Modeling Object-Oriented Generics

A Lattice- & Category-Theoretic Approach

Moez AbdelGawad (moez@cs.rice.edu)
Informatics Research Institute, SRTA-City, Alexandria, Egypt

Goals

- A simple intuitive model of generic object-oriented (OOP) type systems that includes Java wildcards, F-bounded generics, and erasure — and *avoids explicit use of existential types*.
  - Existential types naturally arise in functional programming but do not match well with object-oriented inheritance and subtyping.
- Provide a basis for better compiler diagnostics (no mystic "`capture of …`" compiler error messages).
- Accommodate a first-class approach to generics, where parametric type info is available at runtime.

Code Examples

```
class Real     extends Num
class Integer  extends Real
class PosInt   extends Integer
List<Integer> li
li instanceof List<? extends Num>       // yes
li instanceof List<? super Num>         // no
List<? extends Vector<? extends Num>> // nest
```

Wildcard Types (Variance Annot.)

```
List[Integer-Num] lin
List[PosInt-Num]  lpi
List[Integer] li
lpi instanceof List[Null-Num]       // yes
lpi instanceof List[Integer-Num]    // no
lin := li // yes    li := lin       // no
List[Null-Vector[Null-Num]]         // nest
```

Interval Types (Hypothetical)

```
// Default Type Arguments (DTAs)
class C<T=Object>
// default type arg. is Object

class G<T=Integer>
// default type arg. is Integer

// Erased type, where class name
//  is used as type name
C c   // type of c is C<Object>
G g   // type of g is G<Integer>

// If unspecified, default default
// type arg. is upper bound (Java)
```

Erasure and DTAs (Hypothetical)

```
class C<T>              // T ranges over all types
class D<T extends C<T>>  // T is F-bounded; ranges over C-subtypes (C-coalgebras/C-smalls)
class E<T extends E<T>>  // E=Enum.  T ranges over E-subtypes (E-coalgebras/E-small types)
class E<T super C<T>>    // Hypoth.  T ranges over C-supertypes (C-algebras/C-big types)
```

F-bounded Generics (F-subtypes and F-supertypes)

Constructing Subtyping from Subclassing (and Containment)

A **class** is a member of set **C**.
A **generic class** in **G ⊆ C** is a class having a type parameter.
An (admittable, ground) **type** in set **T** is a non-generic class or a generic class parameterized by an interval.
A (type) **interval** in set **I** is a pair of <:-ordered types.

Interval-based Subtyping Rules (Core)

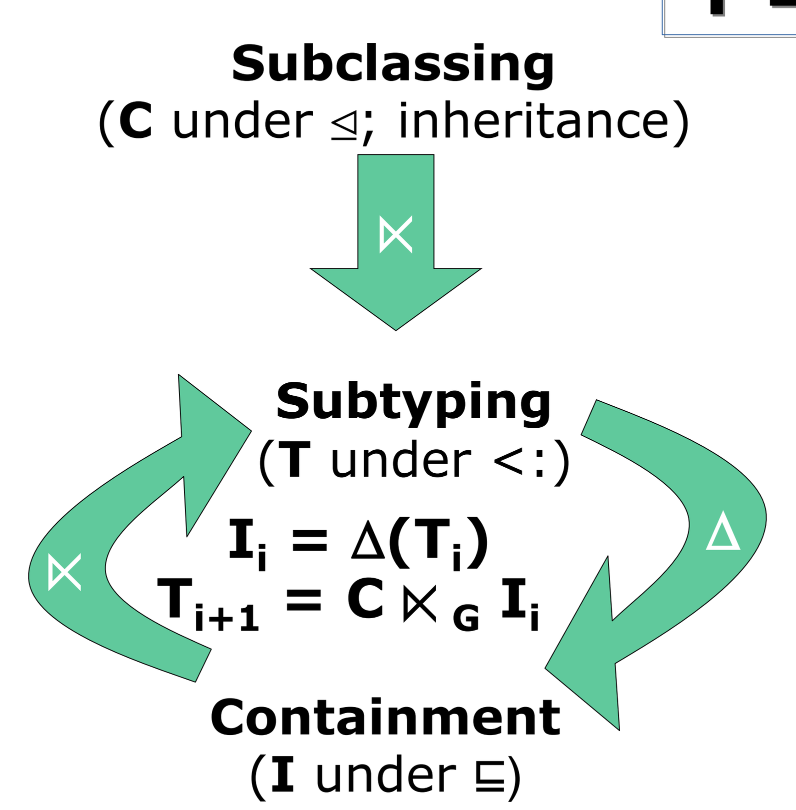
$$\frac{\texttt{Null} \trianglelefteq \texttt{Object}}{\bot(\texttt{Null}_t) <: \top(\texttt{Object}_t)}\; Subt_0$$

$$\frac{\texttt{C} \trianglelefteq \texttt{D} \quad \texttt{I} \sqsubseteq \texttt{J}}{\texttt{C<I>} <: \texttt{D<J>}}\; Subt_{GG}$$

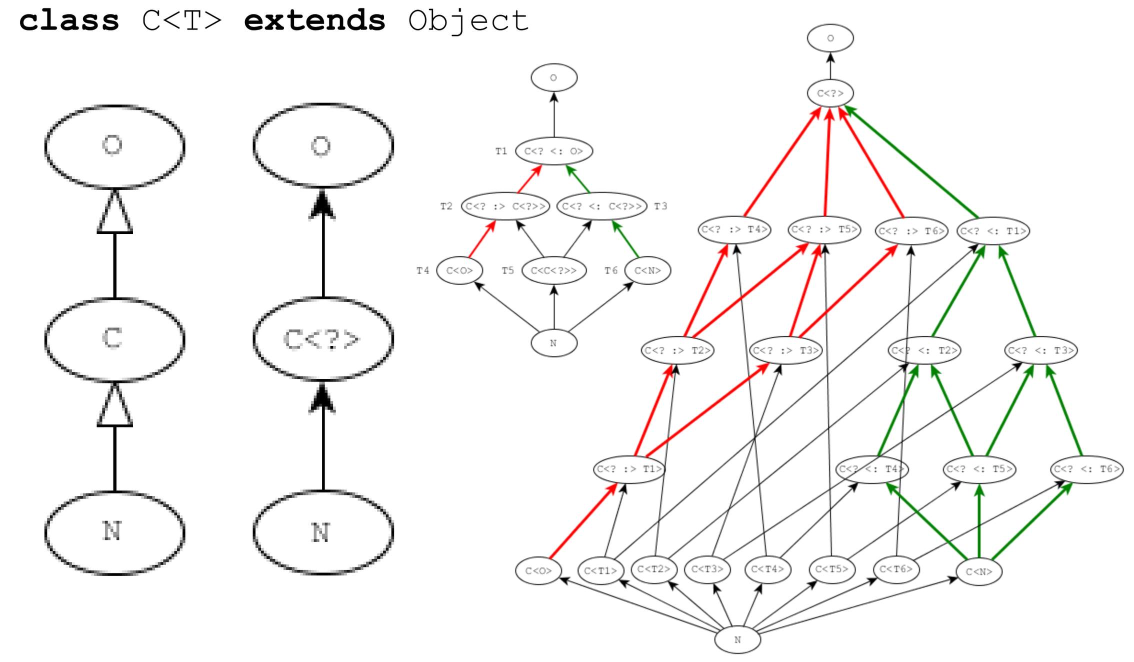
$$\frac{T_l <: S_l \quad S_u <: T_u}{[S_l - S_u] \sqsubseteq [T_l - T_u]}\; Subint/Cont$$

(Ground=No tvars. Rule $Subt_{GG}$ assumes **class** `C<T>` **extends** `D<T>`)

$$\mathbf{T = C \ltimes_G \Delta(T)}$$

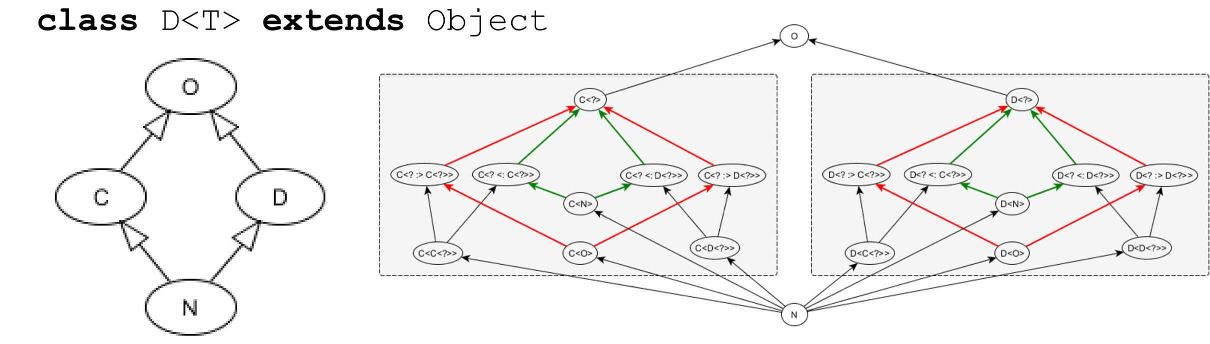


`class C<T> extends Object`

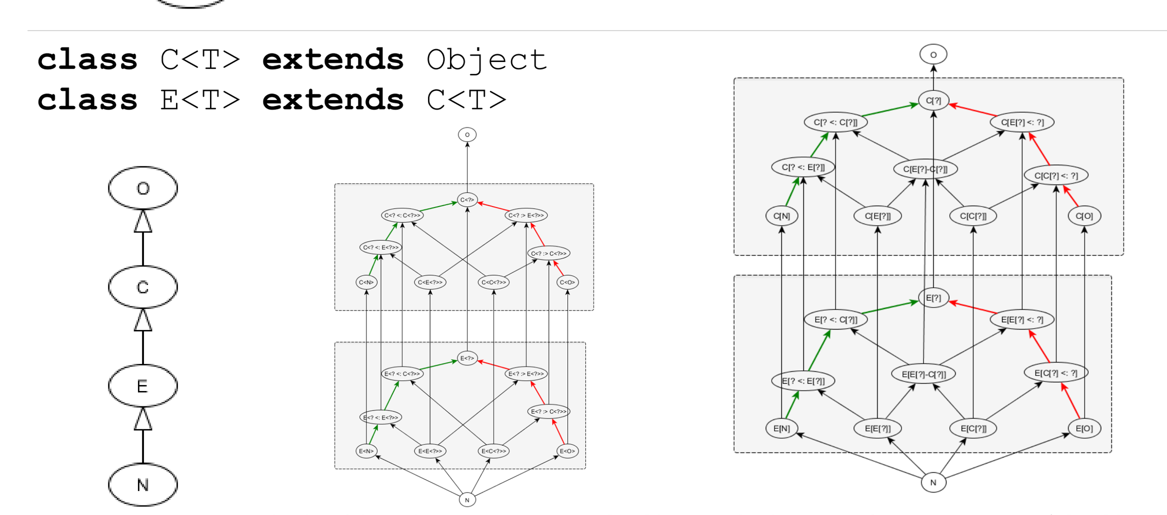


(Note the "triangles" in subtyping, due to co/contra/in-variant wildcard subtyping rules)

```
class C<T> extends Object
class D<T> extends Object
```



```
class C<T> extends Object
class E<T> extends C<T>
```



(w/ wildcards. $\mathbf{T_1 = C \ltimes_G \Delta(T_0)}$)   (w/ intervals. $\mathbf{T_1 = C \ltimes_G ⇳(T_0)}$)

Partial Products and Intervals. Erasure, Free Types, and the Erasure Galois Connection (EGC)

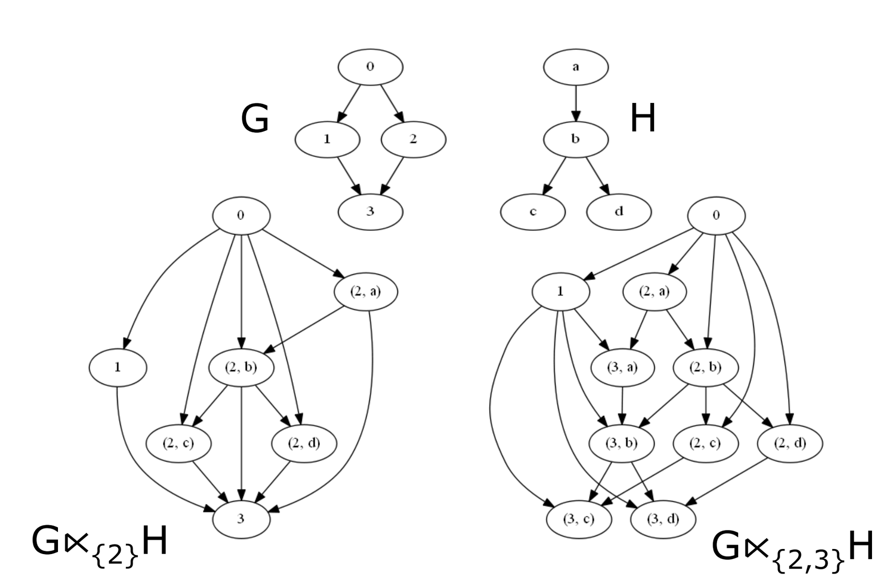
- The partial product operator (⋉) is used to model that *only generic classes* are paired with type arguments to construct new types but all types, incl. *non-generic* ones, share in the subtyping relation.
- Operators Δ and ⇳ ('triangle/wildcards' & 'intervals') construct intervals in Hasse diagram (a directed acyclic graph) of input subtyping relation. Intervals of a directed graph are *paths* modulo endpoints.
  - Operator Δ requires ⊤ (i.e., `Object`, or `O`) as an upper bound or ⊥ (i.e., `Null`, or `N`) as a lower bound of the constructed interval; operator ⇳ only requires the lower bound to be a subtype of the upper bound (i.e., ⇳ is strictly more general than Δ).

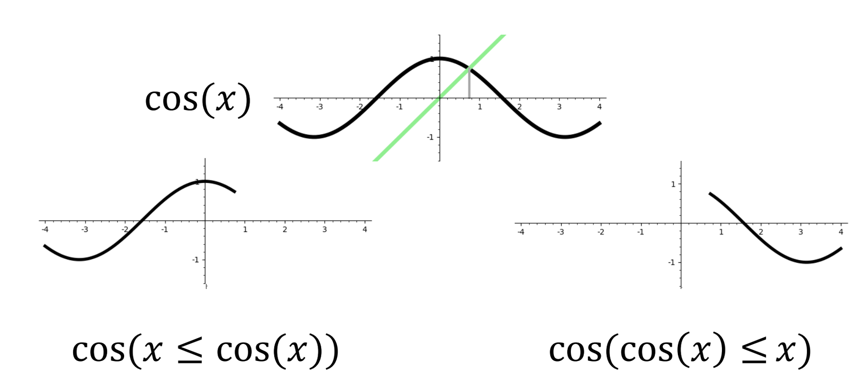
- The **erasure** $E(PT)$ of a parameterized type PT maps type PT to the *class* used to construct the type (e.g., E(`List<Integer>`)=`List`).
- The **free type** $FT(C)$ corresponding to a generic class C is the parameterized type representing the most general instantiation of class C (e.g., FT(`List`)=`List<?>`).
- Erasure and free types define a Galois connection (EGC) between subclassing and subtyping (i.e., are adjoints of an adjunction):

  for all type arguments `T`, classes `C`, `D`, `C ⊴ D` ⇔ `C<T> <: D<?>`
  (`C` is the erasure of `C<T>`; `D<?>` is the free type of `D`)

  which expresses a fundamental property of generic OOP, namely, that **inheritance is the only basis of subtyping** in generic OOP**.**
- The composition $FT \circ E$ maps a parameterized type to its corresponding free type, while the composition $E \circ FT$ maps a class to itself.

Illustrating Partial Graph/Poset Products, and Function-bounded Real Functions





F-sub/supertypes, Cofree Types, and Doubly F-bounded Generics (DFBG)

- The two compositions $FT \circ E$ and $E \circ FT$ define two self-maps (a.k.a., endomaps) over subtyping and over subclassing/inheritance, resp. The two endomaps are *closure operators*.

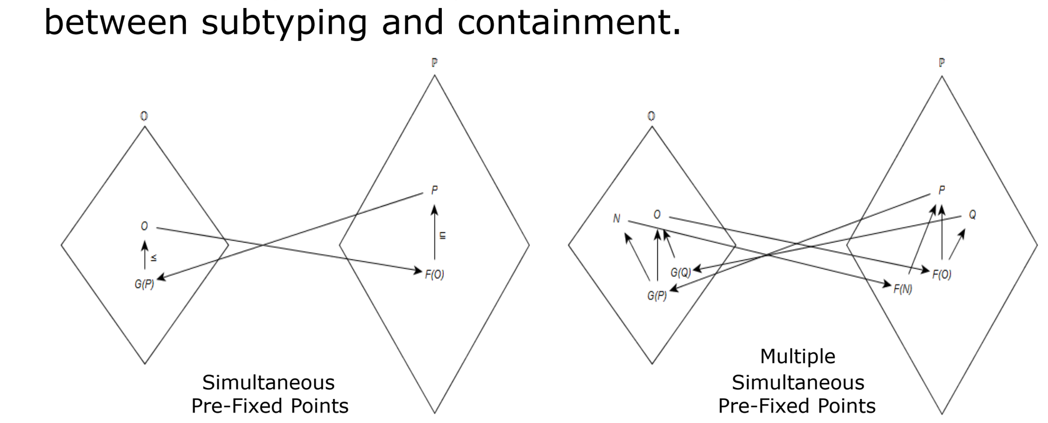
Preorders & Posets	(Thin) Categories
Closure Operator	Monad
Pre-Fixed Point (Inductive Object)	Algebra
Post-Fixed Point (Coinduct. Object)	Coalgebra
Least Pre-FP/Greatest Post-FP	Initial Algebra/Final Coalgebra

- For a generic class `C`, its **F-subtypes** or **C-subtypes** (types `T` in **T** where `T <: C<T>`) are C-coinductive types (or, C-coalgebras); its **F-supertypes** or **C-supertypes** (types `T` in **T** where `C<T> <: T`) are C-inductive types (or, C-algebras).
- Free types are the greatest F-subtypes. **Cofree types** (e.g., `C<!>`; useful as l-bounds in DFBG) are the least F-supertypes.

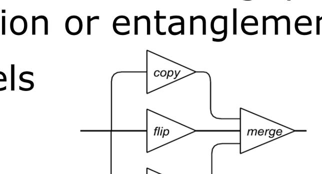
- Real functions—i.e., ones from ℝ to ℝ—with *function-bounded domains* (e.g., the function $\cos(x \leq \cos(x))$) inspire our modeling of F-bounded and doubly F-bounded generics (DFBG).
- A **doubly F-bounded** type variable in DFBG ranges over types that are F-subtypes of the erasure of its upper bound and that are F-supertypes of the erasure of its lower bound.
- DFBG motivates defining **admittible** versus **valid** entities: type args, types (e.g., `Enum<Object>`) and subtype relations.

More

- Future work:
  - Include type variables (tvars) in the subtyping rules.
  - Define a ⋉–like operator to model general inheritance declarations (e.g., **class** `C<T>` **extends** `D<F<T>>`).
  - Model multiple type params (multi-ary generic classes).
- Mutual Coinduction: For modeling mutual generic classes, circular or infinitely-justified subtyping relations, mutually-dependent type params, and the mutual dependency between subtyping and containment.



  - (*Infinite*) *mutual coinduction* useful in modeling quantum phenomena, e.g., superposition or entanglement?
- Java Subtyping Operad (JSO): Models the construction of the *self-similar* subtyping relation between ground generic types in Java.



- References: See http://arxiv.org/a/abdelgawad_m_1