Java type unification with wildcards

Martin Plümicke

University of Cooperative Education
Stuttgart/Horb

29 June 2007

Overview

Motivation

Extensions of the Java 5.0 type–system

- parametrized types, type variables, type terms, wildcards
  e.g.

  `Vector<? extends AbstractList<? super Integer>>`

Motivation

Extensions of the Java 5.0 type–system

- parametrized types, type variables, type terms, wildcards
  e.g.

  `Vector<? extends AbstractList<? super Integer>>`

Complex typings

- Often it is not obvious, which are the *best* types for methods and variables
- Sometimes principal types in Java 5.0 are *intersection types*, which are not expressible (contradictive of writing re-usable code)

Motivation

Extensions of the Java 5.0 type–system

- parametrized types, type variables, type terms, wildcards
  e.g.

  `Vector<? extends AbstractList<? super Integer>>`

Complex typings

- Often it is not obvious, which are the *best* types for methods and variables
- Sometimes principal types in Java 5.0 are *intersection types*, which are not expressible (contradictive of writing re-usable code)

$\Longrightarrow$ Developing a type–inference–system, which determines principal types

Example: Multiplication of matrices

```
class Matrix extends Vector<Vector<Integer>> {
    Matrix 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; }}
```

System determines the principal typing(s)

$$\begin{array}{ll}\texttt{mul:} & \texttt{Matrix} \rightarrow \texttt{Matrix}\ \& \\ & \texttt{Matrix} \rightarrow \texttt{Vector<Vector<Integer>>} \\ & \& \ldots \& \\ & \texttt{Vector<? extends Vector<? extends Integer>>} \\ & \qquad\qquad\qquad \rightarrow \texttt{Vector<? super Vector<Integer>>}\end{array}$$

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; }}
```

Inheritance hierarchy

extends/implements relation: $\leq$ (declared by the `extends` resp. `implements` declarations

Example: `Stack<x>` $\leq$ `Vector<x>`
(declared by `class Stack<x> extends Vector<x>`)

subtyping relation: $\leq^*$ (ordering of the Java 5.0 type terms)

Example:
`Stack<Vector<Integer>>` $\leq^*$ `Vector<Vector<Integer>>`

Inheritance hierarchy cont.

Definition: Finite closure **FC**($\leq$)

- reflexive and transitive closure of relationships in the subtyping ordering, where in the left hand sides all arguments are type variables.

Inheritance hierarchy cont.

Definition: Finite closure **FC**($\leq$)

- reflexive and transitive closure of relationships in the subtyping ordering, where in the left hand sides all arguments are type variables.

Example: `Matrix<x>` $\leq^*$ `Vector<Vector<x>>`
(declared by `class Matrix<x> extends Vector<Vector<x>>`)

`myLi<b, a>` $\leq^*$ `List<a>`
(declared by `class myLi<b,a> extends List<a>`)

Inheritance hierarchy cont.

Definition: Finite closure **FC**($\leq$)

- reflexive and transitive closure of relationships in the subtyping ordering, where in the left hand sides all arguments are type variables.

Example: `Matrix<x>` $\leq^*$ `Vector<Vector<x>>`
(declared by `class Matrix<x> extends Vector<Vector<x>>`)

`myLi<b, a>` $\leq^*$ `List<a>`
(declared by `class myLi<b,a> extends List<a>`)

Lemma

The finite closure is a finite relation.

Wildcards

Subtyping relation: $\texttt{Integer} \leq^* \texttt{Number}$
$\texttt{Stack<a>} \leq^* \texttt{Vector<a>}$

It holds $\texttt{Stack<Integer>} \leq^* \texttt{Vector<Integer>}$

but $\texttt{Stack<Integer>} \not\leq^* \texttt{Vector<Number>}$

In the arguments no subtyping is allowed (soundness condition for the Java 5.0 type system)

Wildcards

Subtyping relation: `Integer` $\leq^*$ `Number`
`Stack<a>` $\leq^*$ `Vector<a>`

It holds `Stack<Integer>` $\leq^*$ `Vector<Integer>`

but `Stack<Integer>` $\not\leq^*$ `Vector<Number>`

In the arguments no subtyping is allowed (soundness condition for the Java 5.0 type system)

Introduction of wildcards

- `Stack<Integer>` $\leq^*$ `Vector<? extends Number>`
  `? extends Number`: all subtypes of `Number` are allowed
- `Stack<Number>` $\leq^*$ `Vector<? super Integer>`
  `? super Intger`: all supertypes of `Integer` are allowed

Abbreviation for wildcard–types

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

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

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

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

Type Unification problem

For two type terms θ_1 and θ_2 a substitution σ is demanded such that:

$$\sigma(\theta_1) \leq^* \sigma(\theta_2).$$

TEL [Smolka 1989]

- Type system without any restrictions
- Type unification algorithm is incomplete
- Open problem mentioned: infinte chains in the type term ordering
  For `List(a)` $\leq$ `myLi(a,b)` holds:

  $\texttt{List(a)} \leq \texttt{myLi(a,List(a))} \leq \texttt{myLi(a,myLi(a,List(a)))} \leq \ldots$

TEL [Smolka 1989]

- Type system without any restrictions
- Type unification algorithm is incomplete
- Open problem mentioned: infinte chains in the type term ordering
  For `List(a)` $\leq$ `myLi(a,b)` holds:

$$\texttt{List(a)} \leq \texttt{myLi(a,List(a))} \leq \texttt{myLi(a,myLi(a,List(a)))} \leq \ldots$$

For `nat` $\leq$ `int` holds

$$\{\, \texttt{nat} \lessdot \texttt{a}, \texttt{int} \lessdot \texttt{a} \,\} \Rightarrow \texttt{a} \mapsto \texttt{nat} \Rightarrow \{\, \texttt{int} \lessdot \texttt{nat} \,\} \Rightarrow \textbf{fail}$$

But there is a unifier $\texttt{a} \mapsto \texttt{int}$.

TEL [Smolka 1989]

- Type system without any restrictions
- Type unification algorithm is incomplete
- Open problem mentioned: infinte chains in the type term ordering
  For $\texttt{List(a)} \le \texttt{myLi(a,b)}$ holds:

$$\texttt{List(a)} \le \texttt{myLi(a,List(a))} \le \texttt{myLi(a,myLi(a,List(a)))} \le \ldots$$

For $\texttt{nat} \le \texttt{int}$ holds

$$\{\, \texttt{nat} \lessdot \mathsf{a}, \texttt{int} \lessdot \mathsf{a} \,\} \Rightarrow \mathsf{a} \mapsto \texttt{nat} \Rightarrow \{\, \texttt{int} \lessdot \texttt{nat} \,\} \Rightarrow \textbf{fail}$$

But there is a unifier $\mathsf{a} \mapsto \texttt{int}$.

The algorithm is incomplete even for types without infinte chains

[Hill, Topor 1992]

- subtype relationships only of type constructors with the same arity
- *most general type unifier (mgtu)* defined as an upper bound of different principal type unifiers

[Hill, Topor 1992]

- subtype relationships only of type constructors with the same arity
- *most general type unifier (mgtu)* defined as an upper bound of different principal type unifiers

For $\texttt{nat} \leq \texttt{int}$, $\texttt{neg} \leq \texttt{int}$ holds
The mgtu of $\{\, \texttt{nat} \lessdot \texttt{a},\ \texttt{neg} \lessdot \texttt{a} \,\}$ is $\{\, \texttt{a} \mapsto \texttt{int} \,\}$

[Hill, Topor 1992]

- subtype relationships only of type constructors with the same arity
- *most general type unifier (mgtu)* defined as an upper bound of different principal type unifiers

For $\texttt{nat} \leq \texttt{int}$, $\texttt{neg} \leq \texttt{int}$ holds
The mgtu of $\{ \texttt{nat} \lessdot \texttt{a}, \texttt{neg} \lessdot \texttt{a} \}$ is $\{ \texttt{a} \mapsto \texttt{int} \}$

Extension: $\texttt{int} \leq \texttt{index}$ and $\texttt{int} \leq \texttt{expr}$:
There are three unifiers $\texttt{a} \mapsto \texttt{int}$, $\texttt{a} \mapsto \texttt{index}$, and $\texttt{a} \mapsto \texttt{expr}$, but none of them is a mgtu.

[Hill, Topor 1992]

- subtype relationships only of type constructors with the same arity
- *most general type unifier (mgtu)* defined as an upper bound of different principal type unifiers

For $\texttt{nat} \leq \texttt{int}$, $\texttt{neg} \leq \texttt{int}$ holds
The mgtu of $\{\, \texttt{nat} \lessdot \texttt{a}, \texttt{neg} \lessdot \texttt{a} \,\}$ is $\{\, \texttt{a} \mapsto \texttt{int} \,\}$

Extension: $\texttt{int} \leq \texttt{index}$ and $\texttt{int} \leq \texttt{expr}$:
There are three unifiers $\texttt{a} \mapsto \texttt{int}$, $\texttt{a} \mapsto \texttt{index}$, and $\texttt{a} \mapsto \texttt{expr}$, but none of them is a mgtu.

In general there is no mgtu in the sense of [Hill, Topor 1992].

PROTOS-L [Beierle 1995]

- no subtype relationship between polymorphic type constructors
- type unification algorithm complete
- unification problem indeed not unitary, but finitary
- the algorithm is also complete for the type system of [Hill, Topor 1992]

PROTOS-L [Beierle 1995]

- no subtype relationship between polymorphic type constructors
- type unification algorithm complete
- unification problem indeed not unitary, but finitary
- the algorithm is also complete for the type system of [Hill, Topor 1992]

For $\texttt{nat} \leq \texttt{int}$, $\texttt{neg} \leq \texttt{int}$, $\texttt{int} \leq \texttt{index}$, $\texttt{int} \leq \texttt{expr}$ and

$$\{\, \texttt{nat} \lessdot \texttt{a}, \texttt{neg} \lessdot \texttt{a} \,\}$$

there are three general unifiers

$$\{\, \texttt{a} \mapsto \texttt{int} \,\},\ \{\, \texttt{a} \mapsto \texttt{index} \,\},\ \text{and}\ \{\, \texttt{a} \mapsto \texttt{expr} \,\}.$$

PROTOS-L [Beierle 1995]

- no subtype relationship between polymorphic type constructors
- type unification algorithm complete
- unification problem indeed not unitary, but finitary
- the algorithm is also complete for the type system of [Hill, Topor 1992]

For $\texttt{nat} \leq \texttt{int}$, $\texttt{neg} \leq \texttt{int}$, $\texttt{int} \leq \texttt{index}$, $\texttt{int} \leq \texttt{expr}$ and

$$\{\, \texttt{nat} \lessdot \texttt{a}, \texttt{neg} \lessdot \texttt{a} \,\}$$

there are three general unifiers

$$\{\, \texttt{a} \mapsto \texttt{int} \,\},\ \{\, \texttt{a} \mapsto \texttt{index} \,\},\ \text{and}\ \{\, \texttt{a} \mapsto \texttt{expr} \,\}.$$

The algorithm do not work on subtype relationships where the constructors have different arities.

Java type unification [Plümicke 2004, Unif'04, Cork]

- no wildcards
- no subtyping in the arguments of the type terms (soundness condition)

Java type unification [Plümicke 2004, Unif'04, Cork]

- no wildcards
- no subtyping in the arguments of the type terms (soundness condition)

For $\texttt{myLi<b, a>} \leq \texttt{List<a>}$ and $\{\, \texttt{myLi<Integer, a>} \lessdot \texttt{List<Boolean>} \,\}$ the general unifier

$$\{\, \texttt{a} \mapsto \texttt{Boolean} \,\}$$

is determined.

Java type unification [Plümicke 2004, Unif'04, Cork]

- no wildcards
- no subtyping in the arguments of the type terms (soundness condition)

For $\texttt{myLi<b, a>} \leq \texttt{List<a>}$ and $\{\, \texttt{myLi<Integer, a>} \lessdot \texttt{List<Boolean>} \,\}$ the general unifier

$$\{\, \texttt{a} \mapsto \texttt{Boolean} \,\}$$

is determined. For

$$\{\, \texttt{myLi<Integer, Integer>} \lessdot \texttt{List<Number>} \,\}$$

the algorithm fails, as indeed $\texttt{Integer} \leq \texttt{Number}$, but subtyping in the arguments is prohibited.

Java type unification [Plümicke 2004, Unif'04, Cork]

- no wildcards
- no subtyping in the arguments of the type terms (soundness condition)

For $\texttt{myLi<b,a>} \leq \texttt{List<a>}$ and $\{\, \texttt{myLi<Integer,a>} \lessdot \texttt{List<Boolean>} \,\}$ the general unifier

$$\{\, \texttt{a} \mapsto \texttt{Boolean} \,\}$$

is determined. For

$$\{\, \texttt{myLi<Integer,Integer>} \lessdot \texttt{List<Number>} \,\}$$

the algorithm fails, as indeed $\texttt{Integer} \leq \texttt{Number}$, but subtyping in the arguments is prohibited.

No infinte chains appears.

Base of Hindley/Milner approach: (Type) unification algorithm [Martelli, Montanari 1982]

(reduce) $$\frac{Eq \cup \{\, C\text{<}\theta_1, \ldots, \theta_n\text{>} \doteq C\text{<}\theta'_1, \ldots, \theta'_n\text{>} \,\}}{Eq \cup \{\, \theta_1 \doteq \theta'_1, \ldots, \theta_n \doteq \theta'_n \,\}}$$

(erase) $$\frac{Eq \cup \{\, \theta \doteq \theta' \,\}}{Eq} \quad \theta = \theta'$$

(swap) $$\frac{Eq \cup \{\, \theta \doteq a \,\}}{Eq \cup \{\, a \doteq \theta \,\}} \quad a \in TV$$

(subst) $$\frac{Eq \cup \{\, a \doteq \theta \,\}}{Eq[a \mapsto \theta] \cup \{\, a \doteq \theta \,\}}$$ a occurs in Eq but not in θ

Type unification algorithm for Java 5.0 type terms without wildcards [Plümicke 2004, Unif'04, Cork]

(adapt) $$\frac{Eq \cup \{\, D\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}{Eq \cup \{\, D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}[a_i \mapsto \theta_i \mid 1 \leqslant i \leqslant n] \lessdot D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with

- $(D\texttt{<}a_1, \ldots, a_n\texttt{>} \leq^* D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}) \in \mathbf{FC}(\ \leq\ )$

Type unification algorithm for Java 5.0 type terms without wildcards [Plümicke 2004, Unif'04, Cork]

(adapt)
$$\frac{Eq \cup \{\, D\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}{Eq \cup \{\, D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}[a_i \mapsto \theta_i \mid 1 \leqslant i \leqslant n] \lessdot D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with

- $(D\texttt{<}a_1, \ldots, a_n\texttt{>} \leq^* D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}) \in \mathbf{FC}(\leq)$

(reduce1)
$$\frac{Eq \cup \{\, C\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot D\texttt{<}\theta'_1, \ldots, \theta'_n\texttt{>} \,\}}{Eq \cup \{\, \theta_{\pi(1)} \doteq \theta'_1, \ldots, \theta_{\pi(n)} \doteq \theta'_n \,\}}$$

where

- $C\texttt{<}a_1, \ldots, a_n\texttt{>} \leq^* D\texttt{<}a_{\pi(1)}, \ldots, a_{\pi(n)}\texttt{>}$
- $\{\, a_1, \ldots, a_n \,\} \subseteq TV$
- π is a permutation

Type unification algorithm for Java 5.0 type terms without wildcards [Plümicke 2004, Unif'04, Cork]

(adapt)
$$\frac{Eq \cup \{\, D\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}{Eq \cup \{\, D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}[a_i \mapsto \theta_i \mid 1 \leqslant i \leqslant n] \lessdot D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}$$
where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with

- $(D\text{<}a_1, \ldots, a_n\text{>} \leq^* D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}) \in \mathbf{FC}(\ \leq\ )$

(reduce1)
$$\frac{Eq \cup \{\, C\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot D\text{<}\theta'_1, \ldots, \theta'_n\text{>} \,\}}{Eq \cup \{\, \theta_{\pi(1)} \doteq \theta'_1, \ldots, \theta_{\pi(n)} \doteq \theta'_n \,\}}$$
where

- $C\text{<}a_1, \ldots, a_n\text{>} \leq^* D\text{<}a_{\pi(1)}, \ldots, a_{\pi(n)}\text{>}$
- $\{\, a_1, \ldots, a_n \,\} \subseteq TV$
- π is a permutation

(reduce2)
$$\frac{Eq \cup \{\, C\text{<}\theta_1, \ldots, \theta_n\text{>} \doteq C\text{<}\theta'_1, \ldots, \theta'_n\text{>} \,\}}{Eq \cup \{\, \theta_1 \doteq \theta'_1, \ldots, \theta_n \doteq \theta'_n \,\}}$$

(erase)
$$\frac{Eq \cup \{\, \theta \doteq \theta' \,\}}{Eq} \qquad \theta = \theta'$$

(swap)
$$\frac{Eq \cup \{\, \theta \doteq a \,\}}{Eq \cup \{\, a \doteq \theta \,\}} \qquad a \in TV$$

(subst)
$$\frac{Eq \cup \{\, a \doteq \theta \,\}}{Eq[a \mapsto \theta] \cup \{\, a \doteq \theta \,\}}$$
where

- a occurs in Eq but not in θ

Type unification rules

$$(\mathrm{redUp})\,\frac{Eq \cup \{\, \theta \lessdot {}^{?}\theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}} \qquad (\mathrm{redUpLow})\,\frac{Eq \cup \{\, {}_{?}\theta \lessdot {}^{?}\theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}} \qquad (\mathrm{redLow})\,\frac{Eq \cup \{\, {}_{?}\theta \lessdot \theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}}$$

Wildcards in outermost position

Type unification rules

$$(\mathrm{redUp})\ \frac{Eq \cup \{\, \theta \lessdot {}^{?}\theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}} \qquad (\mathrm{redUpLow})\ \frac{Eq \cup \{\, {}_{?}\theta \lessdot {}^{?}\theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}} \qquad (\mathrm{redLow})\ \frac{Eq \cup \{\, {}_{?}\theta \lessdot \theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}}$$

Wildcards in outermost position

$$(\mathrm{red1}) \quad \frac{Eq \cup \{\, C{<}\theta_1, \ldots, \theta_n{>} \lessdot D{<}\theta'_1, \ldots, \theta'_n{>} \,\}}{Eq \cup \{\, \theta_{\pi(1)} \lessdot_{?} \theta'_1, \ldots, \theta_{\pi(n)} \lessdot_{?} \theta'_n \,\}}$$

where

- $C{<}a_1, \ldots, a_n{>} \leq^* D{<}a_{\pi(1)}, \ldots, a_{\pi(n)}{>}$
- $\{\, a_1, \ldots, a_n \,\} \subseteq BTV$
- π is a permutation

Reduce rule for outermost type constructor

Type unification rules

$$\text{(redUp)}\,\frac{Eq \cup \{\, \theta \lessdot {}^?\theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}} \qquad \text{(redUpLow)}\,\frac{Eq \cup \{\, {}_?\theta \lessdot {}^?\theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}} \qquad \text{(redLow)}\,\frac{Eq \cup \{\, {}_?\theta \lessdot \theta' \,\}}{Eq \cup \{\, \theta \lessdot \theta' \,\}}$$

Wildcards in outermost position

$$\text{(red1)} \quad \frac{Eq \cup \{\, C\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot D\texttt{<}\theta'_1, \ldots, \theta'_n\texttt{>} \,\}}{Eq \cup \{\, \theta_{\pi(1)} \lessdot_? \theta'_1, \ldots, \theta_{\pi(n)} \lessdot_? \theta'_n \,\}}$$

Reduce rule for outermost type constructor

where

- $C\texttt{<}a_1, \ldots, a_n\texttt{>} \leq^* D\texttt{<}a_{\pi(1)}, \ldots, a_{\pi(n)}\texttt{>}$
- $\{\, a_1, \ldots, a_n \,\} \subseteq BTV$
- π is a permutation

$$\text{(redExt)}\,\frac{Eq \cup \{\, X\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot_? {}_?Y\texttt{<}\theta'_1, \ldots, \theta'_n\texttt{>} \,\}}{Eq \cup \{\, \theta_{\pi(1)} \lessdot_? \theta'_1, \ldots, \theta_{\pi(n)} \lessdot_? \theta'_n \,\}}$$

Reduce rule for extends wildcards

where

- ${}_?Y\texttt{<}a_{\pi(1)}, \ldots, a_{\pi(n)}\texttt{>} \in$ **grArg**$(\, X\texttt{<}a_1, \ldots, a_n\texttt{>} \,)$
- $\{\, a_1, \ldots, a_n \,\} \subseteq BTV$

Type unification rules

(redSup) $$\frac{Eq \cup \{\, X\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot_? {}^{?}Y\texttt{<}\theta'_1, \ldots, \theta'_n\texttt{>} \,\}}{Eq \cup \{\, \theta'_1 \lessdot_? \theta_{\pi(1)}, \ldots, \theta'_n \lessdot_? \theta_{\pi(n)} \,\}}$$

where

- ${}^{?}Y\texttt{<}a_{\pi(1)}, \ldots, a_{\pi(n)}\texttt{>} \in$ $\mathbf{grArg}(\, X\texttt{<}a_1, \ldots, a_n\texttt{>}\,)$
- $\{\, a_1, \ldots, a_n \,\} \subseteq BTV$
- π is a permutation

Reduce rule for super wildcards

Type unification rules

(redSup) $$\frac{Eq \cup \{\, X\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot_? {}^{?}Y\text{<}\theta'_1, \ldots, \theta'_n\text{>} \,\}}{Eq \cup \{\, \theta'_1 \lessdot_? \theta_{\pi(1)}, \ldots, \theta'_n \lessdot_? \theta_{\pi(n)} \,\}}$$

Reduce rule for super wildcards

where

- ${}^{?}Y\text{<}a_{\pi(1)}, \ldots, a_{\pi(n)}\text{>} \in$ **grArg**$(\, X\text{<}a_1, \ldots, a_n\text{>} \,)$
- $\{\, a_1, \ldots, a_n \,\} \subseteq BTV$
- π is a permutation

(redEq) $$\frac{Eq \cup \{\, X\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot_? X\text{<}\theta'_1, \ldots, \theta'_n\text{>} \,\}}{Eq \cup \{\, \theta_{\pi(1)} \doteq \theta'_1, \ldots, \theta_{\pi(n)} \doteq \theta'_n \,\}}$$

Reduce rule for equal type constructors

Type unification rules

$$\text{(redSup)}\quad \frac{Eq \cup \{\, X<\theta_1, \ldots, \theta_n> \lessdot_? {}^{?}Y<\theta'_1, \ldots, \theta'_n> \,\}}{Eq \cup \{\, \theta'_1 \lessdot_? \theta_{\pi(1)}, \ldots, \theta'_n \lessdot_? \theta_{\pi(n)} \,\}}$$

where

- ${}^{?}Y<a_{\pi(1)}, \ldots, a_{\pi(n)}> \in$ **grArg**$(\, X<a_1, \ldots, a_n> \,)$
- $\{\, a_1, \ldots, a_n \,\} \subseteq BTV$
- π is a permutation

Reduce rule for super wildcards

$$\text{(redEq)}\quad \frac{Eq \cup \{\, X<\theta_1, \ldots, \theta_n> \lessdot_? X<\theta'_1, \ldots, \theta'_n> \,\}}{Eq \cup \{\, \theta_{\pi(1)} \doteq \theta'_1, \ldots, \theta_{\pi(n)} \doteq \theta'_n \,\}}$$

Reduce rule for equal type constructors

$$\text{(reduce2)}\quad \frac{Eq \cup \{\, C<\theta_1, \ldots, \theta_n> \doteq C<\theta'_1, \ldots, \theta'_n> \,\}}{Eq \cup \{\, \theta_1 \doteq \theta'_1, \ldots, \theta_n \doteq \theta'_n \,\}}$$

Original reduce rule

Type unification rules

(adapt)

$$\frac{Eq \cup \{\, D\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}{Eq \cup \{\, D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}[a_i \mapsto CC(\,\theta_i\,) \mid 1 \leqslant i \leqslant n] \lessdot D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with outermost adapt rule

- $(D\text{<}a_1, \ldots, a_n\text{>} \leq^* D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}) \in \mathbf{FC}(\, \leq \,)$

Type unification rules

(adapt)

$$\frac{Eq \cup \{\, D\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}{Eq \cup \{\, D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}[a_i \mapsto CC(\,\theta_i\,) \mid 1 \leqslant i \leqslant n] \lessdot D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with outermost adapt rule

- $(D\texttt{<}a_1, \ldots, a_n\texttt{>} \leq^* D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}) \in \mathbf{FC}(\,\leq\,)$

(adaptExt)

$$\frac{Eq \cup \{\, D\texttt{<}\theta_1, \ldots, \theta_n\texttt{>} \lessdot_{?\,?} D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}{Eq \cup \{\, D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}[a_i \mapsto CC(\,\theta_i\,) \mid 1 \leqslant i \leqslant n] \lessdot_{?\,?} D'\texttt{<}\theta'_1, \ldots, \theta'_m\texttt{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with adapt rule: extends wildcard

- $(D\texttt{<}a_1, \ldots, a_n\texttt{>} \leq^* D'\texttt{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\texttt{>}) \in \mathbf{FC}(\,\leq\,)$

Type unification rules

(adapt)

$$\frac{Eq \cup \{\, D\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}{Eq \cup \{\, D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}[a_i \mapsto CC(\,\theta_i\,) \mid 1 \leqslant i \leqslant n] \lessdot D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with outermost adapt rule

- $(D\text{<}a_1, \ldots, a_n\text{>} \leq^* D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}) \in \mathbf{FC}(\,\leq\,)$

(adaptExt)

$$\frac{Eq \cup \{\, D\text{<}\theta_1, \ldots, \theta_n\text{>} \lessdot_{?\,?} D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}{Eq \cup \{\, D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}[a_i \mapsto CC(\,\theta_i\,) \mid 1 \leqslant i \leqslant n] \lessdot_{?\,?} D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with adapt rule: extends wildcard

- $(D\text{<}a_1, \ldots, a_n\text{>} \leq^* D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}) \in \mathbf{FC}(\,\leq\,)$

(adaptSup)

$$\frac{Eq \cup \{\, D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \lessdot_{?}{}^{?} D\text{<}\theta_1, \ldots, \theta_n\text{>} \,\}}{Eq \cup \{\, D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}[a_i \mapsto CC(\,\theta_i\,) \mid 1 \leqslant i \leqslant n] \lessdot_{?\,?} D'\text{<}\theta'_1, \ldots, \theta'_m\text{>} \,\}}$$

where there are $\overline{\theta}'_1, \ldots, \overline{\theta}'_m$ with adapt rule: super wildcard

- $(D\text{<}a_1, \ldots, a_n\text{>} \leq^* D'\text{<}\overline{\theta}'_1, \ldots, \overline{\theta}'_m\text{>}) \in \mathbf{FC}(\,\leq\,)$

Type unification rules

(erase1) $$\frac{Eq \cup \{\, \theta \lessdot \theta' \,\}}{Eq} \qquad \theta \leq^* \theta'$$

(erase2) $$\frac{Eq \cup \{\, \theta \lessdot_? \theta' \,\}}{Eq} \qquad \theta' \in \mathbf{grArg}(\, \theta \,)$$

(erase3) $$\frac{Eq \cup \{\, \theta \doteq \theta' \,\}}{Eq} \qquad \theta = \theta'$$

Type unification rules

(erase1)
$$\frac{Eq \cup \{\, \theta \lessdot \theta' \,\}}{Eq} \qquad \theta \leq^* \theta'$$

(erase2)
$$\frac{Eq \cup \{\, \theta \lessdot_? \theta' \,\}}{Eq} \qquad \theta' \in \mathbf{grArg}(\, \theta \,)$$

(erase3)
$$\frac{Eq \cup \{\, \theta \doteq \theta' \,\}}{Eq} \qquad \theta = \theta'$$

(swap)
$$\frac{Eq \cup \{\, \theta \doteq a \,\}}{Eq \cup \{\, a \doteq \theta \,\}} \qquad \theta \notin BTV, \;\; a \in BTV$$

(subst)
$$\frac{Eq' \cup \{\, a \doteq \theta \,\}}{Eq'[a \mapsto \theta] \cup \{\, a \doteq \theta \,\}} \qquad a \text{ occurs in } Eq' \text{ but not in } \theta$$

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.
2. For each pair $a \lessdot \theta$ and $a \lessdot_? \theta$ for all subtypes $\overline{\theta}$ of θ, constructed by pattern–matching with elements from **FC**$(\leq^* , BTV)$, pairs $a \doteq \overline{\theta}$ are built.

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.
2. For each pair $a \lessdot \theta$ and $a \lessdot_? \theta$ for all subtypes $\overline{\theta}$ of θ, constructed by pattern–matching with elements from **FC**$(\leq^* , BTV)$, pairs $a \doteq \overline{\theta}$ are built.
3. For each pair $\theta \lessdot a$ and $\theta \lessdot_? a$ for all supertypes θ' of θ, constructed by pattern–matching with elements from **FC**$(\leq^* , BTV)$, pairs $a \doteq \theta'$ are built.

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.
2. For each pair $a \lessdot \theta$ and $a \lessdot_? \theta$ for all subtypes $\overline{\theta}$ of θ, constructed by pattern–matching with elements from **FC**$(\leq^* , BTV)$, pairs $a \doteq \overline{\theta}$ are built.
3. For each pair $\theta \lessdot a$ and $\theta \lessdot_? a$ for all supertypes θ' of θ, constructed by pattern–matching with elements from **FC**$(\leq^* , BTV)$, pairs $a \doteq \theta'$ are built.
4. The cartesian product of the sets from step 2 and 3 is built.

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.
2. For each pair $a \lessdot \theta$ and $a \lessdot_? \theta$ for all subtypes $\overline{\theta}$ of θ, constructed by pattern–matching with elements from **FC**($\leq^*$, BTV), pairs $a \doteq \overline{\theta}$ are built.
3. For each pair $\theta \lessdot a$ and $\theta \lessdot_? a$ for all supertypes θ' of θ, constructed by pattern–matching with elements from **FC**($\leq^*$, BTV), pairs $a \doteq \theta'$ are built.
4. The cartesian product of the sets from step 2 and 3 is built.
5. Application of the *subst* rule

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.
2. For each pair $a \lessdot \theta$ and $a \lessdot_? \theta$ for all subtypes $\overline{\theta}$ of θ, constructed by pattern–matching with elements from $\mathbf{FC}(\ \leq^*, BTV\ )$, pairs $a \doteq \overline{\theta}$ are built.
3. For each pair $\theta \lessdot a$ and $\theta \lessdot_? a$ for all supertypes θ' of θ, constructed by pattern–matching with elements from $\mathbf{FC}(\ \leq^*, BTV\ )$, pairs $a \doteq \theta'$ are built.
4. The cartesian product of the sets from step 2 and 3 is built.
5. Application of the *subst* rule
6. For all changed sets of type terms start again with step 1.

Type unification algorithm with wildcards

1. Repeated application of the *reduce* rules, the *erase* rules, the *swap* rule, and the *adapt* rules.
2. For each pair $a \lessdot \theta$ and $a \lessdot_? \theta$ for all subtypes $\overline{\theta}$ of θ, constructed by pattern–matching with elements from $\mathbf{FC}(\ \leq^*, BTV\ )$, pairs $a \doteq \overline{\theta}$ are built.
3. For each pair $\theta \lessdot a$ and $\theta \lessdot_? a$ for all supertypes θ' of θ, constructed by pattern–matching with elements from $\mathbf{FC}(\ \leq^*, BTV\ )$, pairs $a \doteq \theta'$ are built.
4. The cartesian product of the sets from step 2 and 3 is built.
5. Application of the *subst* rule
6. For all changed sets of type terms start again with step 1.
7. Summerize all results.

Example

Subtyping relation:

$\texttt{Integer} \leq^* \texttt{Number}$

$\texttt{Stack<a>} \leq^* \texttt{Vector<a>} \leq^* \texttt{AbstractList<a>} \leq^* \texttt{List<a>}$

Example

Subtyping relation:

$\texttt{Integer} \leq^* \texttt{Number}$

$\texttt{Stack<a>} \leq^* \texttt{Vector<a>} \leq^* \texttt{AbstractList<a>} \leq^* \texttt{List<a>}$

Application of the algorithm:

$\{\, (\texttt{Stack<a>} \lessdot \texttt{Vector<}{}_?\texttt{Number>}), (\texttt{AbstractList<Integer>} \lessdot \texttt{List<a>}) \,\}$

Example

Subtyping relation:

$$\texttt{Integer} \leq^* \texttt{Number}$$

$$\texttt{Stack<a>} \leq^* \texttt{Vector<a>} \leq^* \texttt{AbstractList<a>} \leq^* \texttt{List<a>}$$

Application of the algorithm:

$$\{\, (\texttt{Stack<a>} \lessdot \texttt{Vector<}{}_{?}\texttt{Number>}), (\texttt{AbstractList<Integer>} \lessdot \texttt{List<a>})$$

$$\overset{(\mathit{red}1)}{\Longrightarrow} \{\, \texttt{a} \lessdot_{?} {}_{?}\texttt{Number},\ \texttt{Integer} \lessdot_{?} \texttt{a} \,\}$$

$$\begin{aligned}
\overset{2./3./4.}{\Longrightarrow} \{\, & \{\, \texttt{a} \doteq {}_{?}\texttt{Number}, \texttt{a} \doteq \texttt{Integer} \,\}, \{\, \texttt{a} \doteq {}_{?}\texttt{Number}, \texttt{a} \doteq {}_{?}\texttt{Number} \,\}, \\
& \{\, \texttt{a} \doteq {}_{?}\texttt{Number}, \texttt{a} \doteq {}_{?}\texttt{Integer} \,\}, \{\, \texttt{a} \doteq {}_{?}\texttt{Number}, \texttt{a} \doteq {}^{?}\texttt{Integer} \,\}, \\
& \{\, \texttt{a} \doteq \texttt{Number}, \texttt{a} \doteq \texttt{Integer} \,\}, \{\, \texttt{a} \doteq \texttt{Number}, \texttt{a} \doteq {}_{?}\texttt{Number} \,\}, \\
& \{\, \texttt{a} \doteq \texttt{Number}, \texttt{a} \doteq {}_{?}\texttt{Integer} \,\}, \{\, \texttt{a} \doteq \texttt{Number}, \texttt{a} \doteq {}^{?}\texttt{Integer} \,\}, \\
& \{\, \texttt{a} \doteq {}_{?}\texttt{Integer}, \texttt{a} \doteq \texttt{Integer} \,\}, \{\, \texttt{a} \doteq {}_{?}\texttt{Integer}, \texttt{a} \doteq {}_{?}\texttt{Number} \,\}, \\
& \{\, \texttt{a} \doteq {}_{?}\texttt{Integer}, \texttt{a} \doteq {}_{?}\texttt{Integer} \,\}, \{\, \texttt{a} \doteq {}_{?}\texttt{Integer}, \texttt{a} \doteq {}^{?}\texttt{Integer} \,\}, \\
& \{\, \texttt{a} \doteq \texttt{Integer}, \texttt{a} \doteq \texttt{Integer} \,\}, \{\, \texttt{a} \doteq \texttt{Integer}, \texttt{a} \doteq {}_{?}\texttt{Number} \,\}, \\
& \{\, \texttt{a} \doteq \texttt{Integer}, \texttt{a} \doteq {}_{?}\texttt{Integer} \,\}, \{\, \texttt{a} \doteq \texttt{Integer}, \texttt{a} \doteq {}^{?}\texttt{Integer} \,\} \,\}
\end{aligned}$$

Example cont.

$$\overset{5.\mathit{step}\ (\mathit{subst})}{\Longrightarrow}$$

$$\begin{array}{l}
\{\, \{\, \mathtt{Integer} \doteq {}_{?}\mathtt{Number}, \mathtt{a} \doteq \mathtt{Integer} \,\}, \underline{\{\, {}_{?}\mathtt{Number} \doteq {}_{?}\mathtt{Number}, \mathtt{a} \doteq {}_{?}\mathtt{Number} \,\}}, \\
\quad \{\, {}_{?}\mathtt{Integer} \doteq {}_{?}\mathtt{Number}, \mathtt{a} \doteq {}_{?}\mathtt{Integer} \,\}, \{\, {}^{?}\mathtt{Integer} \doteq {}_{?}\mathtt{Number}, \mathtt{a} \doteq {}^{?}\mathtt{Integer} \,\}, \\
\quad \{\, \mathtt{Integer} \doteq \mathtt{Number}, \mathtt{a} \doteq \mathtt{Integer} \,\}, \{\, {}_{?}\mathtt{Number} \doteq \mathtt{Number}, \mathtt{a} \doteq {}_{?}\mathtt{Number} \,\}, \\
\quad \{\, {}_{?}\mathtt{Integer} \doteq \mathtt{Number}, \mathtt{a} \doteq {}_{?}\mathtt{Integer} \,\}, \{\, {}^{?}\mathtt{Integer} \doteq \mathtt{Number}, \mathtt{a} \doteq {}^{?}\mathtt{Integer} \,\}, \\
\quad \{\, \mathtt{Integer} \doteq {}_{?}\mathtt{Integer}, \mathtt{a} \doteq \mathtt{Integer} \,\}, \{\, {}_{?}\mathtt{Number} \doteq {}_{?}\mathtt{Integer}, \mathtt{a} \doteq {}_{?}\mathtt{Number} \,\}, \\
\quad \underline{\{\, {}_{?}\mathtt{Integer} \doteq {}_{?}\mathtt{Integer}, \mathtt{a} \doteq {}_{?}\mathtt{Integer} \,\}}, \{\, {}^{?}\mathtt{Integer} \doteq {}_{?}\mathtt{Integer}, \mathtt{a} \doteq {}^{?}\mathtt{Integer} \\
\quad \underline{\{\, \mathtt{Integer} \doteq \mathtt{Integer}, \mathtt{a} \doteq \mathtt{Integer} \,\}}, \{\, {}_{?}\mathtt{Number} \doteq \mathtt{Integer}, \mathtt{a} \doteq {}_{?}\mathtt{Number} \,\}, \\
\quad \{\, {}_{?}\mathtt{Integer} \doteq \mathtt{Integer}, \mathtt{a} \doteq {}_{?}\mathtt{Integer} \,\} \{\, {}^{?}\mathtt{Integer} \doteq \mathtt{Integer}, \mathtt{a} \doteq {}^{?}\mathtt{Integer} \,\} \,\}
\end{array}$$

$$\overset{(\mathit{erase}3)}{\Longrightarrow} \{\, \{\, \mathtt{a} \mapsto {}_{?}\mathtt{Number} \,\}, \{\, \mathtt{a} \mapsto {}_{?}\mathtt{Integer} \,\}, \{\, \mathtt{a} \mapsto \mathtt{Integer} \,\} \,\}$$

Example: Infinite chains

Subtyping relation:

$$\texttt{myLi<b, a>} \leq \texttt{List<a>}$$

Example: Infinite chains

Subtyping relation:

$$\texttt{myLi<b}, \texttt{a>} \leq \texttt{List<a>}$$

There is an infinte chain:

$$\ldots \leq^* \texttt{myLi<}_{?}\texttt{myLi<}_{?}\texttt{List<a>}, \texttt{a>}, \texttt{a>} \leq^* \texttt{myLi<}_{?}\texttt{List<a>}, \texttt{a>} \leq^* \texttt{List<a>}$$

Example: Infinite chains

Subtyping relation:

$$\texttt{myLi<b, a>} \leq \texttt{List<a>}$$

There is an infinte chain:

$$\ldots \leq^* \texttt{myLi<}{}_?\texttt{myLi<}{}_?\texttt{List<a>, a>, a>} \leq^* \texttt{myLi<}{}_?\texttt{List<a>, a>} \leq^* \texttt{List<a>}$$

Application of the algorithm:

$$\begin{array}{ll} & \{\, \texttt{List<x>} \lessdot \texttt{List<}{}_?\texttt{List<Integer>>} \,\} \\ \overset{(red1)}{\Longrightarrow} & \{\, \texttt{x} \lessdot_? {}_?\texttt{List<Integer>} \,\} \\ \overset{3.step}{\Longrightarrow} & \{\, \{\, \texttt{x} \mapsto \texttt{List<Integer>} \,\}, \{\, \texttt{x} \mapsto {}_?\texttt{List<Integer>} \,\}, \\ & \;\{\, \texttt{x} \mapsto \texttt{myLi<b, Integer>} \,\}, \{\, \texttt{x} \mapsto {}_?\texttt{myLi<b, Integer>} \,\} \,\}. \end{array}$$

Example: Infinite chains

Subtyping relation:

$$\texttt{myLi<b, a>} \leq \texttt{List<a>}$$

There is an infinte chain:

$$\ldots \leq^* \texttt{myLi<}{}_?\texttt{myLi<}{}_?\texttt{List<a>, a>, a>} \leq^* \texttt{myLi<}{}_?\texttt{List<a>, a>} \leq^* \texttt{List<a>}$$

Application of the algorithm:

$$\{\, \texttt{List<x>} \lessdot \texttt{List<}{}_?\texttt{List<Integer>>} \,\}$$

$$\overset{(red1)}{\Rightarrow} \{\, \texttt{x} \lessdot_? {}_?\texttt{List<Integer>} \,\}$$

$$\overset{3.step}{\Rightarrow} \{\, \{\, \texttt{x} \mapsto \texttt{List<Integer>} \,\}, \{\, \texttt{x} \mapsto {}_?\texttt{List<Integer>} \,\}, \{\, \texttt{x} \mapsto \texttt{myLi<b, Integer>} \,\}, \{\, \texttt{x} \mapsto {}_?\texttt{myLi<b, Integer>} \,\} \,\}.$$

Although, there are infinte subtypes of `x` only a finite number of general unifiers is determined.

Results

Theorem (Soundness und Completeness)

The type unification algorithm is sound and complete.

Corollary (Finitary)

The type unification of Java 5.0 type terms with wildcards is finitary.

Corollary (Termination)

The type unification algorithm terminates.

The finitary corollary means that the open problem of [Smolka 1989] is solved by our type unification algorithm.

Conclusion and future work

Conclusion

- Java 5.0 type unification problem corresponds to the type unification problem of logical programming languages
- Type unification algorithm for subtype relationships
- Subtype relationships of different arities are allowed

Conclusion and future work

Conclusion

- Java 5.0 type unification problem corresponds to the type unification problem of logical programming languages
- Type unification algorithm for subtype relationships
- Subtype relationships of different arities are allowed

Future work

- Integration of the exended type unification algorithm into the Java 5.0 type inference algorithm
- Completion of the implementation