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
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 }
}
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
Scala doesn’t enforce Subtyping

```scala
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

\[
\text{OddICell} \leftarrow \text{Double} \leftarrow \text{Inc} \leftarrow \text{BICell}
\]

\[
\text{OddICell}<\text{this}> =\text{BICell}<\text{this},p> * \text{Inc}<p,q> * \text{Double}<q,\text{null}>
\]

\[
\text{EvenICell} \leftarrow \text{Inc} \leftarrow \text{Double} \leftarrow \text{BICell}
\]

\[
\text{EvenICell}<\text{this}> =\text{BICell}<\text{this},p> * \text{Double}<p,q> * \text{Inc}<q,\text{null}>
\]
From Subtyping to Entailment

• Subtyping can be reduced to checking entailment between the predicates

\[
m(oic) \quad \text{OddICell}<oic> \vdash \text{BICell}<c,p> * \text{Inc}<p,q> * \text{Double}<q,null>
\]

Valid

\[
m(eic) \quad \text{EvenICell}<eic> \vdash \text{BICell}<c,p> * \text{Inc}<p,q> * \text{Double}<q,null>
\]

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

- sleek.lib
- sleek.dsl
- sleek.inter
- Scala Programs
- Scala Interpreter
- SLEEK exe

13/5/14

FSFMA 2014
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

<table>
<thead>
<tr>
<th>Class Hierarchy</th>
<th>Mixins in the Hierarchy</th>
<th>Mixins with Verified Subtyping</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exceptions</td>
<td>11</td>
<td>11</td>
<td>100</td>
</tr>
<tr>
<td>Maths</td>
<td>5</td>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>Combinators</td>
<td>6</td>
<td>6</td>
<td>100</td>
</tr>
<tr>
<td>Collections</td>
<td>27</td>
<td>12</td>
<td>44</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>49</strong></td>
<td><strong>33</strong></td>
<td><strong>67</strong></td>
</tr>
</tbody>
</table>

*Traits provide more flexibility, 33% of Mixins use Traits in a way that does not conform to subtyping*
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
• Contact
  – asankhaya@nus.edu.sg