Generics und Wildcards in Java 5.0

Martin Plümicke

University of Cooperative Education
Stuttgart/Horb

13. Februar 2008

Overview

Type terms in Java 5.0:

explicitly used:

```
Vector<Vector<Integer>>
Vector<? extends List<Object>>
Vector<? super List<Object>>
```

only inferred:

```
? extends List<Object>
? super List<Object>
```

Formal description of type terms

Formal description of type terms

Definition *Rank alphabet*:

- $\Theta = \Theta^{(n)}_{n \in \mathbb{N}}$ (rank alphabet)
- TV (set of type variables)
- $T_\Theta(\ TV\ )$ (Java 5.0 type terms)

Formal description of type terms

Definition *Rank alphabet*:

- $\Theta = \Theta^{(n)}_{n \in \mathbb{N}}$ (rank alphabet)
- TV (set of type variables)
- $T_\Theta(\ TV\ )$ (Java 5.0 type terms)

Example:

```
class A<a> implements I<a> { ... }
class B<a> extends A<a> { ... }
class C<a extends I<b>,b> { ... }
interface I<a> { ... }
```

- $\Theta^{(1)} = \{\, \mathtt{A}, \mathtt{B}, \mathtt{I} \,\}$
  $\Theta^{(2)} = \{\, \mathtt{C} \,\}$
- `A<Integer>`, `A<B<Boolean>>`, `C<A<Object>, Object>` $\in T_\Theta(\ TV\ )$

Formal description of type terms

Definition *Rank alphabet*:

- $\Theta = \Theta^{(n)}_{n \in \mathbb{N}}$ (rank alphabet)
- TV (set of type variables)
- $T_\Theta(\ TV\ )$ (Java 5.0 type terms)

Example:

```
class A<a> implements I<a> { ... }
class B<a> extends A<a> { ... }
class C<a extends I<b>,b> { ... } Bound
interface I<a> { ... }
```

- $\Theta^{(1)} = \{\, \mathtt{A}, \mathtt{B}, \mathtt{I} \,\}$
  $\Theta^{(2)} = \{\, \mathtt{C} \,\}$
- `A<Integer>`, `A<B<Boolean>>`, `C<A<Object>, Object>` $\in T_\Theta(\ TV\ )$
- `C<C<a>, a>` $\in T_\Theta(\ TV\ )$, **but no type term!!!**

Bounded type variables

Bounded type variables

Definition:

- $BTV = (BTV^{(ty)})_{ty}$ (set of bounded type variables)
  Notation: $a|_{ty}$ means a is bounded by the (intersection) type ty.

Bounded type variables

Definition:

- $BTV = (BTV^{(ty)})_{ty}$ (set of bounded type variables)
  Notation: $a|_{ty}$ means a is bounded by the (intersection) type ty.

Example:

```
class BoundedTypeVars<a extends Number> {
    <t extends Vector<Integer> & J<a> & I,
     r extends Number> void m ( ... ) { ... } }
```

- $BTV^{(\texttt{Number})} = \{\,\texttt{a}, \texttt{r}\,\}$
- $BTV^{(\texttt{Vector<Integer>}\,\&\,\texttt{J<a>}\,\&\,\texttt{I})} = \{\,\texttt{t}\,\}$

Rank alphabet becomes *type signature*

Rank alphabet becomes *type signature*

Definition *Type signature*:

- $TS = (TC^{(tv_1 \ldots tv_n)})_{tv_i \in BTV}$ (BTV^*–indexed set of *type constructors*)

Rank alphabet becomes *type signature*

Definition *Type signature*:

- $TS = (TC^{(tv_1 \ldots tv_n)})_{tv_i \in BTV}$ (BTV^*–indexed set of *type constructors*)

Example:

```
class A<a> implements I<a> { ... }
class B<a> extends A<a> { ... }
class C<a extends I<b>,b> { ... }
interface I<a> { ... }
interface J<a> { ... }
class D<a extends B<a> & J<b>, b> { ... }
```

- $TC^{(\mathtt{a}|_{\mathtt{Object}})} = \{\mathtt{A}, \mathtt{B}, \mathtt{I}, \mathtt{J}\}$
- $TC^{(\mathtt{a}|_{\mathtt{I<b>}}\ \mathtt{b}|_{\mathtt{Object}})} = \{\mathtt{C}\}$
- $TC^{(\mathtt{a}|_{\mathtt{B<a>\&J<b>}}\ \mathtt{b}|_{\mathtt{Object}})} = \{\mathtt{D}\}$

Simple types $\mathrm{SType}_{TS}(\,BTV\,)$ (first approach)

- $BTV \subseteq \mathrm{SType}_{TS}(\,BTV\,)$
- $TC^{()} \subseteq \mathrm{SType}_{TS}(\,BTV\,)$
- For $C \in TC^{(a_1|_{ty_1} \ldots a_n|_{ty_n})}$ and a substitution σ

$$C\text{<}\sigma(\,a_1\,), \ldots, \sigma(\,a_n\,)\text{>} \in \mathrm{SType}_{TS}(\,BTV\,),$$

if $\sigma(\,a_i\,) \leq^* \sigma(\,ty_i\,)$ ($\leq^*$ is the subtyping relation).

Example:

```
class A<a> implements I<a> { ... }

class B<a> extends A<a> { ... }

class C<a extends I<b>,b> { ... }

interface I<a> { ... }
```

- $TC^{(\texttt{a}|_{\texttt{Object}})} = \{\, \texttt{A}, \texttt{B}, \texttt{I} \,\}$
  $TC^{(\texttt{a}|_{\texttt{I<b>}}\ \texttt{b}|_{\texttt{Object}})} = \{\, \texttt{C} \,\}$
- $\texttt{A<Integer>}, \texttt{A<B<Boolean>>}, \texttt{C<A<Object>, Object>}$
  $\in \mathrm{SType}_{TS}(\,BTV\,)$
- $\texttt{C<C<a>, a>} \notin \mathrm{SType}_{TS}(\,BTV\,)$

Subtyping ordering $\leq^*$

Definition:

- if θ `extends/implements` θ' then $\theta \leq^* \theta'$.
- if $\theta_1 \leq^* \theta_2$ then $\sigma_1(\theta_1) \leq^* \sigma_2(\theta_2)$ for all substitutions σ_1, σ_2 with $\sigma_1(a) = \sigma_2(a)$ (soundness condition).

Subtyping ordering $\leq^*$

Definition:

- if θ `extends`/`implements` θ' then $\theta \leq^* \theta'$.
- if $\theta_1 \leq^* \theta_2$ then $\sigma_1(\theta_1) \leq^* \sigma_2(\theta_2)$ for all substitutions σ_1, σ_2 with $\sigma_1(a) = \sigma_2(a)$ (soundness condition).

Example:

```
class A<a> implements I<a> { ... }
class B<a> extends A<a> { ... }
class C<a extends I<b>,b> { ... }
interface I<a> { ... }
```

- `A<a>` $\leq^*$ `I<a>`, as `A<a> implements I<a>`
- `A<Integer>` $\leq^*$ `I<Integer>`, where $\sigma_1 = [a \mapsto \texttt{Integer}] = \sigma_2$
- `A<Integer>` $\not\leq^*$ `I<Object>`, although indeed `Integer` $\leq^*$ `Object`, but $\sigma_1(\texttt{a}) \neq \sigma_2(\texttt{a})$.

Soundness condition: $\sigma_1(\, a\,) = \sigma_2(\, a\,)$

```
class Super { ... }
class Sub extends Super { ... }

class Application {
  public static void main(String[] args) {
    Vector<Super> v = new Vector<Sub> ();  //not allowed
    v.addElement(new Super());      //wrong: Super ≰* Sub
  }
```

Soundness condition: $\sigma_1(a)=\sigma_2(a)$

```
class Super { ... }
class Sub extends Super { ... }

class Application {
  public static void main(String[] args) {
    Vector<Super> v = new Vector<Sub> ();  //not allowed
    v.addElement(new Super());     //wrong: Super ≰* Sub
  }

  void meth (Vector<Sub> subvec) {
    Vector<Super> supervec = subvec; //could make sense
    ...                              //but not allowed
  }
}
```

Introduction of wildcards

`extends`–**wildcards**

```
Vector<? extends Super> v = new Vector<Sub> ();
```

Means: The elements of the vector `v` are subtypes of `Super`.

Introduction of wildcards

`extends`–**wildcards**

```
Vector<? extends Super> v = new Vector<Sub> ();
```

Means: The elements of the vector `v` are subtypes of `Super`.

$\Rightarrow$

- `Vector<Sub>` $\leq^*$ `Vector<? extends Super>`
- `Super` $\not\leq^*$ `? extends Super`

Introduction of wildcards

extends–**wildcards**

```
Vector<? extends Super> v = new Vector<Sub> ();
```

Means: The elements of the vector v are subtypes of Super.

$\Rightarrow$

- Vector<Sub> $\leq^*$ Vector<? extends Super>
- Super $\not\leq^*$? extends Super
- `v.addElement(new Super());` not valid

super–wildcards

Problem:

```
void addSuperElement(Vector<Sub> v) {
    v.addElement(new Sub ());
}
```

Application of `addSuperElement(new Vector<Super> ());` is not allowed, although it make sense.

super–wildcards

Problem:

```
void addSuperElement(Vector<Sub> v) {
    v.addElement(new Sub ());
}
```

Application of `addSuperElement(new Vector<Super> ());` is not allowed, although it make sense.

Solution:

```
void addSuperElement(Vector<? super Sub> v) {
    v.addElement(new Sub ());
}
```

Now `addSuperElement(new Vector<Super> ());` is allowed.

Properties of super–wildcards

- `Vector<Super>` $\leq^*$ `Vector<? super Sub>`, as `Sub` $\leq^*$ `Super`.
- `? super Sub` $\not\leq^*$ `Sub`
- `Vector<? super Sub> v;`
  `Sub e = v.elementAt(0);` not valid.

Abbreviations for wildcards:

Instead of A<? extends B> we write

$$A<_{?}B>$$

and instead of C<? super D> we write

$$C<^{?}D>.$$

Nested extends–declarations

```
class Matrix<a> extends Vector<Vector<a>> { }
```

From `Matrix<a>` $\leq^*$ `Vector<Vector<a>>` follows

- `Matrix<Super>` $\leq^*$ `Vector<Vector<Super>>`

Nested extends–declarations

```
class Matrix<a> extends Vector<Vector<a>> { }
```

From $\texttt{Matrix<a>} \leq^* \texttt{Vector<Vector<a>>}$ follows

- $\texttt{Matrix<Super>} \leq^* \texttt{Vector<Vector<Super>>}$

Is the following correct?

- $\texttt{Matrix<Sub>} \leq^* \texttt{Vector<Vector<}_?\texttt{Super>>}$

Nested extends–declarations

```
class Matrix<a> extends Vector<Vector<a>> { }
```

From Matrix<a> $\leq^*$ Vector<Vector<a>> follows

- Matrix<Super> $\leq^*$ Vector<Vector<Super>>

Is the following correct?

- Matrix<Sub> $\leq^*$ Vector<Vector<$_?$Super>>

```
Vector<Vector<? extends Super>> v = new Matrix<Sub>();
v.addElement(new Vector<Super>());
                              //wrong: Vector<Super> ≰* Vector<Sub>
```

Nested extends–declarations

```
class Matrix<a> extends Vector<Vector<a>> { }
```

From Matrix<a> $\leq^*$ Vector<Vector<a>> follows

- Matrix<Super> $\leq^*$ Vector<Vector<Super>>

Is the following correct?

- Matrix<Sub> $\leq^*$ Vector<Vector<$_?$Super>>

```
Vector<Vector<? extends Super>> v = new Matrix<Sub>();
v.addElement(new Vector<Super>());
                              //wrong: Vector<Super> ≰* Vector<Sub>
```

$\Rightarrow$ Matrix<Sub> $\not\leq^*$ Vector<Vector<$_?$Super>>

Capture conversion (for type constructors without bounds)

Let $C \in TC^{(a_1|_{\texttt{Object}},\dots,a_n|_{\texttt{Object}})}$.

$CC(\ C\texttt{<}\theta_1,\dots,\theta_n\texttt{>}\ ) = C\texttt{<}\overline{\theta}_1,\dots,\overline{\theta}_n\texttt{>}$ is defined as:

- if $\theta_i = {}_{?}\theta'_i$: $\overline{\theta}_i = b_i|^{\theta'_i}$, b_i fresh type variable with *upper bound* θ'_i.
- if $\theta_i = {}^{?}\theta'_i$: $\overline{\theta}_i = {}_{\theta'_i}|b_i$, b_i fresh type variable with *lower bound* θ'_i.
- otherwise $\overline{\theta}_i = \theta_i$

Capture conversion (for type constructors with bounds)

Let $C \in TC^{(a_1|_{u_1}, \ldots, a_n|_{u_n})}$.
$CC(\, C\text{<}\theta_1, \ldots, \theta_n\text{>}\,) = C\text{<}\overline{\theta}_1, \ldots, \overline{\theta}_n\text{>}$ is defined as:

- if $\theta_i = ?$: $\overline{\theta}_i = b_i|^{u_i[a_j \mapsto \overline{\theta}_j \,|\, 1 \leqslant j \leqslant n]}$, b_i fresh type variable
- if $\theta_i = {}_?\theta'_i$: $\overline{\theta}_i = b_i|^{\theta'_i \,\&\, u_i[a_j \mapsto \overline{\theta}_j \,|\, 1 \leqslant j \leqslant n]}$, b_i fresh type variable with *upper bound* $\theta'_i \,\&\, u_i[a_j \mapsto \overline{\theta}_j \,|\, 1 \leqslant j \leqslant n]$).
- if $\theta_i = {}^?\theta'_i$: $\overline{\theta}_i = {}_{\theta'_i}|b_i|^{u_i[a_j \mapsto \overline{\theta}_j \,|\, 1 \leqslant j \leqslant n]}$, b_i fresh type variable with *lower bound* θ'_i and *upper bound* $u_i[a_j \mapsto \overline{\theta}_j \,|\, 1 \leqslant j \leqslant n]$).
- otherwise $\overline{\theta}_i = \theta_i$

Simple types $\text{SType}_{TS}(\,BTV\,)$ (t.c. without bounds)

- $BTV \subseteq \text{SType}_{TS}(\,BTV\,)$
- $TC^{()} \subseteq \text{SType}_{TS}(\,BTV\,)$
- For $ty_i \in \text{SType}_{TS}(\,BTV\,) \cup \{\,?\,\} \cup \{\,?\ \texttt{extends}\ \tau \mid \tau \in \text{SType}_{TS}(\,BTV\,)\,\} \cup \{\,?\ \texttt{super}\ \tau \mid \tau \in \text{SType}_{TS}(\,BTV\,)\,\}$

  and $C \in TC^{(a_1|_{\texttt{Object}} \dots a_n|_{\texttt{Object}})}$ holds

$$C\texttt{<}ty_1, \dots, ty_n\texttt{>} \in \text{SType}_{TS}(\,BTV\,).$$

Simple types $\mathrm{SType}_{TS}(\,BTV\,)$ (t.c. with bounds)

- $BTV \subseteq \mathrm{SType}_{TS}(\,BTV\,)$
- $TC^{()} \subseteq \mathrm{SType}_{TS}(\,BTV\,)$
- For $\;ty_i \in \mathrm{SType}_{TS}(\,BTV\,)$
  $\cup \{\,?\,\}$
  $\cup \{\,?\ \texttt{extends}\ \tau \mid \tau \in \mathrm{SType}_{TS}(\,BTV\,)\,\}$
  $\cup \{\,?\ \texttt{super}\ \tau \mid \tau \in \mathrm{SType}_{TS}(\,BTV\,)\,\}$

  and $C \in TC^{(a_1|_{b_1} \ldots a_n|_{b_n})}$ holds

$$C\texttt{<}ty_1, \ldots, ty_n\texttt{>} \in \mathrm{SType}_{TS}(\,BTV\,)$$

if for $CC(\,C\texttt{<}ty_1, \ldots, ty_n\texttt{>}\,) = C\texttt{<}\overline{ty_1}, \ldots, \overline{ty_n}\texttt{>}$ holds:

$$\overline{ty_i} \leq^* b_i[a_j \mapsto \overline{ty_j} \mid 1 \leqslant j \leqslant n],$$

Subtyping ordering $\leq^*$ (extension)

- if θ `extends/implements` θ' then $\underline{\theta \leq^* \theta'}$.
- if $\theta_1 \leq^* \theta_2$ then $\underline{\sigma_1(\,\theta_1\,) \leq^* \sigma_2(\,\theta_2\,)}$, where for each type variable a of θ_2 holds $\sigma_1(\,a\,) = \sigma_2(\,a\,) \in \mathrm{SType}_{TS}(\,BTV\,)$ (soundness condition).
- It holds $\underline{C<\theta_1, \ldots, \theta_n> \leq^* C<\theta'_1, \ldots, \theta'_n>}$ if for θ_i and θ'_i either
  - $\theta_i = {}_?\overline{\theta}_i$, $\theta'_i = {}_?\overline{\theta}'_i$ and $\overline{\theta}_i \leq^* \overline{\theta}'_i$ or
  - $\theta_i = {}^?\overline{\theta}_i$, $\theta'_i = {}^?\overline{\theta}'_i$ and $\overline{\theta}'_i \leq^* \overline{\theta}_i$ or
  - θ_i, θ'_i are no wildcard arguments and $\theta_i = \theta'_i$ or
  - $\theta'_i = {}_?\theta_i$ or
  - $\theta'_i = {}^?\theta_i$
- From $C<\overline{\theta}_1, \ldots, \overline{\theta}_n> \leq^* C<\theta'_1, \ldots, \theta'_n>$ follows with $C<\overline{\theta}_1, \ldots, \overline{\theta}_n> = CC(\,C<\theta_1, \ldots, \theta_n>\,)$: $\underline{C<\theta_1, \ldots, \theta_n> \leq^* C<\theta'_1, \ldots, \theta'_n>}$
- $\underline{T|^{(\theta_1 \& \ldots \& \theta_n)} \leq^* \theta_i}$ for any $1 \leqslant i \leqslant n$.
- $\underline{\theta \leq^* {}_{\theta}|\,T}$

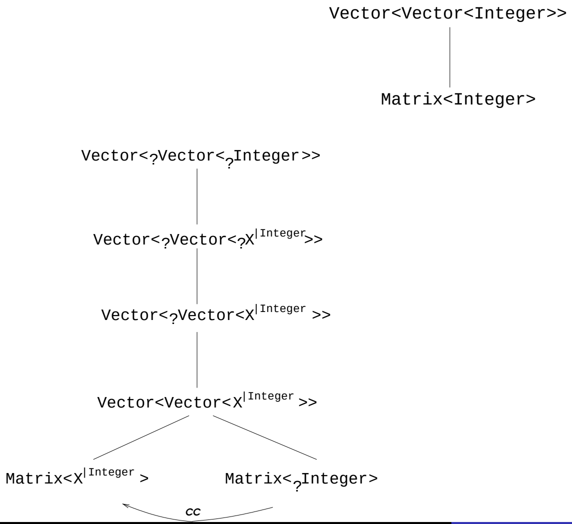
Example: Matrix<a> $\leq^*$ Vector<Vector<a>>

Example: Matrix<a> $\leq^*$ Vector<Vector<a>>

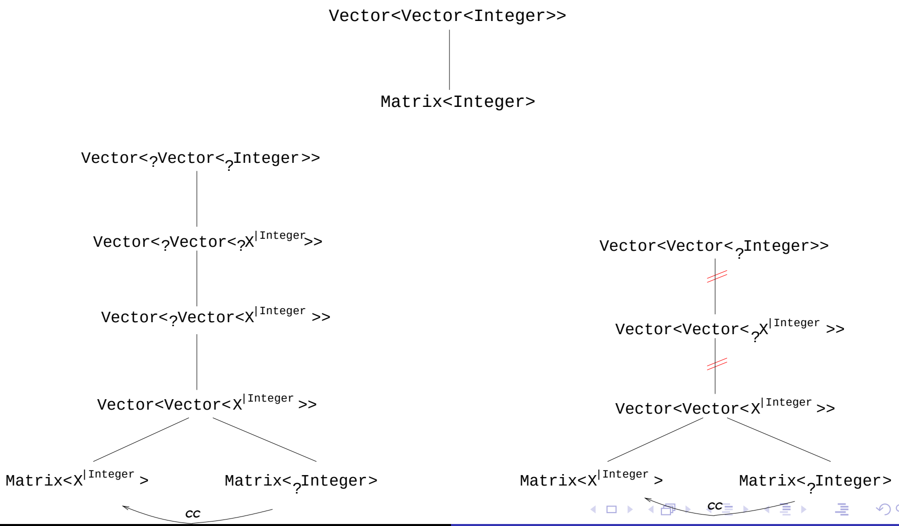
Vector<Vector<Integer>>

Matrix<Integer>

Example: Matrix<a> $\leq^*$ Vector<Vector<a>>



Example: Matrix<a> $\leq^*$ Vector<Vector<a>>



Motivation for a continuation of $\leq^*$ on wildcard types

An element should be read from a vector of a wildcard type:

```
Vector<? extends Super> v = new Vector<Sub> ();
Super superElement = v.elementAt(i);
```

$\Longrightarrow$ `? extends Super` $\leq^*$ `Super`

Motivation for a continuation of $\leq^*$ on wildcard types

An element should be read from a vector of a wildcard type:

```
Vector<? extends Super> v = new Vector<Sub> ();
Super superElement = v.elementAt(i);
```

$\Longrightarrow$ `? extends Super` $\leq^*$ `Super`

An element of a subclass should be added to a vector of a superclass:

```
Vector<? super Super> v2 = new Vector<Super> ();
v2.addElement(new Sub());
```

$\Longrightarrow$ `Super`, `Sub` $\leq^*$ `? super Super`

Extended simple types

$\text{SType}_{TS}(\,BTV\,)$ is a set of simple types:
The corresponding set of *extended simple types* is given as

$$\begin{array}{rcl} \text{ExtSType}_{TS}(\,BTV\,) & = & \text{SType}_{TS}(\,BTV\,) \\ & \cup & \{\,?\,\} \\ & \cup & \{\,?\,\texttt{extends}\ \theta \mid \theta \in \text{SType}_{TS}(\,BTV\,)\,\} \\ & \cup & \{\,?\,\texttt{super}\ \theta \mid \theta \in \text{SType}_{TS}(\,BTV\,)\,\} \end{array}.$$

Extended simple types

$\mathrm{SType}_{TS}(\,BTV\,)$ is a set of simple types:
The corresponding set of *extended simple types* is given as

$$\begin{aligned}\mathrm{ExtSType}_{TS}(\,BTV\,) = &\ \mathrm{SType}_{TS}(\,BTV\,) \\ \cup &\ \{\,?\,\} \\ \cup &\ \{\,?\,\texttt{extends}\ \theta \mid \theta \in \mathrm{SType}_{TS}(\,BTV\,)\,\} \\ \cup &\ \{\,?\,\texttt{super}\ \theta \mid \theta \in \mathrm{SType}_{TS}(\,BTV\,)\,\}\end{aligned}.$$

Wildcard types cannot be used, explicitly. They are only inferred.

$\leq^*$ on $\mathrm{ExtSType}_{TS}(\, BTV\,)$

For $\theta, \theta' \in \mathrm{SType}_{TS}(\, BTV\,)$ with $\theta \leq^* \theta'$ holds:

- $\theta \leq^* {}^{?}\theta'$,
- ${}_{?}\theta \leq^* \theta'$, and
- ${}_{?}\theta \leq^* {}^{?}\theta'$.

Properties of type constructor applying

For $\theta \leq^* \theta'$:

- $\texttt{Cl<}\theta\texttt{>} \not\leq^* \texttt{Cl<}\theta'\texttt{>}$
- $\texttt{Cl<}\theta'\texttt{>} \not\leq^* \texttt{Cl<}\theta\texttt{>}$

Properties of type constructor applying

For $\theta \leq^* \theta'$:

- $\texttt{Cl<}\theta\texttt{>} \not\leq^* \texttt{Cl<}\theta'\texttt{>}$
- $\texttt{Cl<}\theta'\texttt{>} \not\leq^* \texttt{Cl<}\theta\texttt{>}$
- It holds

$$\texttt{? extends } \theta \leq^* \theta'$$

  but

$$\texttt{Cl<}\theta\texttt{>} \leq^* \texttt{Cl<? extends } \theta'\texttt{>}$$

Properties of type constructor applying

For $\theta \leq^* \theta'$:

- `Cl<`θ`>` $\not\leq^*$ `Cl<`θ'`>`
- `Cl<`θ'`>` $\not\leq^*$ `Cl<`θ`>`
- It holds

  $$\texttt{? extends } \theta \leq^* \theta'$$

  but

  $$\texttt{Cl<}\theta\texttt{>} \leq^* \texttt{Cl<? extends } \theta'\texttt{>}$$

- It holds

  $$(\texttt{? extends})\ \theta \leq^* \texttt{? super } \theta'$$

  but

  $$\texttt{CL<}\theta'\texttt{>} \leq^* \texttt{CL<? super } \theta\texttt{>}$$

Conclusion

- Formalization of the Java 5.0 type system
- Subtyping ordering on $\mathrm{SType}_{TS}(\,BTV\,)$ and $\mathrm{ExtSType}_{TS}(\,BTV\,)$
- Base for the type inference algorithm, which works, if
  - there is a subtyping ordering and
  - there is a type unification algorithm