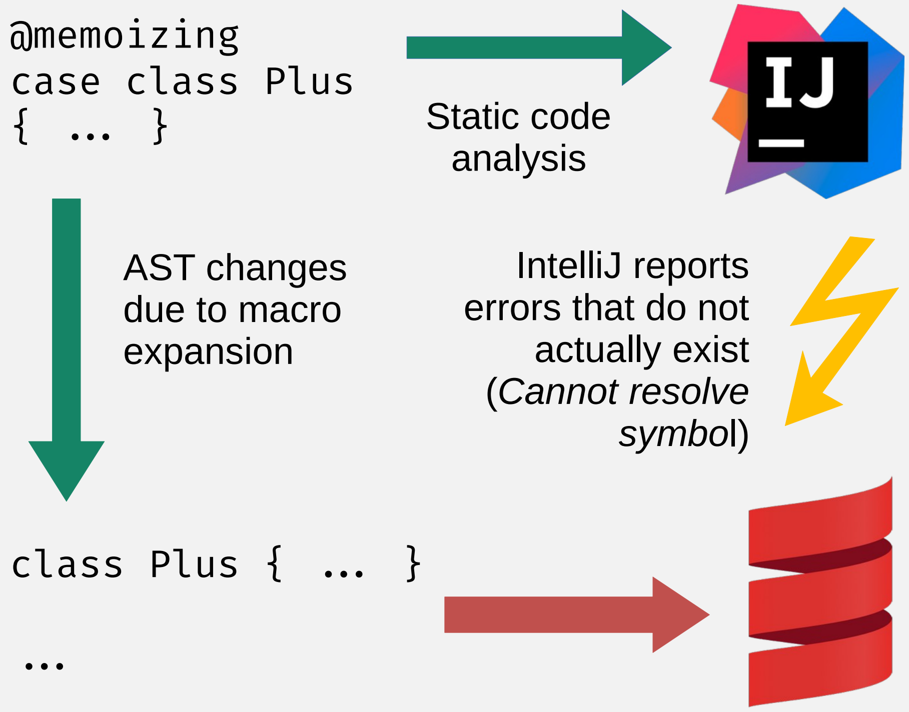
IDE INTEGRATION



```
case class Plus
{  …  }
```

Static code analysis

IDE sees the same as compiler



```
@memoizing
case class Plus
{  …  }
```

Static code analysis

AST changes due to macro expansion

IntelliJ reports errors that do not actually exist (*Cannot resolve symbol*)

```
class Plus {  …  }

…
```

CORE GOALS

- Research other possible solutions for similar problems
- Implement proposed solution approach
- Evaluate performance gains
- Build macro annotations
- Ensure IDE support for macro annotations

EXTENSION GOALS

- Profiling parts of Silicon that perform many operations on terms
  - Some datastructures for example may perform better over others with fast equality checks
- Apply same approach using flyweight ASTs on the Viper AST
- Further extend AST simplifications to improve performance
  - Possibly use a DSL in combination with macros to auto-generate simplifications