

# Java type inference with wildcards

Martin Plümicke

University of Cooperative Education  
Stuttgart/Horb

7. November 2006

# Overview

## Introduction

Problem

Related work

## The Algorithm

Type unification

Type-inference-algorithm

Tool demonstration

Principle type

## Conclusion

# Problem

## Extensions of the Java 5.0 type-system

- ▶ parametrized types, type variables, type terms, wildcards

e.g.

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

# Problem

## Extensions of the Java 5.0 type-system

- ▶ parametrized types, type variables, type terms, wildcards

e.g.

```
Vector<? extends ArrayList<? 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)

# Problem

## Extensions of the Java 5.0 type-system

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

```
Vector<? extends ArrayList<? 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)

⇒ 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; }}
```

# Alternative Typing

```
class Matrix extends Vector<Vector<Integer>> {
    Matrix/Vector<Vector<Integer>> mul(Matrix/Vector<Vector<Integer>> m) {
        Matrix/Vector<Vector<Integer>> 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

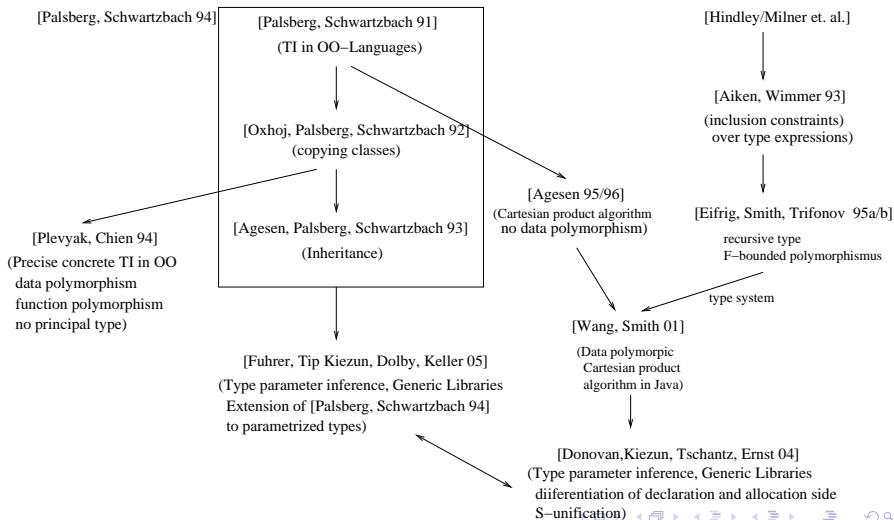
```
class Matrix extends Vector<Vector<Integer>> {
    mul(m) {
        ret = new Matrix();
        i = 0;
        while(i < size()) {
            v1 = this.elementAt(i);
            v2 = new Vector<Integer>();
            j = 0;
            while(j < v1.size()) {
                erg = 0;
          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



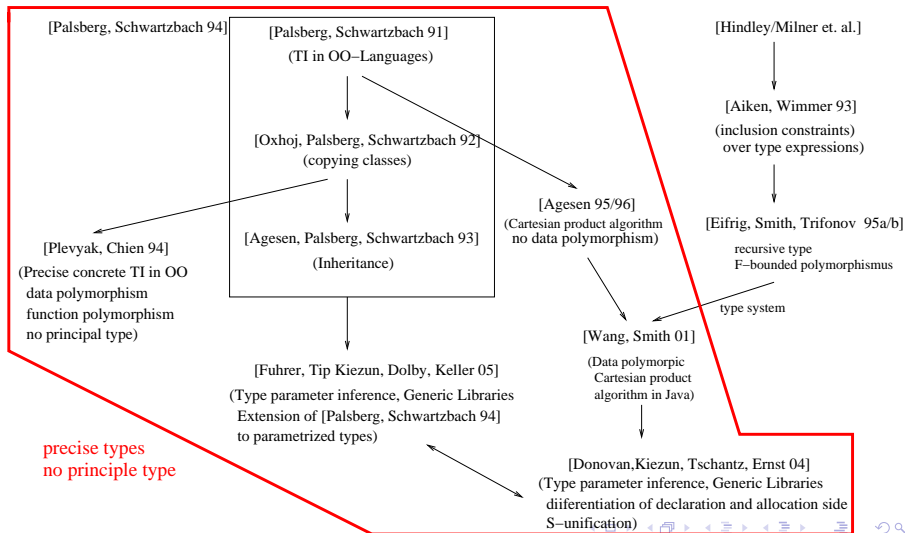
# System determines the principle typing(s)

```
mul: Matrix → Matrix &  
     Matrix → Vector<Vector<Integer>>  
     &...&  
     Vector<? extends Vector<? extends Integer>>  
           → Vector<? super Vector<Integer>>
```

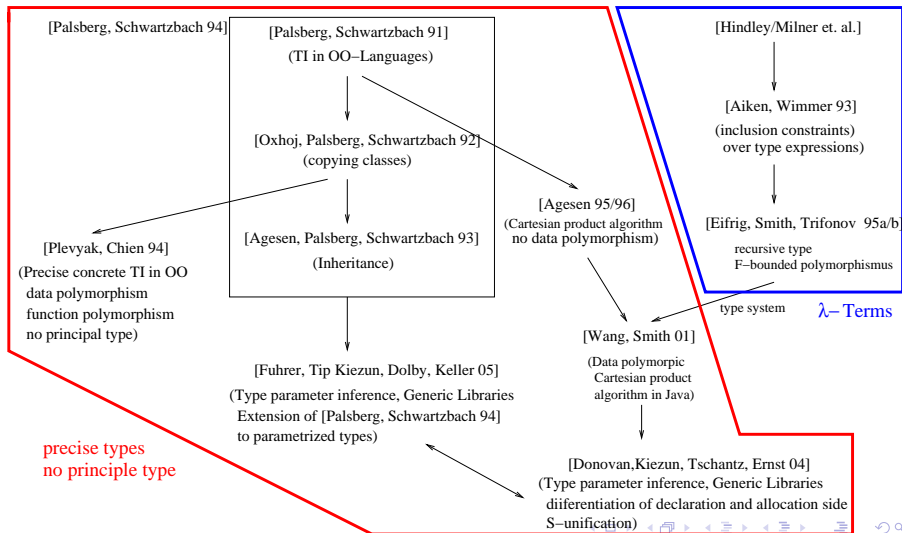
# Related work



# Related work



# Related work



# Our approach

**[Hindley/Milner et al]**

- function type constructor  $\rightarrow$  (no higher-order functions)

# Our approach

## [Hindley/Milner et al]

- function type constructor  $\rightarrow$  (no higher-order functions)
- + function template  $(ty_1 \times \dots \times ty_n) \rightarrow ty_0$   
(first-order functions)

# Our approach

## [Hindley/Milner et al]

- function type constructor  $\rightarrow$  (no higher-order functions)
- + function template  $(ty_1 \times \dots \times ty_n) \rightarrow ty_0$   
(first-order functions)
- + data and function polymorphism (overloading)

# Abbreviation for wildcard-types

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

`A<?B>`

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

`C<?D>`.



# The algorithm

## Type unification

Subtyping relation for type terms:  $\leq^*$

Type Unification problem:

For two type terms  $\theta_1$  and  $\theta_2$  a substitution  $\sigma$  is demanded such that:

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

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

$$\text{(reduce)} \quad \frac{Eq \cup \{ C \langle \theta_1, \dots, \theta_n \rangle \doteq C \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_1 \doteq \theta'_1, \dots, \theta_n \doteq \theta'_n \}}$$

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

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

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

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

(adapt) 
$$\frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \theta'_1, \dots, \theta'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$
 where there are  $\bar{\theta}'_1, \dots, \bar{\theta}'_m$  with

- ▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

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

(adapt) 
$$\frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \theta'_1, \dots, \theta'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$
 where there are  $\bar{\theta}'_1, \dots, \bar{\theta}'_m$  with
 

- ▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

(reduce1) 
$$\frac{Eq \cup \{ C \langle \theta_1, \dots, \theta_n \rangle \leq D \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_{\pi(1)} \doteq \theta'_1, \dots, \theta_{\pi(n)} \doteq \theta'_n \}}$$
 where
 

- ▶  $C \langle a_1, \dots, a_n \rangle \leq^* D \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle$
- ▶  $\{ a_1, \dots, a_n \} \subseteq TV$
- ▶  $\pi$  is a permutation

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

(adapt) 
$$\frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \theta'_1, \dots, \theta'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$
 where there are  $\bar{\theta}'_1, \dots, \bar{\theta}'_m$  with
 

- ▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

(reduce1) 
$$\frac{Eq \cup \{ C \langle \theta_1, \dots, \theta_n \rangle \leq D \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_{\pi(1)} \doteq \theta'_1, \dots, \theta_{\pi(n)} \doteq \theta'_n \}}$$
 where
 

- ▶  $C \langle a_1, \dots, a_n \rangle \leq^* D \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle$
- ▶  $\{ a_1, \dots, a_n \} \subseteq TV$
- ▶  $\pi$  is a permutation

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

- ▶  $a$  occurs in  $Eq$  but not in  $\theta$

(reduce2) 
$$\frac{Eq \cup \{ C \langle \theta_1, \dots, \theta_n \rangle \doteq C \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_1 \doteq \theta'_1, \dots, \theta_n \doteq \theta'_n \}}$$

# Type unification algorithm with wildcards

$$(\text{redUp}) \frac{Eq \cup \{\theta \leq ?\theta'\}}{Eq \cup \{\theta \leq \theta'\}}$$

$$(\text{redUpLow}) \frac{Eq \cup \{?\theta \leq ?\theta'\}}{Eq \cup \{\theta \leq \theta'\}}$$

$$(\text{redLow}) \frac{Eq \cup \{?\theta \leq \theta'\}}{Eq \cup \{\theta \leq \theta'\}}$$

Wildcards in outermost position

## Type unification algorithm with wildcards

$$(\text{redUp}) \frac{Eq \cup \{\theta \leq ?\theta'\}}{Eq \cup \{\theta \leq \theta'\}} \quad (\text{redUpLow}) \frac{Eq \cup \{?\theta \leq ?\theta'\}}{Eq \cup \{\theta \leq \theta'\}} \quad (\text{redLow}) \frac{Eq \cup \{?\theta \leq \theta'\}}{Eq \cup \{\theta \leq \theta'\}}$$

Wildcards in outermost position

$$(\text{red1}) \frac{Eq \cup \{C \langle \theta_1, \dots, \theta_n \rangle \leq D \langle \theta'_1, \dots, \theta'_n \rangle\}}{Eq \cup \{\theta_{\pi(1)} \leq ?\theta'_1, \dots, \theta_{\pi(n)} \leq ?\theta'_n\}}$$

Reduce rule for outermost  
type symbol

where

- $C \langle a_1, \dots, a_n \rangle \leq^* D \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle$
- $\{a_1, \dots, a_n\} \subseteq BTV$
- $\pi$  is a permutation

## Type unification algorithm with wildcards

$$(\text{redUp}) \frac{Eq \cup \{ \theta \leq ? \theta' \}}{Eq \cup \{ \theta \leq \theta' \}} \quad (\text{redUpLow}) \frac{Eq \cup \{ ? \theta \leq ? \theta' \}}{Eq \cup \{ \theta \leq \theta' \}} \quad (\text{redLow}) \frac{Eq \cup \{ ? \theta \leq \theta' \}}{Eq \cup \{ \theta \leq \theta' \}}$$

Wildcards in outermost position

$$(\text{red1}) \frac{Eq \cup \{ C \langle \theta_1, \dots, \theta_n \rangle \leq D \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_{\pi(1)} \leq ? \theta'_1, \dots, \theta_{\pi(n)} \leq ? \theta'_n \}}$$

Reduce rule for outermost  
type symbol

where

- $C \langle a_1, \dots, a_n \rangle \leq^* D \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle$
- $\{ a_1, \dots, a_n \} \subseteq BTV$
- $\pi$  is a permutation

$$(\text{redExt}) \frac{Eq \cup \{ X \langle \theta_1, \dots, \theta_n \rangle \leq ? ? Y \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_{\pi(1)} \leq ? \theta'_1, \dots, \theta_{\pi(n)} \leq ? \theta'_n \}}$$

Reduce rule for  
extends wildcards

where

- $? Y \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle \in \mathbf{gr}(X \langle a_1, \dots, a_n \rangle)$
- $\{ a_1, \dots, a_n \} \subseteq BTV$
- $\pi$  is a permutation



# Type unification algorithm with wildcards

$$(\text{redSup}) \frac{Eq \cup \{ X \langle \theta_1, \dots, \theta_n \rangle \triangleleft ? Y \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta'_1 \triangleleft ? \theta_{\pi(1)}, \dots, \theta'_n \triangleleft ? \theta_{\pi(n)} \}}$$

Reduce rule for  
super wildcards

where

- $? Y \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle \in \mathbf{gr}( X \langle a_1, \dots, a_n \rangle )$
- $\{ a_1, \dots, a_n \} \subseteq BTV$
- $\pi$  is a permutation

# Type unification algorithm with wildcards

$$\text{(redSup)} \quad \frac{Eq \cup \{ X \langle \theta_1, \dots, \theta_n \rangle \triangleleft ? Y \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta'_1 \triangleleft ? \theta_{\pi(1)}, \dots, \theta'_n \triangleleft ? \theta_{\pi(n)} \}}$$

Reduce rule for  
super wildcards

where

- $? Y \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle \in \mathbf{gr}(X \langle a_1, \dots, a_n \rangle)$
- $\{ a_1, \dots, a_n \} \subseteq \mathit{BTV}$
- $\pi$  is a permutation

$$\text{(redEq)} \quad \frac{Eq \cup \{ X \langle \theta_1, \dots, \theta_n \rangle \triangleleft ? X \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_{\pi(1)} \doteq \theta'_1, \dots, \theta_{\pi(n)} \doteq \theta'_n \}}$$

Reduce rule for  
equal type symbols

## Type unification algorithm with wildcards

$$(\text{redSup}) \frac{Eq \cup \{ X \langle \theta_1, \dots, \theta_n \rangle \triangleleft ? Y \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta'_1 \triangleleft ? \theta_{\pi(1)}, \dots, \theta'_n \triangleleft ? \theta_{\pi(n)} \}}$$

Reduce rule for  
super wildcards

where

- $? Y \langle a_{\pi(1)}, \dots, a_{\pi(n)} \rangle \in \mathbf{gr}(X \langle a_1, \dots, a_n \rangle)$
- $\{ a_1, \dots, a_n \} \subseteq \mathit{BTV}$
- $\pi$  is a permutation

$$(\text{redEq}) \frac{Eq \cup \{ X \langle \theta_1, \dots, \theta_n \rangle \triangleleft ? X \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_{\pi(1)} \doteq \theta'_1, \dots, \theta_{\pi(n)} \doteq \theta'_n \}}$$

Reduce rule for  
equal type symbols

$$(\text{reduce2}) \frac{Eq \cup \{ C \langle \theta_1, \dots, \theta_n \rangle \doteq C \langle \theta'_1, \dots, \theta'_n \rangle \}}{Eq \cup \{ \theta_1 \doteq \theta'_1, \dots, \theta_n \doteq \theta'_n \}}$$

Original reduce rule

# Type unification algorithm with wildcards

(adapt)

$$\frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$

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

▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

## Type unification algorithm with wildcards

$$\text{(adapt)} \quad \frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$

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

$$\blacktriangleright (D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$$

$$\text{(adaptExt)} \quad \frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq ?? D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq ?? D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$

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

$$\blacktriangleright (D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$$

## Type unification algorithm with wildcards

(adapt) 
$$\frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$
 where there are  $\bar{\theta}'_1, \dots, \bar{\theta}'_m$  with outermost adapt rule

▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

(adaptExt) 
$$\frac{Eq \cup \{ D \langle \theta_1, \dots, \theta_n \rangle \leq ?? D' \langle \theta'_1, \dots, \theta'_m \rangle \}}{Eq \cup \{ D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq ?? D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$
 where there are  $\bar{\theta}'_1, \dots, \bar{\theta}'_m$  with adapt rule: extends wildcard

▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

(adaptSup) 
$$\frac{Eq \cup \{ D' \langle \theta'_1, \dots, \theta'_m \rangle \leq ? D \langle \theta_1, \dots, \theta_n \rangle \}}{Eq \cup \{ D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle [a_i \mapsto \theta_i \mid 1 \leq i \leq n] \leq ?? D' \langle \theta'_1, \dots, \theta'_m \rangle \}}$$
 where there are  $\bar{\theta}'_1, \dots, \bar{\theta}'_m$  with adapt rule: super wildcard

▶  $(D \langle a_1, \dots, a_n \rangle \leq^* D' \langle \bar{\theta}'_1, \dots, \bar{\theta}'_m \rangle) \in \mathbf{FC}(\leq)$

# Type unification algorithm with wildcards

$$\text{(erase1)} \quad \frac{Eq \cup \{\theta \triangleleft \theta'\}}{Eq} \quad \theta \leq^* \theta'$$

$$\text{(erase2)} \quad \frac{Eq \cup \{\theta \triangleleft? \theta'\}}{Eq} \quad \theta' \in \mathbf{gr}(\theta)$$

$$\text{(erase3)} \quad \frac{Eq \cup \{\theta \doteq \theta'\}}{Eq} \quad \theta = \theta'$$

# Type unification algorithm with wildcards

$$\text{(erase1)} \quad \frac{Eq \cup \{\theta \triangleleft \theta'\}}{Eq} \quad \theta \leq^* \theta'$$

$$\text{(erase2)} \quad \frac{Eq \cup \{\theta \triangleleft ? \theta'\}}{Eq} \quad \theta' \in \mathbf{gr}(\theta)$$

$$\text{(erase3)} \quad \frac{Eq \cup \{\theta \doteq \theta'\}}{Eq} \quad \theta = \theta'$$

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

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



## Example

**Subtyping relation:**  $\text{Matrix}\langle a \rangle \leq^* \text{Vector}\langle \text{Vector}\langle a \rangle \rangle$   
 $\text{Vector}\langle a \rangle \leq^* \text{AbstractList}\langle a \rangle \leq^* \text{List}\langle a \rangle$

**Application of the algorithm:**

$\{ \text{Matrix}\langle b \rangle \triangleleft \text{Vector}\langle \text{Vector}\langle ?\text{AbstractList}\langle \text{Object} \rangle \rangle \rangle, a \triangleleft b \}$

*(adapt)*  
 $\implies \{ \text{Vector}\langle \text{Vector}\langle b \rangle \rangle \triangleleft \text{Vector}\langle \text{Vector}\langle ?\text{AbstractList}\langle \text{Object} \rangle \rangle \rangle, a \triangleleft b \}$

*(red1, redEq)*  
 $\implies \{ b \doteq ?\text{AbstractList}\langle \text{Object} \rangle, a \triangleleft b \}$

*(subst)*  
 $\implies \{ b \doteq ?\text{AbstractList}\langle \text{Object} \rangle, a \triangleleft ?\text{AbstractList}\langle \text{Object} \rangle \}$

*(new sets)*  
 $\implies \{ \{ b \doteq \text{List}\langle \text{Object} \rangle, a \doteq \text{AbstractList}\langle \text{Object} \rangle \}, \{ b \doteq \text{List}\langle \text{Object} \rangle, a \doteq ?\text{AbstractList}\langle \text{Object} \rangle \}, \{ b \doteq \text{List}\langle \text{Object} \rangle, a \doteq \text{Vector}\langle \text{Object} \rangle \}, \{ b \doteq \text{List}\langle \text{Object} \rangle, a \doteq ?\text{Vector}\langle \text{Object} \rangle \} \}$

# Type-inference-algorithm

**Type assumptions:** For each **absent type** in the program a **type-placeholder** (fresh type variable) is assumed.

**Run over the abstract syntax tree:** During the run over abstract syntax tree of the corresponding java class the types are **calculated** gradually by **type unification**.

**Multiplying the assumptions:** If the result of a **type unification** contains **more than one result** or if there is **data polymorphism**, the set of type assumptions is **multiplied**.

**Erase type assumptions:** If the **type unification fails**, the corresponding set of type assumptions is **erased**.

**New method type parameters:** At the end remained **type-placeholders** are replaced by new introduced **method type parameters**.

**Intersection types:** At the end **each** remained set of type assumptions forms **one element** of the result's **intersection type**.

## Type-inference rules

$$[\mathbf{Assign}] \frac{(O, \tau, \tau') \triangleright_{Expr} e_1 : \theta', (O, \tau, \tau') \triangleright_{Expr} e_2 : \theta}{(O, \tau, \tau') \triangleright_{Expr} \mathbf{Assign}(e_1, e_2) : \theta} \quad \theta \leq^* \theta'$$

## Type-inference rules

$$\text{[Assign]} \quad \frac{(O, \tau, \tau') \triangleright_{Expr} e_1 : \theta', \quad (O, \tau, \tau') \triangleright_{Expr} e_2 : \theta}{(O, \tau, \tau') \triangleright_{Expr} \text{Assign}(e_1, e_2) : \theta'} \quad \theta \leq^* \theta'$$

$$\text{[Method-Call]} \quad \frac{\begin{array}{l} (O, \tau, \tau') \triangleright_{Expr} re : \bar{\theta} \\ \forall 1 \leq i \leq n : (O, \tau, \tau') \triangleright_{Expr} e_i : \theta_i, \\ (\theta'_1 \dots \theta'_n, \theta) = \text{lub}(\bar{\theta}, f, (\theta_1, \dots, \theta_n)) \end{array}}{(O, \tau, \tau') \triangleright_{Expr} \text{MethodCall}(re, f(e_1, \dots, e_n)) : \theta}$$

## Type-inference rules

$$\text{[Assign]} \quad \frac{(O, \tau, \tau') \triangleright_{Expr} e_1 : \theta', \quad (O, \tau, \tau') \triangleright_{Expr} e_2 : \theta}{(O, \tau, \tau') \triangleright_{Expr} \text{Assign}(e_1, e_2) : \theta} \quad \theta \leq^* \theta'$$

$$\text{[Method-Call]} \quad \frac{\begin{array}{l} (O, \tau, \tau') \triangleright_{Expr} re : \bar{\theta} \\ \forall 1 \leq i \leq n : (O, \tau, \tau') \triangleright_{Expr} e_i : \theta_i, \\ (\theta'_1 \dots \theta'_n, \theta) = \text{lub}(\bar{\theta}, f, (\theta_1, \dots, \theta_n)) \end{array}}{(O, \tau, \tau') \triangleright_{Expr} \text{MethodCall}(re, f(e_1, \dots, e_n)) : \theta}$$

$$\text{[Return]} \quad \frac{(O, \tau, \tau') \triangleright_{Expr} e : \theta}{(O, \tau, \tau') \triangleright_{Stmt} \text{Return}(e) : \theta}$$

# Example: Multiplication of matrices: Type assumptions

```
class Matrix extends Vector<Vector<Integer>> {
  { $\alpha$ } mul(({ $\beta$ } m) {
    { $\gamma$ } ret = new Matrix();
    int i = 0;
    while(i < size()) {
      { $\iota$ } v1 = this.elementAt(i);
      { $\kappa$ } v2 = new Vector<Integer>();
      int j = 0;
      while(j < v1.size()) {
        { $\chi$ } erg = 0;
        int k = 0;
        while(k < v1.size()) {
          erg = erg + ({ $\xi$ }({ $\iota$ } v1).elementAt(k))
            * ({ $\psi$ }({ $\phi$ } ({ $\beta$ } m).elementAt(k)).elementAt(j)); k++; }
          v2.addElement({ $\chi$ } erg); j++; }
        ret.addElement({ $\mu$ } v2); i++; }
      return ret; }}}
```

```
ret = new Matrix ()
```

```
{  $\alpha$  } mul({  $\beta$  } m) {  
    {  $\gamma$  } ret = { Matrix } new Matrix();  
    ...  
    return {  $\gamma$  } ret;  
}
```

**Unification:** **Matrix**  $\leq \gamma$

$\Rightarrow$

$\gamma = \mathbf{Matrix}$

$\gamma = \mathbf{Vector}\langle\mathbf{Vector}\langle\mathbf{Integer}\rangle\rangle$

$\gamma = \mathbf{Vector}\langle?\mathbf{Vector}\langle\mathbf{Integer}\rangle\rangle$

$\gamma = \mathbf{Vector}\langle?\mathbf{Vector}\langle?\mathbf{Integer}\rangle\rangle$

$\gamma = \mathbf{Vector}\langle?\mathbf{Vector}\langle?\mathbf{Integer}\rangle\rangle$

$\gamma = \mathbf{Vector}\langle?\mathbf{Vector}\langle\mathbf{Integer}\rangle\rangle$

# Type assumptions after the first unification

```
class Matrix extends Vector<Vector<Integer>> {
    { $\alpha$ ,  $\alpha$ ,  $\alpha$ ,  $\alpha$ ,  $\alpha$ ,  $\alpha$ } mul({ $\beta$ ,  $\beta$ ,  $\beta$ ,  $\beta$ ,  $\beta$ ,  $\beta$ } m) {
        {Matrix, Vector<Vector<Integer>>, Vector<?Vector<Integer>>,
        Vector<?Vector<?Integer>>, Vector<?Vector<?Integer>>,
        Vector<?Vector<Integer>>} ret = new Matrix();
    int i = 0; while(i < size()) {
        { $\iota$ ,  $\iota$ ,  $\iota$ ,  $\iota$ ,  $\iota$ ,  $\iota$ } v1 = this.elementAt(i);
        { $\kappa$ ,  $\kappa$ ,  $\kappa$ ,  $\kappa$ ,  $\kappa$ ,  $\kappa$ } v2 = new Vector<Integer>();
        int j = 0; while(j < v1.size()) {
            { $\chi$ ,  $\chi$ ,  $\chi$ ,  $\chi$ ,  $\chi$ ,  $\chi$ } erg = 0;
            int k = 0; while(k < v1.size()) {
                erg = erg + ({ $\xi$ ,  $\xi$ ,  $\xi$ ,  $\xi$ ,  $\xi$ ,  $\xi$ } ({ $\iota$ ,  $\iota$ ,  $\iota$ ,  $\iota$ ,  $\iota$ ,  $\iota$ } v1).elementAt(k))
                * ({ $\psi$ ,  $\psi$ ,  $\psi$ ,  $\psi$ ,  $\psi$ ,  $\psi$ }
                ({ $\phi$ ,  $\phi$ ,  $\phi$ ,  $\phi$ ,  $\phi$ ,  $\phi$ }
                ({ $\beta$ ,  $\beta$ ,  $\beta$ ,  $\beta$ ,  $\beta$ ,  $\beta$ } m).elementAt(k)).elementAt(j)); k++; }
            v2.addElement({ $\chi$ ,  $\chi$ ,  $\chi$ ,  $\chi$ ,  $\chi$ ,  $\chi$ } erg); j++; }
        ret.addElement({ $\mu$ ,  $\mu$ ,  $\mu$ ,  $\mu$ ,  $\mu$ ,  $\mu$ } v2); i++; }
    return ret; } }
```



```
v1 = this.elementAt(i);
```

```
{ $\alpha$ } mul(({ $\beta$ } m) {  
    ...  
    { $\iota$ } v1 = ({Matrix} this).elementAt(i);  
    ...  
})
```

**Unification:** `Matrix`  $\triangleleft$  `Vector< $\iota$ >`

$\Rightarrow$

```
 $\iota$  = Vector<Integer>  
 $\iota$  = Vector<?Integer>  
 $\iota$  = Vector<?Integer>
```

```
v2 = new Vector<Integer> ();
```

```
{  $\alpha$  } mul ({  $\beta$  } m) {  
  ...  
  {  $\kappa$  } v2 = { Vector<Integer> } new Vector<Integer>();  
  ...  
}
```

**Unification:** `Vector<Integer>`  $\triangleleft$   $\kappa$

$\Rightarrow$

```
 $\kappa$  = Vector<Integer>  
 $\kappa$  = Vector<?Integer>  
 $\kappa$  = Vector<?Integer>
```

```
erg = erg + v1.elementAt(k)
    * m.elementAt(k)).elementAt(j)); (l)
```

```
{ $\alpha$ } mul({ $\beta$ } m) {
  ...
  { $x$ } erg = { $x$ } erg + ({ $\xi$ }({ $\iota$ }v1).elementAt(k)
    * ({ $\psi$ }({ $\phi$ } ({ $\beta$ } m).elementAt(k)).elementAt(j)); k++;}
  ...
}
```

({ $\beta$ } m).elementAt(k): **Unification:**  $\beta \triangleleft \text{Vector}\langle\phi\rangle$

$\beta = \text{Vector}\langle\phi\rangle$

$\beta = \text{Matrix} \Rightarrow \phi = \text{Vector}\langle\text{Integer}\rangle$  or

$\phi = ?\text{Vector}\langle\text{Integer}\rangle$  or

$\phi = ?\text{Vector}\langle?\text{Integer}\rangle$  or

$\phi = ?\text{Vector}\langle\text{Integer}\rangle$

```
erg = erg + v1.elementAt(k)
    * m.elementAt(k)).elementAt(j)); (II)
```

$(\{\phi\}(\{\beta\}_m).elementAt(k)).elementAt(j):$

**Unification:**  $\phi \triangleleft \text{Vector}\langle\psi\rangle$

$\beta = \text{Vector}\langle\phi\rangle \Rightarrow \phi = \text{Vector}\langle\psi\rangle$  or  
 $\phi = ?\text{Vector}\langle\psi\rangle$  or  
 $\phi = \text{Matrix}$

$\beta = \text{Matrix} \Rightarrow \phi = \text{Vector}\langle\text{Integer}\rangle \Rightarrow \text{OK}$   
 $\phi = \dots$

```
erg = erg + v1.elementAt(k)
    * m.elementAt(k)).elementAt(j)); (III)
```

$$\frac{(\{\xi\}(\{\iota\}v1).elementAt(k)) \quad *(\{\psi\}(\{\phi\}(\{\beta\}m).elementAt(k)).elementAt(j))}{\beta = \text{Vector}\langle\phi\rangle :}$$

$\beta = \text{Vector}\langle\phi\rangle :$

From  $* : (\text{int}, \text{int}) \rightarrow \text{int}$  follows

**Unification:**  $\psi \triangleleft \text{Integer}$

$\phi = \text{Vector}\langle\psi\rangle \Rightarrow \psi = \text{Integer} \Rightarrow \text{OK}$   
 $\Rightarrow \psi = ?\text{Integer} \Rightarrow \text{OK}$

$\phi = ?\text{Vector}\langle\psi\rangle \Rightarrow \psi = \text{Integer} \Rightarrow \text{OK}$   
 $\Rightarrow \psi = ?\text{Integer} \Rightarrow \text{OK}$

$\phi = \text{Matrix} \Rightarrow (\psi =)\text{Vector}\langle\text{Integer}\rangle \triangleleft \text{Integer} \Rightarrow \text{fail}$   
 $\Rightarrow$  assumption is erased

```
erg = erg + v1.elementAt(k)
    * m.elementAt(k)).elementAt(j)); (IV)
```

```
{  $\alpha$  } mul({  $\beta$  } m) {  
    ...  
}
```

Result for  $\beta$ :

```
 $\beta$  = Vector<Vector<Integer>>  
 $\beta$  = Vector<Vector<?Integer>>  
 $\beta$  = Vector<?Vector<Integer>>  
 $\beta$  = Vector<?Vector<?Integer>>  
 $\beta$  = Matrix
```

```
erg = erg + v1.elementAt(k)
    * m.elementAt(k)).elementAt(j)); (V)
```

$$\frac{\{\chi\} \text{erg} = \{\chi\} \text{erg} + (\{\xi\}(\{\iota\}v1).\text{elementAt}(k))$$
$$* (\{\psi\}(\{\phi\}(\{\beta\}m).\text{elementAt}(k)).\text{elementAt}(j))}{:}$$

From  $+, * : (\text{int}, \text{int}) \rightarrow \text{int}$   
with

$$\xi = \text{Integer} \triangleleft \text{Integer}$$
$$\xi = ?\text{Integer} \triangleleft \text{Integer}$$

and

$$\psi = \text{Integer} \triangleleft \text{Integer}$$
$$\psi = ?\text{Integer} \triangleleft \text{Integer}$$

follows

$$\chi = \text{Integer}$$

```
v2.addElement(erg);
```

```
{  $\alpha$  } mul({  $\beta$  } m) {  
  ...  
  {  $\kappa$  } v2 = new Vector<Integer>();  
  ...  
  v2.addElement({  $\chi$  } erg);  
  ...  
}
```

$\chi = \text{Integer}, \kappa = \text{Vector}<\text{Integer}> \Rightarrow \text{Integer} \triangleleft \text{Integer} \Rightarrow \text{OK}$

$\chi = \text{Integer}, \kappa = \text{Vector}<? \text{Integer}> \Rightarrow \text{Integer} \triangleleft ? \text{Integer} \Rightarrow \text{fail}$   
 $\Rightarrow$  **assumption** is erased

$\chi = \text{Integer}, \kappa = \text{Vector}<? \text{Integer}> \Rightarrow \text{Integer} \triangleleft ? \text{Integer} \Rightarrow \text{OK}$



```
ret.addElement(v2); (I)
```

```
{ $\gamma$ } ret = new Matrix(); ... ; ret.addElement({ $\kappa$ } v2);
```

$\kappa = \text{Vector}\langle\text{Integer}\rangle$ ,  $\gamma = \text{Matrix}$

$\Rightarrow \text{Vector}\langle\text{Integer}\rangle \triangleleft \text{Vector}\langle\text{Integer}\rangle \Rightarrow \text{OK}$

$\kappa = \text{Vector}\langle\text{Integer}\rangle$ ,  $\gamma = \text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle$

$\Rightarrow \text{Vector}\langle\text{Integer}\rangle \triangleleft \text{Vector}\langle\text{Integer}\rangle \Rightarrow \text{OK}$

$\kappa = \text{Vector}\langle\text{Integer}\rangle$ ,  $\gamma = \text{Vector}\langle?\text{Vector}\langle\text{Integer}\rangle\rangle$

$\Rightarrow \text{Vector}\langle\text{Integer}\rangle \triangleleft ?\text{Vector}\langle\text{Integer}\rangle \Rightarrow \text{fail}$

$\kappa = \text{Vector}\langle\text{Integer}\rangle$ ,  $\gamma = \text{Vector}\langle?\text{Vector}\langle?\text{Integer}\rangle\rangle$

$\Rightarrow \text{Vector}\langle\text{Integer}\rangle \triangleleft ?\text{Vector}\langle?\text{Integer}\rangle \Rightarrow \text{fail}$

$\kappa = \text{Vector}\langle\text{Integer}\rangle$ ,  $\gamma = \text{Vector}\langle?\text{Vector}\langle?\text{Integer}\rangle\rangle$

$\Rightarrow \text{Vector}\langle\text{Integer}\rangle \triangleleft ?\text{Vector}\langle?\text{Integer}\rangle \Rightarrow \text{fail}$

$\kappa = \text{Vector}\langle\text{Integer}\rangle$ ,  $\gamma = \text{Vector}\langle?\text{Vector}\langle\text{Integer}\rangle\rangle$

$\Rightarrow \text{Vector}\langle\text{Integer}\rangle \triangleleft ?\text{Vector}\langle\text{Integer}\rangle \Rightarrow \text{OK}$

```
ret.addElement(v2); (II)
```

```
{ $\gamma$ } ret = new Matrix(); ... ; ret.addElement({ $\kappa$ } v2);
```

With  $\kappa = \text{Vector}\langle ?\text{Integer}\rangle$  for all assumptions of  $\gamma$  type unification fails.

## Result:

$\gamma = \text{Matrix}$

$\gamma = \text{Vector}\langle \text{Vector}\langle \text{Integer}\rangle \rangle$

$\gamma = \text{Vector}\langle ?\text{Vector}\langle \text{Integer}\rangle \rangle$

```
return ret;
```

```
{  $\alpha$  } mul({  $\beta$  } m) {  
    ...  
    return {  $\gamma$  } ret;  
}
```

**Unification:**  $\gamma \leq \alpha$  for

$\gamma = \text{Matrix}$

$\gamma = \text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle$

$\gamma = \text{Vector}\langle?\text{Vector}\langle\text{Integer}\rangle\rangle$

**Result:**  $\alpha = \text{Matrix}$

$\alpha = \text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle$

$\alpha = \text{Vector}\langle?\text{Vector}\langle\text{Integer}\rangle\rangle$

$\alpha = \text{Vector}\langle?\text{Vector}\langle\text{Integer}\rangle\rangle$

$\alpha = \text{Vector}\langle?\text{Vector}\langle?\text{Integer}\rangle\rangle$

$\alpha = \text{Vector}\langle?\text{Vector}\langle?\text{Integer}\rangle\rangle$

## Result:

$$\text{mul} : \&_{\alpha, \beta} (\alpha \rightarrow \beta),$$

where

$$\alpha \leq^* \text{Vector} \langle ? \text{Vector} \langle ? \text{Integer} \rangle \rangle,$$
$$\text{Matrix} \leq^* \beta$$

# Principal type

**Definition [Damas, Milner 1982]:**

“A type-scheme for a declaration is a *principal type-scheme*, if any other type-scheme for the declaration is a generic instance of it.”

# Principal type

## Definition [Damas, Milner 1982]:

“A type-scheme for a declaration is a *principal type-scheme*, if any other type-scheme for the declaration is a generic instance of it.”

## Generalization to the Java 5.0 type system

“An **intersection** type-scheme for a declaration is a *principal type-scheme*, if any other type-scheme for the declaration is a generic instance of **one element of the intersection type-scheme**.”

# Principal type

## Definition

An intersection type of a method  $m$  in a class  $C$

$$m : (\theta_{1,1} \times \dots \times \theta_{1,n} \rightarrow \theta_1) \\ \& \dots \& \\ (\theta_{m,1} \times \dots \times \theta_{m,n} \rightarrow \theta_m)$$

is called *principal* if for any correct type annotated method declaration

$$rty\ m(ty_1\ a_1, \dots, ty_n\ a_n) \{ \dots \}$$

there is an element  $(\theta_{i,1} \times \dots \times \theta_{i,n} \rightarrow \theta_i)$  of the intersection type and there is a substitution  $\sigma$ , such that

$$\sigma(\theta_i) \leq^* rty, ty_1 \leq^* \sigma(\theta_{i,1}), \dots, ty_n \leq^* \sigma(\theta_{i,n})$$

## Reduced principal type:

The inferred type of the example

$$\text{mul} : \&_{\alpha, \beta} (\alpha \rightarrow \beta),$$

where  $\alpha \leq^* \text{Vector} \langle ? \text{Vector} \langle ? \text{Integer} \rangle \rangle$  and  $\text{Matrix} \leq^* \beta$ .

is a principle type.



## Reduced principal type:

The inferred type of the example

$$\text{mul} : \&_{\alpha, \beta} (\alpha \rightarrow \beta),$$

where  $\alpha \leq^* \text{Vector}\langle ? \text{Vector}\langle ? \text{Integer} \rangle \rangle$  and  $\text{Matrix} \leq^* \beta$ .

is a principle type.

But there is also a reduced principle type:

$$\text{mul} : \text{Vector}\langle ? \text{Vector}\langle ? \text{Integer} \rangle \rangle \rightarrow \text{Matrix}$$

# Principle type property

## Theorem

*The type inference algorithm calculates a principle type.*

# Tool demonstration

- ▶ Matrix-Example
- ▶ Overloading-Example

# Conclusion and future work

## Conclusion

- ▶ Type-inference-algorithm for Java 5.0
- ▶ Type unification
- ▶ Principle type property

# Conclusion and future work

## Conclusion

- ▶ Type-inference-algorithm for Java 5.0
- ▶ Type unification
- ▶ Principle type property

## Future work

- ▶ Reduce the number of calculated typings.
- ▶ Handling of intersection types (adaption of byte-code generation)