Taming the Wildcards
Combining Definition- and Use-Site Variance

John Altidor\textsuperscript{1}, Shan Shan Huang\textsuperscript{2}, and Yannis Smaragdakis\textsuperscript{1,3}

University of Massachusetts Amherst\textsuperscript{1}
LogicBlox Inc.\textsuperscript{2}
University of Athens, Greece\textsuperscript{3}
Outline

- Motivation for Variance.
- Existing Approaches to Variance.
- Our Approach: Combine Def-Site and Use-Site Variance.
- Case Study and Results.
- Summary.
Introduction

Existing Approaches

Combine Def-Site and Use-Site

Case Study

Motivation

Variance Introduction

Motivation for Variance

- Generics have been added to mainstream languages (e.g. Java, Scala, C#) to support parametric polymorphism.
- Generics conflict with subtyping.
- Dog <: Animal does not imply
  List<Dog> <: List<Animal>.

```
List<Dog> ld = new ArrayList<Dog>();
List<Animal> la = ld;
la.add(new Cat());
Dog d = ld.get(0);  // Assigning a Cat to a Dog!
```
Introduction to Variance

Under what conditions for type expressions \( \text{Exp1} \) and \( \text{Exp2} \) is \( \text{C<Exp1>} \) a subtype of \( \text{C<Exp2>} \)?

Four common flavors of variance:

1. **Covariance**: \( T <: U \implies \text{C<T>} <: \text{C<U>} \)
2. **Contravariance**: \( T <: U \implies \text{C<U>} <: \text{C<T>} \)
3. **Bivariance**: \( \text{C<T>} <: \text{C<U>} \) for all \( T \) and \( U \).
4. **Invariance**: \( \text{C<T>} <: \text{C<U>} \implies T <: U \) and \( U <: T \).

Existing specifications: *Definition-Site* and *Use-Site* Variance
Definition-Site Variance

- As in Scala, the definition of generic class `C[X]` determines its variance.
- Each type parameter is declared with a variance annotation.
- The variances of each type parameter’s positions should be at most the declared variance.

```scala
class RList[+X] { def get(i:Int):X }
class Func[-K, +V] { def apply(k:K):V }
class WList[-X] { def set(i:Int, x:X):Unit }
```
Use-Site Variance

- Clients declare desired variance at *use*-site.
- Java Wildcards.
  - List<? extends T> - covariant instantiation
  - List<? super T> - contravariant instantiation
  - List<? > - bivariant instantiation
  - List<T> - invariant instantiation
- List<? extends Animal> can call “Animal get(int i)” but not “void set(int i, Animal a)”.

```java
void mapSpeak(List<? extends Animal> animals) {
    for(int i = 0; i < animals.size(); i++)
        animals.get(i).speak();
}
```
Definition-Site: Pros and Cons

- Conceptual simplicity; class definition specifies its variance.
- Burden on library designers; not on users.
- Forces splitting the definitions of data types into co-, contra-, bi-, and invariant versions.
  - scala.collection.immutable.Map[A, +B]
  - scala.collection.mutable.Map[A, B]
  - A generic with n type parameters can require $3^n$ interfaces (or $4^n$ if bivariance is allowed).
Use-Site: Pros and Cons

- Flexibility: co-, contra-, and bivariant versions on the fly.
- Design classes in natural way, but burden shifts to users.
- Type signatures quickly become complicated.
- Heavy variance annotations required for subtyping; from Apache Commons-Collections Library:

```java
Iterator<? extends Map.Entry<? extends K,V>>
createEntrySetIterator(
    Iterator<? extends Map.Entry<? extends K,V>>)```

John Altidor, Shan Shan Huang, and Yannis Smaragdakis
Our Approach: Combine Def-Site and Use-Site

- Take advantage of simplicity of def-site and flexibility of use-site variance.
- Flexibility of use-site removes need for redundant classes.
- Simplicity of def-site takes complexity burden off clients, and requires far fewer variance annotations for subtyping.
- Enable reasoning about classes with both def-site and use-site annotations.
Integrating Use-Site with Def-Site

class C[-X] { def set(arg1:X):Unit }
class D[+X] { def compare(arg2:C[+X]):Unit }

▶ C[+X] says to pass X to a version of C that is at least covariant.
▶ Use-site annotation corresponds to a join operation in the standard variance lattice.

Variance of X in C[v_uX] is v_C ⊔ v_u, where v_C is def-site var of C.
Variance Composition

- What is variance of \( x \) in \( A<B<C<<X>> > \)? (Ignore use-site)
- In general, what is variance of \( x \) in \( C<E> \)?
- Defined “transform” binary operator \( \otimes \) to reason about variance of arbitrarily nested type expressions.

\[ v_1 \otimes v_2 = v_3 \]

If the variance of a type variable \( x \) in type expression \( E \) is \( v_2 \) and the def-site variance of class \( C \) is \( v_1 \), then the variance of \( x \) in type expression \( C<E> \) is \( v_3 \).
Deriving Transform Operator

- Example Case $\oplus \otimes - = -$  

Need to show $c<E>$ is contravariant wrt $x$ when generic $c$ is covariant in its type parameter and type expression $E$ is contravariant in $x$. This holds because, for any $T_1, T_2$:

$$T_1 <: T_2 \implies (\text{by contravariance of } E)$$

$$E[T_2/x] <: E[T_1/x] \implies (\text{by covariance of } c)$$

$$c<E[T_2/x]> <: c<E[T_1/x]> \implies$$

$$c<E>[T_2/x] <: c<E>[T_1/x]$$

Hence, $c<E>$ is contravariant with respect to $x$.

- See paper for remaining cases.
Summary of Transform

- Invariance transforms everything into invariance.
- Bivariance transforms everything into bivariance.
- Covariance preserves a variance.
- Contravariance reverses it.

**Definition of variance transformation:** $\otimes$

<table>
<thead>
<tr>
<th></th>
<th>+ $\otimes$ + = +</th>
<th>− $\otimes$ + = −</th>
<th>* $\otimes$ + = *</th>
<th>0 $\otimes$ + = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>− $\otimes$ − = −</td>
<td>+ $\otimes$ − = +</td>
<td>* $\otimes$ − = *</td>
<td>0 $\otimes$ − = 0</td>
</tr>
<tr>
<td>+</td>
<td>− $\otimes$ * = −</td>
<td>+ $\otimes$ * = +</td>
<td>* $\otimes$ * = *</td>
<td>0 $\otimes$ * = 0</td>
</tr>
<tr>
<td>+</td>
<td>− $\otimes$ o = o</td>
<td>+ $\otimes$ o = o</td>
<td>* $\otimes$ o = *</td>
<td>0 $\otimes$ o = 0</td>
</tr>
</tbody>
</table>
Def-Site Inference via Variance Calculus: *VarLang*

- **Java Classes:**

```java
class C<X> {
    X foo (C<? super X> csx) { ... }
    void bar (D<? extends X> dsx) { ... }
}
class D<Y> { void baz (C<Y> cx) { ... } }
```

- **Translation to *VarLang***:

```plaintext
module C<X> { X+, C<-X>- , void+, D<+X>-}
module D<Y> { void+, C<oY>- }
```
Constraint Generation

module C<X> { X+, C<-X>-, void+, D>+X>- }  
module D<Y> { void+, C<oY>- }  

- Generate constraints from VarLang modules

```plaintext
foo return type \implies \quad c \sqsubseteq + \otimes (c \sqcup X) 
foo arg type \implies \quad c \sqsubseteq - \otimes (c \sqcup C<-X>) 
bar arg type \implies \quad c \sqsubseteq - \otimes (d \sqcup D>+X>) 
baz arg type \implies \quad d \sqsubseteq - \otimes (c \sqcup C<oY>)
```
class C<X> {
    X foo (C<? super X> csx) { ... }
    void bar (D<? extends X> dsx) { ... }
}
class D<Y> { void baz (C<Y> cx) { ... } }

- C cannot be contravariant.
  - foo return type \(\Rightarrow c \sqsubseteq + \) but \(- \sqsubseteq +\)
- Constraints correspond to checking def-site variance annotations.
Constraint Solving Enables Inference

\[\begin{align*}
\text{foo return type} & \implies c \sqsubseteq + \\
\text{foo arg type} & \implies c \sqsubseteq - \otimes (c \sqcup -) \\
\text{bar arg type} & \implies c \sqsubseteq - \otimes (d \sqcup +) \\
\text{baz arg type} & \implies d \sqsubseteq - \otimes (c \sqcup o)
\end{align*}\]

- Trivial solution: \(c = o\) and \(d = o\).
- Most general solution: \(c = +\) and \(d = -\).
- Solve constraints by fix-point computation running in polynomial of the program size (number of constraints).
Case Study: Def-Site Inference for Java

- Mapped Java classes to *VarLang* modules.
  - Argument types map to contravariant positions.
  - Types of non-final fields map to covariant and contravariant.
  - etc.
- Applied inference to large, standard Java libraries.
- Example inferences: `java.util.Iterator<E>` is covariant and `java.util.Comparator<T>` is contravariant.
Sample Results from Inference

<table>
<thead>
<tr>
<th>Library</th>
<th># Type defs</th>
<th># Gen defs</th>
<th>Type Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.*</td>
<td>classes</td>
<td>5550</td>
<td>def</td>
</tr>
<tr>
<td></td>
<td>interfaces</td>
<td>1710</td>
<td>defs</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>7260</td>
<td>invar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>99</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>44</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>143</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>69%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>43%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>57%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>39%</td>
<td></td>
</tr>
<tr>
<td>JScience</td>
<td>classes</td>
<td>70</td>
<td>def</td>
</tr>
<tr>
<td></td>
<td>interfaces</td>
<td>51</td>
<td>defs</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>121</td>
<td>invar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>36</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>76%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>55%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>45%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>69%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>31%</td>
<td></td>
</tr>
<tr>
<td>Apache Collec.</td>
<td>classes</td>
<td>226</td>
<td>def</td>
</tr>
<tr>
<td></td>
<td>interfaces</td>
<td>23</td>
<td>defs</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>249</td>
<td>invar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>187</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>209</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>66%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>55%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>45%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>65%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>35%</td>
<td></td>
</tr>
<tr>
<td>Google Guava</td>
<td>classes</td>
<td>204</td>
<td>def</td>
</tr>
<tr>
<td></td>
<td>interfaces</td>
<td>35</td>
<td>defs</td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>239</td>
<td>invar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>101</td>
<td>variant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>127</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>90%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>46%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>54%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>81%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>19%</td>
<td></td>
</tr>
</tbody>
</table>

- Analysis was modular but conservative (e.g. ignored method bodies).
  - “foo(List<Animal> arg)” could have been “foo(List<? extends Animal> arg)”.
Summary

- Combine def-site and use-site variance to reap their advantages and remove their disadvantages.
- Our reasoning enables adding def-site variance inference to Java and checking Scala classes with use-site variance annotations.
- Analysis over Java libraries shows potential impact even with a conservative analysis.
- See PLDI 2011 paper for further details.