Formalization of the Java 5.0 Type System

Martin Plümicke

University of Cooperative Education
Stuttgart/Horb

3. Mai 2007

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>
```

Example: Multiplication of matrices

```
class Matrix extends Vector<Vector<Integer>> {
    Vector<? extends Vector<? extends Integer>> mul(Matrix m) {
        Matrix ret = new Matrix();
        int i = 0;
        while(i <size()) {
            Vector<Integer> v1 = this.elementAt(i);
            Vector<Integer> v2 = new Vector<Integer>();
            int j = 0;
            while(j < v1.size()) {
                int erg = 0;
                int k = 0;
                while(k < v1.size()) {
                    erg = erg + v1.elementAt(k)
                        * m.elementAt(k).elementAt(j); k++; }
                v2.addElement(new Integer(erg)); j++; }
            ret.addElement(v2); i++; }
        return ret; }}
```

Purpose: Typless

```
class Matrix extends Vector<Vector<Integer>> {
    mul(m) {
        ret = new Matrix();
        i = 0;
        while(i <size()) {
            v1 = this.elementAt(i);
            v2 = new Vector<Integer>();
            j = 0;
            while(j < v1.size()) {
                erg = 0;
                k = 0;
                while(k < v1.size()) {
                    erg = erg + v1.elementAt(k)
                        * m.elementAt(k).elementAt(j); k++; }
                v2.addElement(new Integer(erg)); j++; }
            ret.addElement(v2); i++; }
        return ret; }}
```

The type inference algorithm

There is a generic algorithm for Java, parametrized by:

- the subtyping ordering
- the type unification on the corresponding ordering

The type inference algorithm

There is a generic algorithm for Java, parametrized by:

- the subtyping ordering
- the type unification on the corresponding ordering

Focus: Subtyping ordering on the type system of Java 5.0

Rank alphabet of classes and interfaces

Definition:

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

Rank alphabet of classes and interfaces

Definition:

- $\Theta = \Theta^{(n)}_{n \in \mathbb{N}}$ (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> { ... }
```

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

Rank alphabet of classes and interfaces

Definition:

- $\Theta = \Theta^{(n)}_{n \in \mathbb{N}}$ (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> { ... }
```

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

Bounded type variables

Definition:

- $\mathsf{SType}_{TS}(\ BTV\ ) \subseteq T_{\Theta}(\ TV\ )$
  (Java 5.0 simple types, mutually recursive definition)
- $\mathsf{I}(\ \mathsf{SType}_{TS}(\ BTV\ )\ )$ (set of intersections over simple types)
- $BTV = (BTV^{(ty)})_{ty \in \mathsf{I}(\ \mathsf{SType}_{TS}(\ BTV\ )\ )}$ (set of bounded type variables)
  Notation: $a|_{ty}$ means a is bounded by the (intersection) type ty.

Bounded type variables

Definition:

- $\mathsf{SType}_{TS}(\,BTV\,) \subseteq T_{\Theta}(\,TV\,)$
  (Java 5.0 simple types, mutually recursive definition)
- $\mathsf{I}(\,\mathsf{SType}_{TS}(\,BTV\,)\,)$ (set of intersections over simple types)
- $BTV = (BTV^{(ty)})_{ty \in \mathsf{I}(\,\mathsf{SType}_{TS}(\,BTV\,)\,)}$ (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 ( ... ) { ... } }
```

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

Type signature, type constructor

Definition:

- $(\mathrm{SType}_{TS}(BTV), TC)$ (type signature)
- TC ($(BTV)^*$–indexed set of *type constructors*)

Type signature, type constructor

Definition:

- $(\mathsf{SType}_{TS}(BTV),\ TC)$ (type signature)
- TC ($(BTV)^*$–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 $\mathrm{SType}_{TS}(\,BTV\,)$

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

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

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

  where
  - $CC(\ldots)$ denotes the capture conversion
  - $\leq^*$ is the subtyping ordering.

Beispiel:

```
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> { ... }
class BJ<c> extends B<BJ<c>> implements J<c> { ... }
```

- A<Integer> $\in \mathsf{SType}_{TS}(\mathit{BTV})$
- C<A<Boolean>, Boolean> $\in \mathsf{SType}_{TS}(\mathit{BTV})$
- D<BJ<Integer>, Integer> $\in \mathsf{SType}_{TS}(\mathit{BTV})$

Beispiel:

```
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> { ... }
class BJ<c> extends B<BJ<c>> implements J<c> { ... }
```

- A<Integer> $\in \mathsf{SType}_{TS}(\mathit{BTV})$
- C<A<Boolean>, Boolean> $\in \mathsf{SType}_{TS}(\mathit{BTV})$
- D<BJ<Integer>, Integer> $\in \mathsf{SType}_{TS}(\mathit{BTV})$
- C<Integer, Integer> $\notin \mathsf{SType}_{TS}(\mathit{BTV})$
- D<BJ<Integer>, Boolean> $\notin \mathsf{SType}_{TS}(\mathit{BTV})$

Subtyping

Abbreviations and auxillary definitions:

Instead of `A<? extends B>` we write

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

and instead of `C<? super D>` we write

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

Subtyping

Abbreviations and auxillary definitions:

Instead of `A<? extends B>` we write

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

and instead of `C<? super D>` we write

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

Implicit type variables with lower and upper bounds:

We denote a lower bound ty of a type variable T by ${}_{ty}|T$ and an upper bound ty' by $T|^{ty'}$.

Capture conversion

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

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

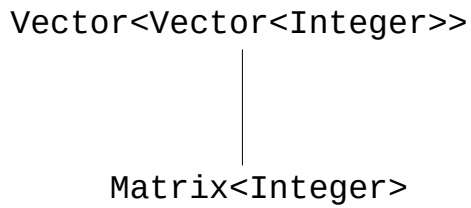
Subtyping ordering $\leq^*$

Reflexive and transitiv closure of

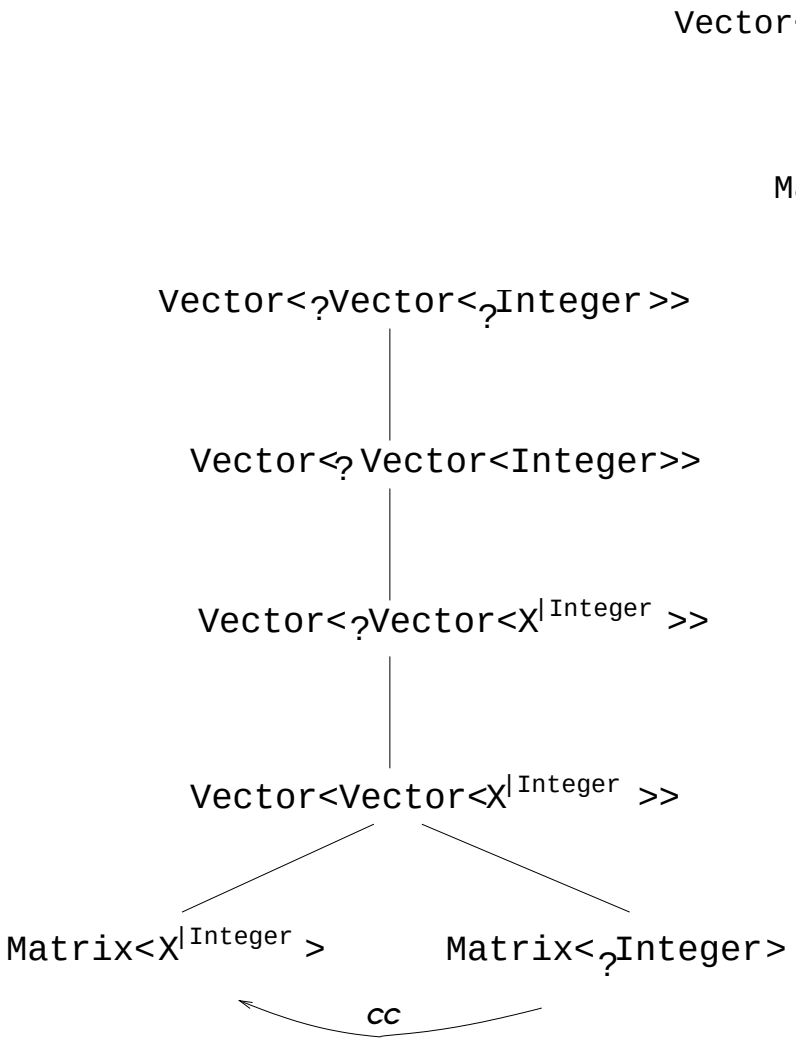
- if θ `extends` θ' then $\underline{\theta \leq^* \theta'}$.
- if $\theta_1 \leq^* \theta_2$ then $\underline{\sigma_1(\theta_1) \leq^* \sigma_2(\theta_2)}$, where for each type variable a of θ_2 holds $\sigma_1(a) = \sigma_2(a) \in \mathrm{SType}_{TS}(BTV)$ (soundness condition).
- $\underline{a \leq^* \theta'}$ for $a \in BTV^{(\theta_1 \& \ldots \& \theta_n)}$ where $\exists \theta_i : \theta_i \leq^* \theta'$.
- It holds $\underline{C\langle\theta_1, \ldots, \theta_n\rangle \leq^* C\langle\theta'_1, \ldots, \theta'_n\rangle}$ 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 $C\langle\overline{\theta}_1, \ldots, \overline{\theta}_n\rangle \leq^* C\langle\theta'_1, \ldots, \theta'_n\rangle$ follows with $C\langle\overline{\theta}_1, \ldots, \overline{\theta}_n\rangle = CC(C\langle\theta_1, \ldots, \theta_n\rangle)$: $\underline{C\langle\theta_1, \ldots, \theta_n\rangle \leq^* C\langle\theta'_1, \ldots, \theta'_n\rangle}$
- $\underline{T|^{(\theta_1 \& \ldots \& \theta_n)} \leq^* \theta_i}$ for any $1 \leqslant i \leqslant n$.

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

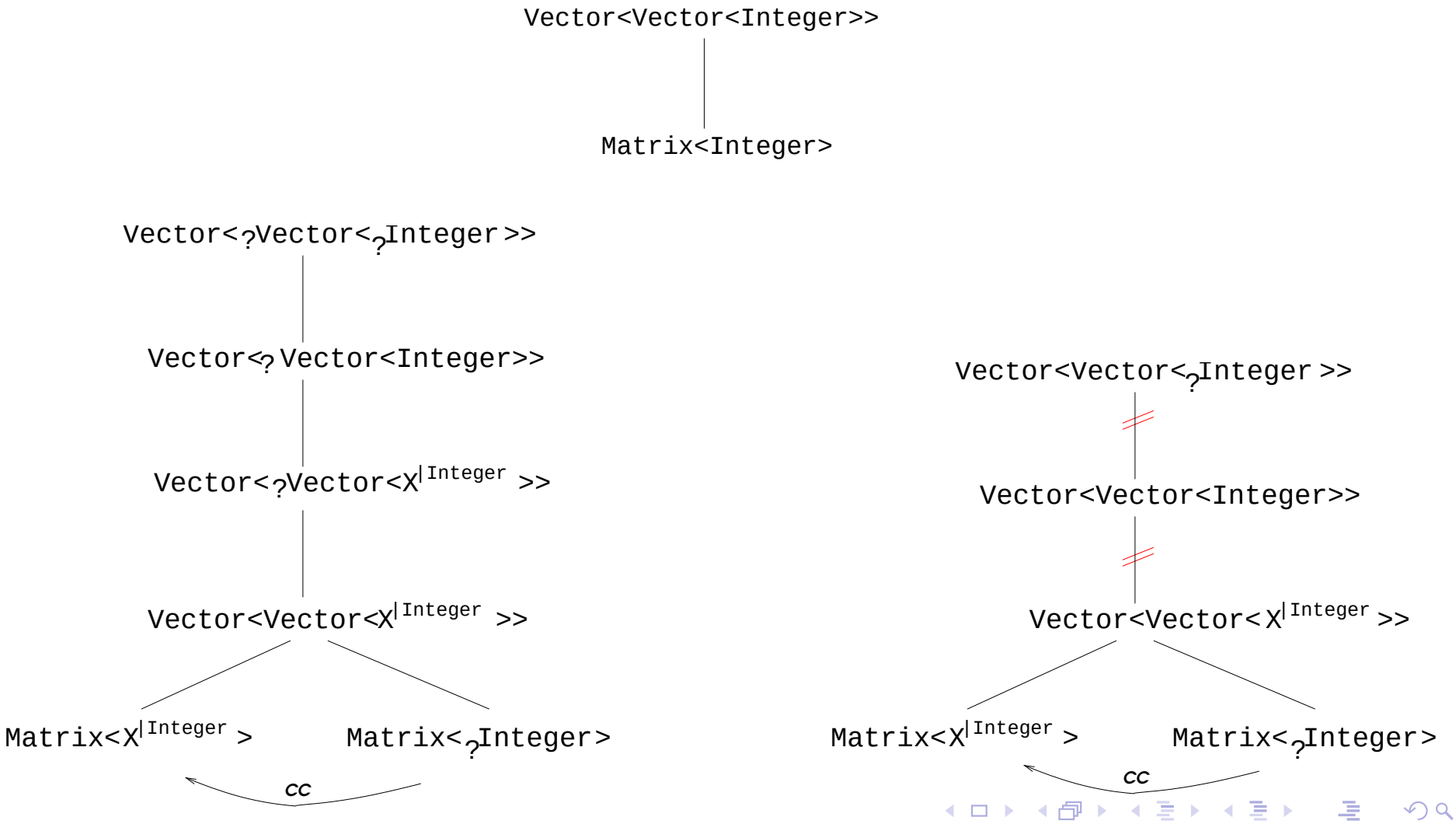
Example: Matrix<a> $\leq^*$ Vector<Vector<a>>



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



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

Introduction of wildcards:

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

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

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
}
```

Introduction of wildcards:

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

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

  `Super` $\not\leq^*$ `? extends Super`

  (as $\leq^*$ is not defined on wildcard types)

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

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

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

Extended simple types

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

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

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

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

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

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

Properties of type constructor applying

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

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

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

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

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

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

Conclusion

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