Verified Subtyping with Traits and Mixins

Asankhaya Sharma
National University of Singapore

Object Oriented Design

- S – Single Responsibility
- O – Open Close
- L – Liskov Substitution
- I – Interface Segregation
- D – Dependency Inversion

First proposed by Robert C. Martin in comp.object newsgroup in March 1995

Liskov Substitution Principle

"Let q(x) be a property provable about objects x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T"

- Barbara Liskov and Jeannette Wing

Behavior Subtyping

- Useful to reason about design of class hierarchies
- Stronger notion than typical subtyping of functions defined in type theory
  - Function subtyping is based on contravariance of argument types and covariance of the return type
- Behavioral subtyping is trivially undecidable
  - If the property is "this method always terminates"

Contributions

- Subtyping with Traits and Mixins in Scala
  - By checking entailments in separation logic
- Extend Scala with a domain specific language (SLEEK DSL)
  - Allows Scala programmers to insert subtyping checks in their programs
- Case study on subtyping in Scala Standard Library
  - Verified subtyping in 67% of Mixins

Traits and Mixins

- Traits
  - Fine grained unit of reuse
  - Similar to abstract classes but some methods can have implementations as well
- Mixin Composition (Mixin)
  - A class which contains a combination of methods from other traits and classes
  - Similar to multiple inheritance if the combination contains all methods of combined classes

Traits Example

```
trait ICell {
def get() : Int
def set(x : Int)
}
trait BICell extends ICell {
private var x : Int  = 0
def get() { x }
def set(x : Int) { this.x = x }
}
```

Similar to an Abstract Class

Implementation of ICell

Traits Example (cont.)

```
trait Double extends ICell {
abstract override def set(x : Int) {
    super.set(2*x)
    }
}
trait Inc extends ICell {
abstract override def set(x : Int) {
    super.set(x+1)
    }
}
```

Mixins Example

- OddICell (odd values)
  - class OddICell extends BICell with Inc with Double
- EvenICell (even values)
  - class EvenICell extends BICell with Double with Inc

OddICell←Double←Inc←BICell

EvenICell←Inc←Double←BICell

Class Linearization

Scala doesn't enforce Subtyping

```
def m (c: BICell with Inc with Double) : Int
{ c.get }
val oic = new OddICell
val eic = new EvenICell
m(oic)
m(eic)
```

Both calls are allowed by the type system

Only object oic is subtype of c

Verified Subtyping

- A mixin can be represented as a separation logic predicate based on class linearization

OddICell←Double←Inc←BICell
OddICell<this> == BICell<this,p> * Inc<p,q> * Double<q,null>

EvenICell←Inc←Double←BICell
EvenICell<this> == BICell<this,p> * Double<p,q> * Inc<q,null>

From Subtyping to Entailment

- Subtyping can be reduced to checking entailment between the predicates

m(oic)
OddICell<oic> |- BICell<c,p> * Inc<p,q> * Double<q,null>

Valid

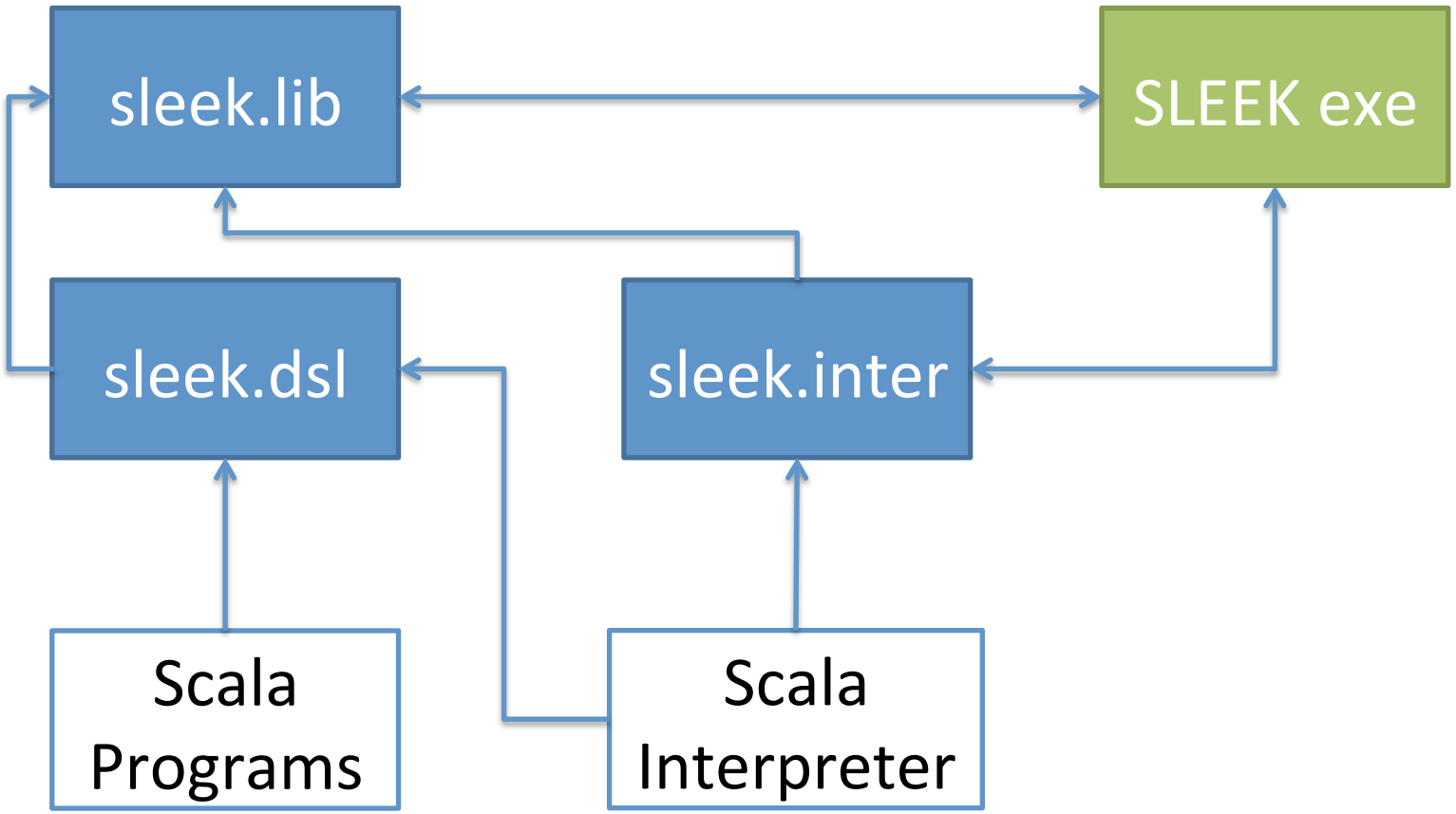
m(eic)
EvenICell<eic> |- BICell<c,p> * Inc<p,q> * Double<q,null>

Invalid

SLEEK DSL

- Implementation based on SLEEK
  - An existing separation logic based entailment prover
  - Supports user defined predicates and user specified lemmas
  - With Shape, Size, and Bag properties

Implementation Overview



Scala with SLEEK DSL

- Insert checks in Scala programs to verify subtyping using SLEEK DSL

```
def m (c: BICell with Inc with Double) : Int { c.get }
val oic = new OddICell
val eic = new EvenICell
if (OddICell<oic> |- BICell<c,p> * Inc<p,q> * Double<q,null> )
    m(oic)
if (EvenICell<eic> |- BICell<c,p> * Inc<p,q> * Double<q,null>)
    m(eic)
```

Experiments

- Scala Standard Library
- Considered four class hierarchies
  - Exceptions
  - Maths
  - Parser Combinators
  - Collections

Results

Class Hierarchy	Mixins in the Hierarchy	Mixins with Verified Subtyping	Percentage
Exceptions	11	11	100
Maths	5	4	80
Combinators	6	6	100
Collections	27	12	44
Total	**49**	**33**	**67**

Traits provide more flexibility, 33% of Mixins use Traits in a way that does not conform to subytping

Conclusions

- We use entailment proving in separation logic to check subtyping with Traits and Mixins
- A domain specific language based on SLEEK to check entailments from Scala programs
- Case study based on Scala Standard Library

Perspectives

- Lays the foundation for verifying OO Scala programs
  - Specification reuse with traits and mixins
  - Inheritance verification
- Static and Dynamic Specifications for traits and mixins
  - Avoid re-verification
  - Compositional and modular

Thank You !

- Questions ?
- Scala with SLEEK DSL
  - Web Tool, Source Code and Sample Programs
  - http://loris-7.ddns.comp.nus.edu.sg/~project/SLEEKDSL/
- Contact
  - asankhaya@nus.edu.sg