Type inference in the generic type system of Java 5.0

Martin Plümicke

University of Cooperative Education
Stuttgart/Horb

12. Januar 2007

Overview

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 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)

Problem

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; }}
```

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; }}
```

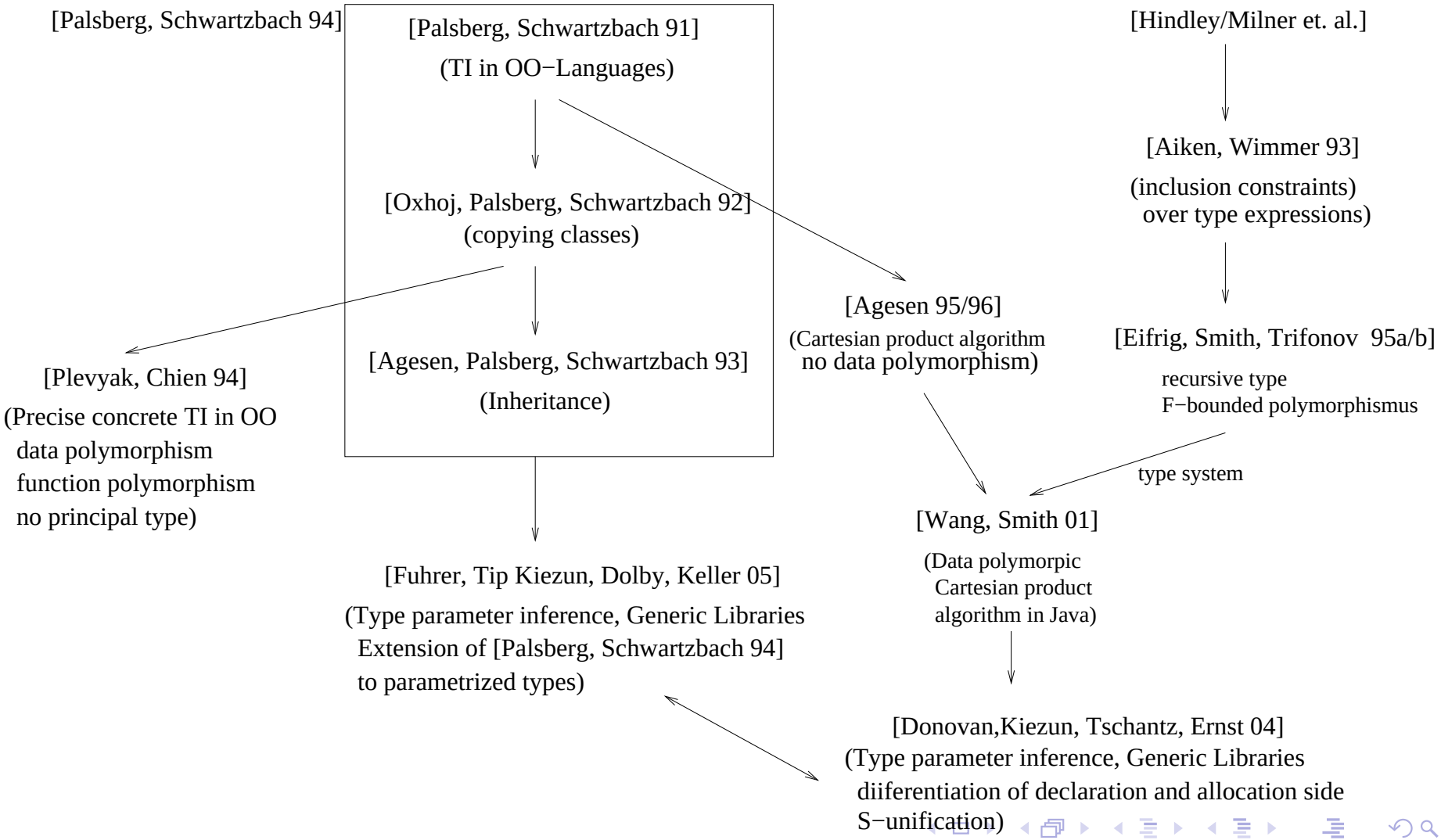
Purpose: Typless

```
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; }}
```

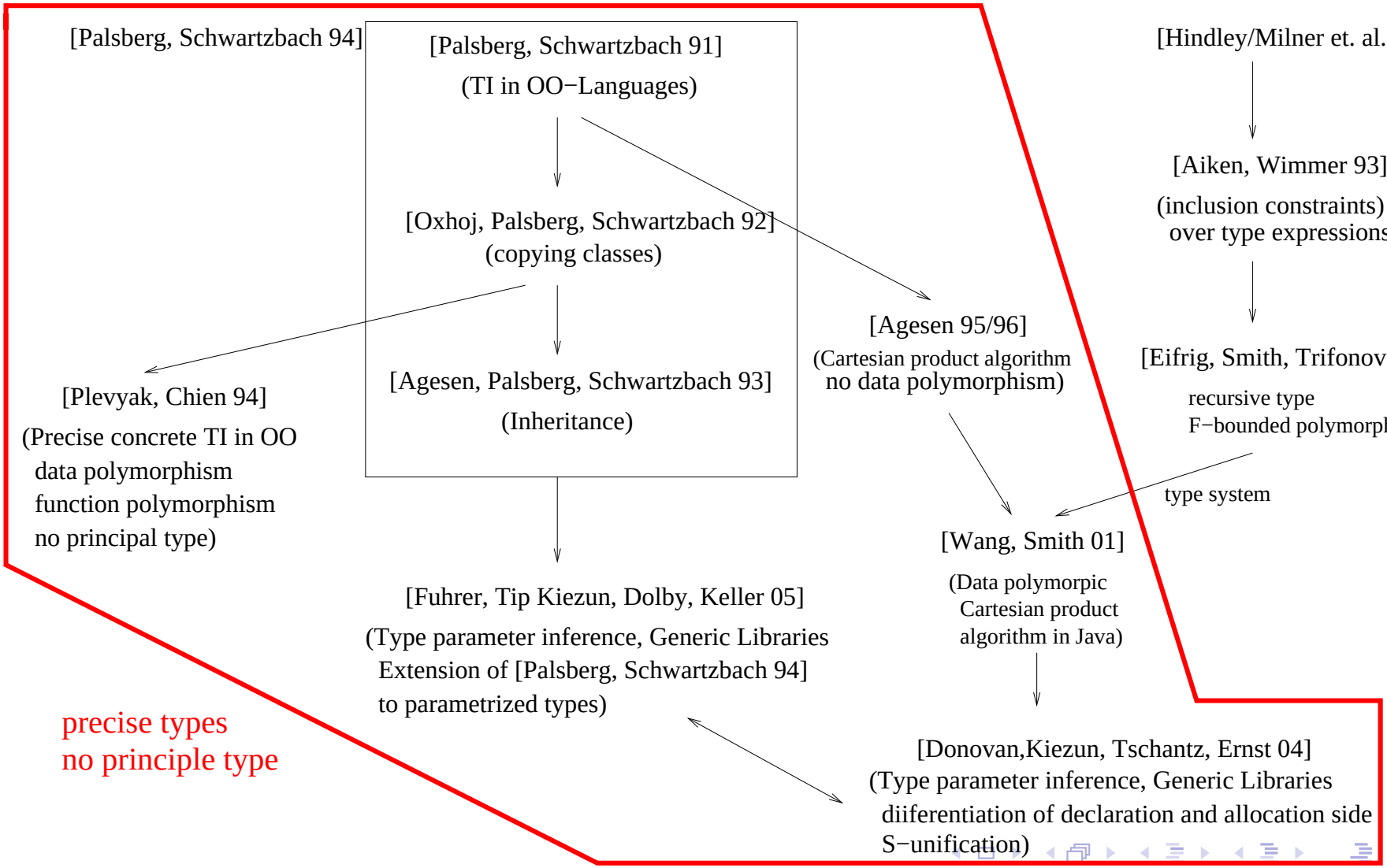
System determines the principle typing(s)

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

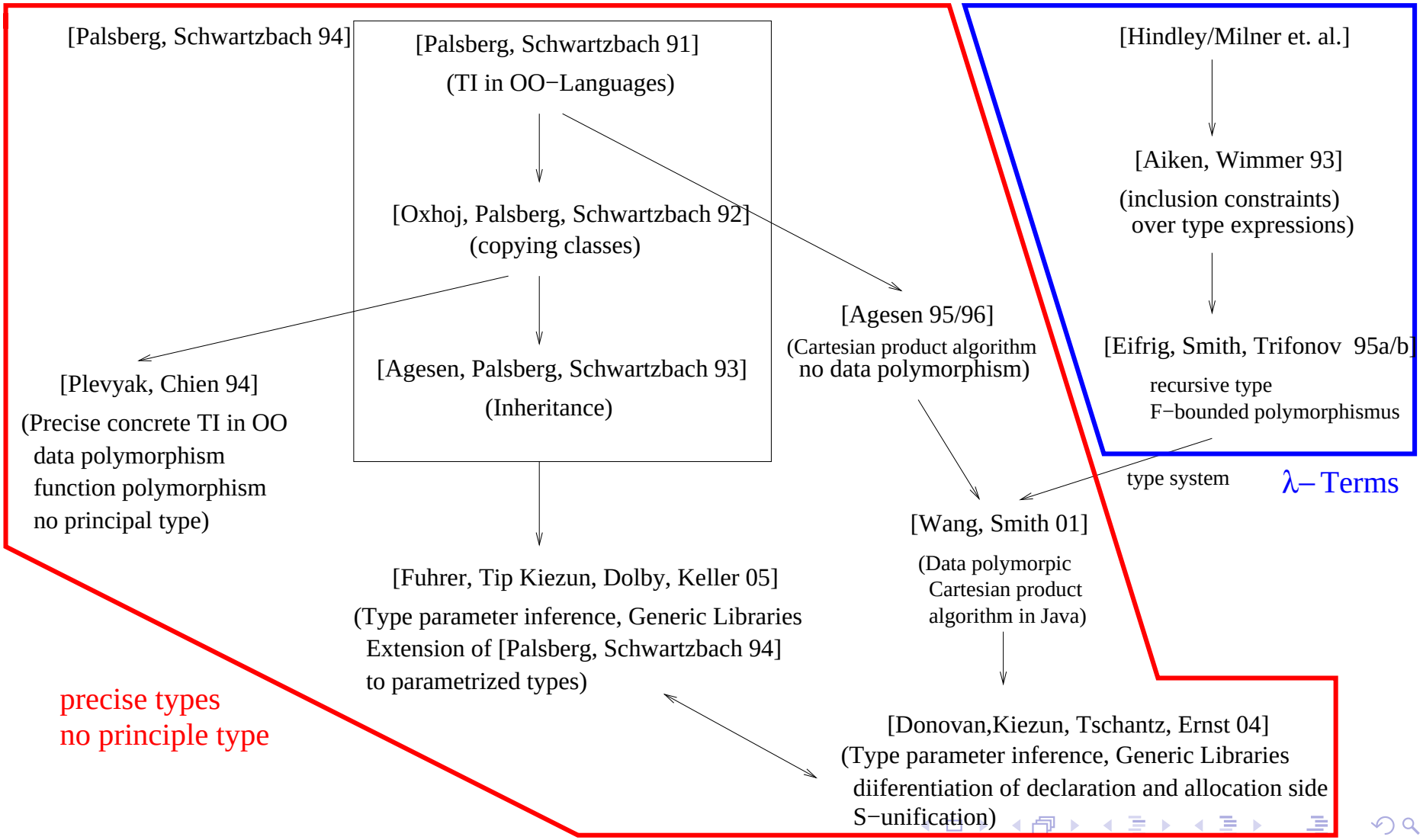
Related work



Related work



Related work



Our approach

[Hindley/Milner et al]

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

Our approach

[Hindley/Milner et al]

- function type constructor $\longrightarrow$ (no higher–order functions)
+ function template $(ty_1 \times \ldots \times ty_n) \longrightarrow ty_0$ (first–order functions)

Our approach

[Hindley/Milner et al]

- function type constructor $\longrightarrow$ (no higher–order functions)
+ function template $(ty_1 \times \ldots \times ty_n) \longrightarrow ty_0$ (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>`.

Subtyping ordering $\leq^*$

Reflexive and transitiv closure of

- if θ `extends` θ' then $\theta \leq^* \theta'$.
- if $\theta_1 \leq^* \theta_2$ then $\sigma_1(\theta_1) \leq^* \sigma_2(\theta_2)$, where for each type variable a of θ_2 holds $\sigma_1(a) = \sigma_2(a)$ (soundness condition).
- $a \leq^* \theta'$ for $a \in BTV^{(\theta_1 \& \ldots \& \theta_n)}$ where $\exists \theta_i : \theta_i \leq^* \theta'$.
- It holds $C\langle\theta_1, \ldots, \theta_n\rangle \leq^* C\langle\theta'_1, \ldots, \theta'_n\rangle$ 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
  - $\theta'_i = {}^?\theta_i$
- It holds $C\langle\theta_1, \ldots, \theta_n\rangle \leq^* C\langle\theta'_1, \ldots, \theta'_n\rangle$ if it holds $C\langle\overline{\theta}_1, \ldots, \overline{\theta}_n\rangle \leq^* C\langle\theta'_1, \ldots, \theta'_n\rangle$ and $C\langle\overline{\theta}_1, \ldots, \overline{\theta}_n\rangle$ is the capture conversions of $C\langle\theta_1, \ldots, \theta_n\rangle$.

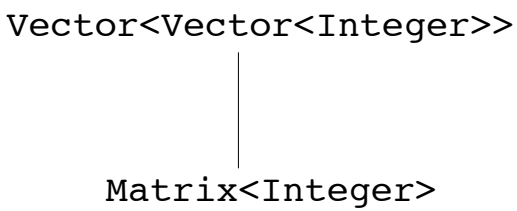
Capture conversion

The *capture conversions* $C\langle\overline{\theta}_1, \ldots, \overline{\theta}_n\rangle$ of a type $C\langle\theta_1, \ldots, \theta_n\rangle$ is defined as:

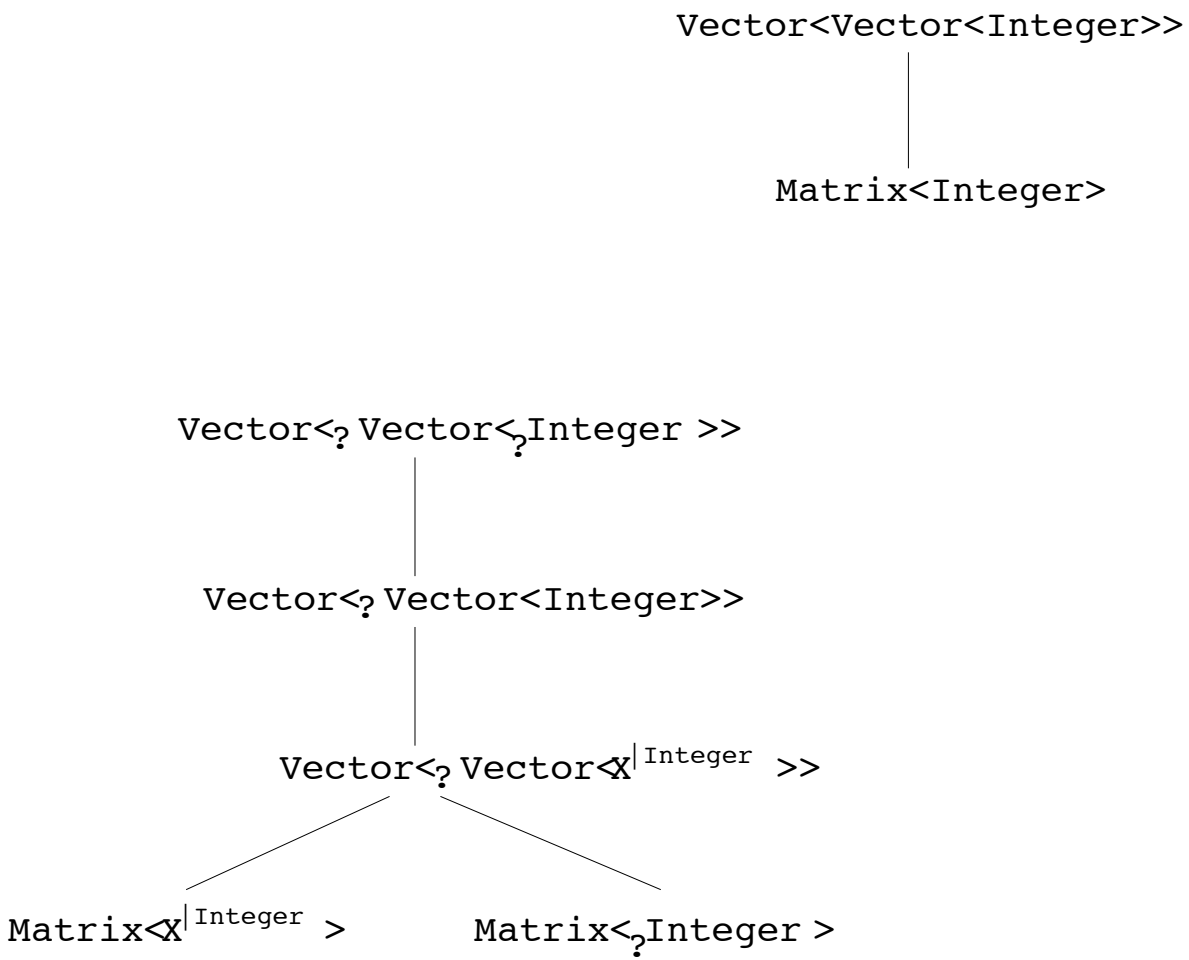
- for all $\theta_i = ?$ there is a fresh type variable $b_i\big|^{u_i[a_i=b_i \mid 1\leqslant i\leqslant n]}$ such that $\overline{\theta}_i = b_i$.
- for all $\theta_i = {}_?\theta'_i$ there is a fresh type variable $b_i\big|^{\theta'_i \,\&\, u_i[a_i=b_i \mid 1\leqslant i\leqslant n]}$, such that $\overline{\theta}_i = b_i$.
- for all $\theta_i = {}^?\theta'_i$ there is a fresh type variable ${}_{\theta'}\big|b_i\big|^{u_i[a_i=b_i \mid 1\leqslant i\leqslant n]}$, such that $\overline{\theta}_i = b_i$.
- otherwise $\overline{\theta}_i = \theta_i$

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

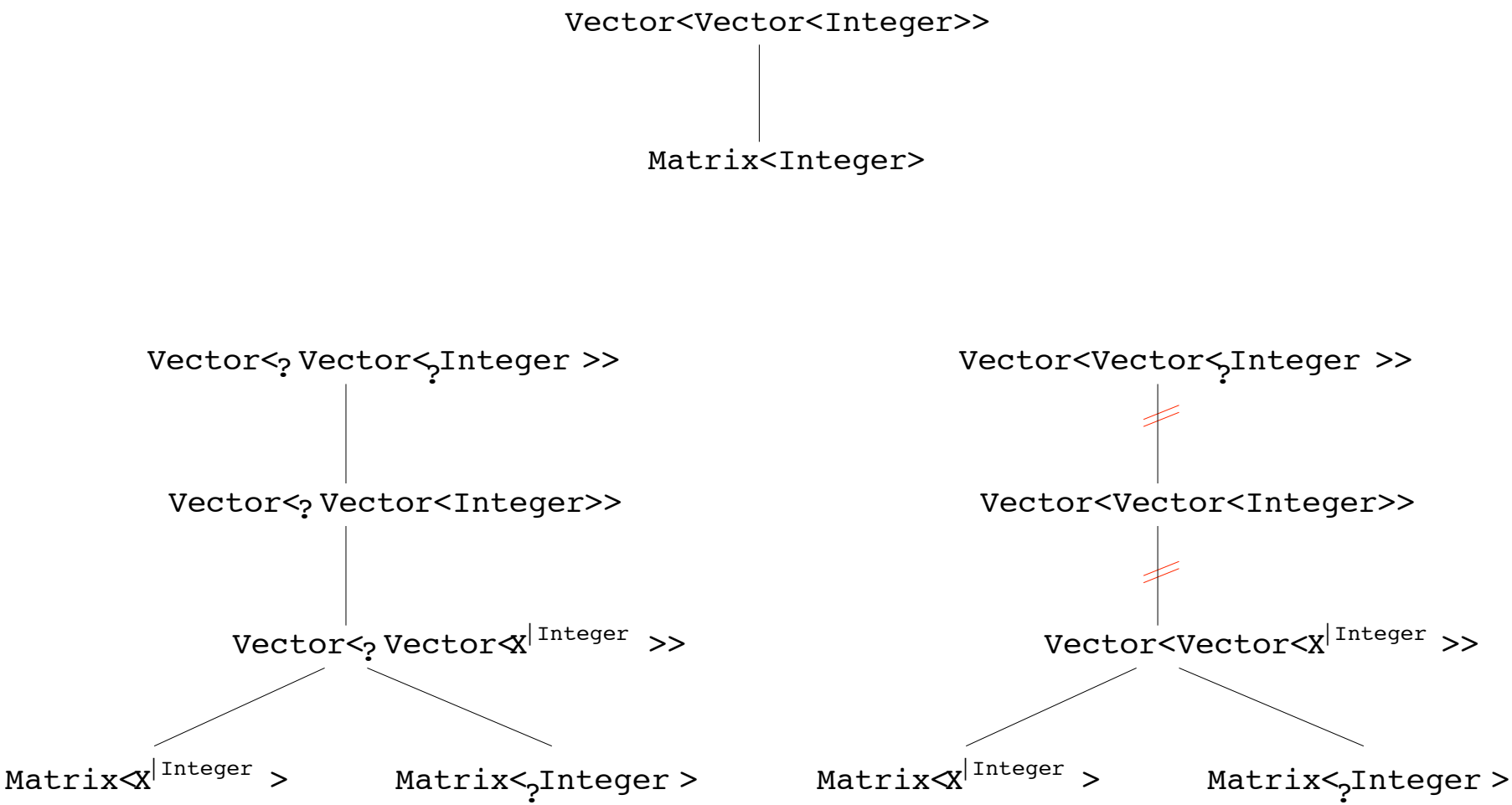
Example: `Matrix<a>` $\leq^*$ `Vector<Vector<a>>`



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



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



The algorithm

Type unification

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

Type Unification problem:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where

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

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

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

where

- a occurs in Eq but not in θ

Type unification algorithm with wildcards

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

Wildcards in outermost position

Type unification algorithm with wildcards

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

Wildcards in outermost position

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

Reduce rule for outermost type symbol

where

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

Type unification algorithm with wildcards

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

Wildcards in outermost position

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

Reduce rule for outermost type symbol

where

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

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

Reduce rule for extends wildcards

where

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

Type unification algorithm with wildcards

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

where

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

Reduce rule for super wildcards

Type unification algorithm with wildcards

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

Reduce rule for super wildcards

where

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

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

Reduce rule for equal type symbols

Type unification algorithm with wildcards

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

Reduce rule for super wildcards

where

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

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

Reduce rule for equal type symbols

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

Original reduce rule

Type unification algorithm with wildcards

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

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

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

Type unification algorithm with wildcards

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

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

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

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

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

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

Type unification algorithm with wildcards

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

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

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

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

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

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

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

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

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

Type unification algorithm with wildcards

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

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

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

Type unification algorithm with wildcards

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

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

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

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

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

Example

Subtyping relation: $\texttt{Matrix<a>} \leq^* \texttt{Vector<Vector<a>>}$

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

Application of the algorithm:

$$\{\, \texttt{Matrix<b>} \lessdot \texttt{Vector<Vector<}_?\texttt{AbstractList<Object>>>}, \texttt{a} \doteq \texttt{b} \,\}$$

$$\overset{(\textit{adapt})}{\Longrightarrow} \quad \{\, \texttt{Vector<Vector<}\mathrm{CC}(\,\texttt{b}\,)\texttt{>>} \lessdot \texttt{Vector<Vector<}_?\texttt{AbstractList<Object>>>}, \texttt{a} \doteq \texttt{b} \,\}$$

$$\overset{(\textit{red1},\textit{redEq})}{\Longrightarrow} \quad \{\, \mathrm{CC}(\,\texttt{b}\,) \doteq {}_?\texttt{AbstractList<Object>}, \texttt{a} \doteq \texttt{b} \,\}$$

$$\Longrightarrow \quad \textbf{fail}$$

Example cont.

$$\{\texttt{Matrix<b>} \lessdot \texttt{Vector<?Vector<?AbstractList<Object>>>},\ \texttt{a} \doteq \texttt{b}\}$$

$$\xRightarrow{adapt} \{\texttt{Vector<Vector<}\mathrm{CC}(\texttt{b})\texttt{>>} \lessdot \texttt{Vector<?Vector<?AbstractList<Object>>>},\ \texttt{a} \doteq \texttt{b}\}$$

$$\xRightarrow{red1, redExt} \{\mathrm{CC}(\texttt{b}) \lessdot_? \texttt{?AbstractList<Object>},\ \texttt{a} \doteq \texttt{b}\}$$

$$\xRightarrow{newsets, subst} \{\{\texttt{b} \doteq \texttt{AbstractList<Object>}, \texttt{a} \doteq \texttt{AbstractList<Object>}\},\ \{\texttt{b} \doteq \texttt{?AbstractList<Object>}, \texttt{a} \doteq \texttt{?AbstractList<Object>}\},\ \{\texttt{b} \doteq \texttt{Vector<Object>}, \texttt{a} \doteq \texttt{Vector<Object>}\},\ \{\texttt{b} \doteq \texttt{?Vector<Object>}, \texttt{a} \doteq \texttt{?Vector<Object>}\}\}$$

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 coresponding 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

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

Type–inference rules

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

$$[\textbf{Method-Call}] \quad \frac{\begin{array}{c}(O,\tau,\tau') \rhd_{Expr} re : \overline{\theta} \\ \forall 1 \leqslant i \leqslant n : \; (O,\tau,\tau') \rhd_{Expr} e_i : \theta_i, \\ (\theta'_1 \ldots \theta'_n, \theta) = \texttt{lub}(\overline{\theta}, f, (\theta_1, \ldots, \theta_n))\end{array}}{(O,\tau,\tau') \rhd_{Expr} \text{MethodCall}(re, f(e_1, \ldots, e_n)) : \theta}$$

Type–inference rules

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

$$[\textbf{Method-Call}] \quad \frac{\begin{array}{c}(O,\tau,\tau') \rhd_{Expr} re : \overline{\theta} \\ \forall 1 \leqslant i \leqslant n : \; (O,\tau,\tau') \rhd_{Expr} e_i : \theta_i, \\ (\theta'_1 \ldots \theta'_n, \theta) = \texttt{lub}(\overline{\theta}, f, (\theta_1, \ldots, \theta_n))\end{array}}{(O,\tau,\tau') \rhd_{Expr} \text{MethodCall}(re, f(e_1, \ldots, e_n)) : \theta}$$

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

Example: Multiplication of matrices: Type assumptions

```
class Matrix extends Vector<Vector<Integer>> {
   {α} mul({β} m) {
      {γ} ret = new Matrix();
      int i = 0;
      while(i <size()) {
         {ι} v1 = this.elementAt(i);
         {κ} v2 = new Vector<Integer>();
         int j = 0;
         while(j < v1.size()) {
            {χ} erg = 0;
            int k = 0;
            while(k < v1.size()) {
               erg = erg + ({ξ}({ι} v1).elementAt(k))
                  * ({ψ}({φ} ({β} m).elementAt(k)).elementAt(j)); k++;}
            v2.addElement({χ} erg); j++; }
         ret.addElement({μ} v2); i++; }
      return ret; }}
```

ret = new Matrix ()

```
{α} mul({β} m) {
    {γ} ret = {Matrix} new Matrix();
    ...
    return {γ} ret;
}
```

Unification: $\texttt{Matrix} \lessdot \gamma$

$\Rightarrow$

$\gamma = \texttt{Matrix}$

$\gamma = \texttt{Vector<Vector<Integer>>}$

$\gamma = \texttt{Vector<?Vector<Integer>>}$

$\gamma = \texttt{Vector<?Vector<?Integer>>}$

$\gamma = \texttt{Vector<?Vector<?Integer>>}$

$\gamma = \texttt{Vector<?Vector<Integer>>}$

Type assumptions after the first unification

```
class Matrix extends Vector<Vector<Integer>> {
   {α, α, α, α, α, α} mul({β, β, β, β, β, β} 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()) {
         {ι, ι, ι, ι, ι, ι} v1 = this.elementAt(i);
         {κ, κ, κ, κ, κ, κ} v2 = new Vector<Integer>();
         int j = 0; while(j < v1.size()) {
            {χ, χ, χ, χ, χ, χ} erg = 0;
            int k = 0; while(k < v1.size()) {
               erg = erg +({ξ,ξ,ξ,ξ,ξ,ξ}({ι,ι,ι,ι,ι,ι} v1).elementAt(k))
                * ({ψ,ψ,ψ,ψ,ψ,ψ}
                   ({ϕ,ϕ,ϕ,ϕ,ϕ,ϕ}
                    ({β,β,β,β,β,β} m).elementAt(k)).elementAt(j)); k++;}
            v2.addElement({χ, χ, χ, χ, χ, χ} erg); j++; }
         ret.addElement({μ, μ, μ, μ, μ, μ} v2); i++; }
      return ret; }}
```

v1 = this.elementAt(i);

```
{α} mul({β} m) {
    ...
    {ι} v1 = ({Matrix} this).elementAt(i);
    ...
}
```

Unification: Matrix $\lessdot$ Vector<ι>

$\Rightarrow$

$$\iota = \texttt{Vector<Integer>}$$
$$\iota = \texttt{Vector<?Integer>}$$
$$\iota = \texttt{Vector<?Integer>}$$

v2 = new Vector<Integer> ();

```
{α} mul({β} m) {
    ...
    {κ} v2 = {Vector<Integer>} new Vector<Integer>();
    ...
}
```

Unification: $\texttt{Vector<Integer>} \lessdot \kappa$

$\Rightarrow$

$$\kappa = \texttt{Vector<Integer>}$$
$$\kappa = \texttt{Vector<?Integer>}$$
$$\kappa = \texttt{Vector<?Integer>}$$

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

```
{α} mul({β} m) {
  ...
        {χ} erg = {χ} erg + ({ξ}({ι}v1).elementAt(k))
         * ({ψ}({φ} ({β} m).elementAt(k)).elementAt(j)); k++;}
  ...
}
```

({β} m).elementAt(k): **Unification:** $\beta \lessdot$ Vector<ϕ>

β = Vector<ϕ>

β = Matrix $\Rightarrow$ ϕ = Vector<Integer> or

ϕ = ?Vector<Integer> or

ϕ = ?Vector<?Integer> or

ϕ = ?Vector<Integer>

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

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

Unification: $\phi \lessdot$ Vector<ψ>

$\beta =$ Vector<ϕ> $\Rightarrow$ $\phi =$ Vector<ψ> or
$\phi =$?Vector<ψ> or
$\phi =$ Matrix

$\beta =$ Matrix $\Rightarrow$ $\phi =$ Vector<Integer> $\Rightarrow$ **OK**
$\phi = \ldots$

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

$(\{\xi\}(\{\iota\}$v1$)$.elementAt(k)$)$
$*(\{\psi\}(\{\phi\}(\{\beta\}m)$.elementAt(k)$)$.elementAt(j)$)$:

$\beta =$ Vector<ϕ> :

From $*$: (int, int) $\rightarrow$ int follows

Unification: $\psi \lessdot$ Integer

$\phi =$ Vector<ψ> $\Rightarrow \psi =$ Integer $\Rightarrow$ **OK**
$\Rightarrow \psi =$?Integer $\Rightarrow$ **OK**

$\phi =$?Vector<ψ> $\Rightarrow \psi =$ Integer $\Rightarrow$ **OK**
$\Rightarrow \psi =$?Integer $\Rightarrow$ **OK**

$\phi =$ Matrix $\Rightarrow (\psi =)$Vector<Integer> $\lessdot$ Integer $\Rightarrow$ **fail**

$\Rightarrow$ assumption is erased

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

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

Result for β:

$\beta =$ 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)
```

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

From $+, * : (\texttt{int}, \texttt{int}) \longrightarrow \texttt{int}$
with

$$\xi = \texttt{Integer} \lessdot \texttt{Integer}$$
$$\xi = {}_?\texttt{Integer} \lessdot \texttt{Integer}$$

and

$$\psi = \texttt{Integer} \lessdot \texttt{Integer}$$
$$\psi = {}_?\texttt{Integer} \lessdot \texttt{Integer}$$

follows

$$\chi = \texttt{Integer}$$

v2.addElement(erg);

```
{α} mul({β} m) {
  ...
  {κ} v2 = new Vector<Integer>();
  ...
  v2.addElement({χ} erg);
  ...
}
```

$\chi = \texttt{Integer}, \kappa = \texttt{Vector<Integer>} \Rightarrow \texttt{Integer} \lessdot \texttt{Integer} \Rightarrow$ **OK**

$\chi = \texttt{Integer}, \kappa = \texttt{Vector<?Integer>} \Rightarrow \texttt{Integer} \lessdot \texttt{?Integer} \Rightarrow$ **fail**

$\Rightarrow$ assumption is erased

$\chi = \texttt{Integer}, \kappa = \texttt{Vector<?Integer>} \Rightarrow \texttt{Integer} \lessdot \texttt{?Integer} \Rightarrow$ **OK**

ret.addElement(v2); (I)

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

$\kappa =$ Vector<Integer>, $\gamma =$ Matrix
$\Rightarrow$ Vector<Integer> $\lessdot$ Vector<Integer> $\Rightarrow$ **OK**

$\kappa =$ Vector<Integer>, $\gamma =$ Vector<Vector<Integer>>
$\Rightarrow$ Vector<Integer> $\lessdot$ Vector<Integer> $\Rightarrow$ **OK**

$\kappa =$ Vector<Integer>, $\gamma =$ Vector<?Vector<Integer>>
$\Rightarrow$ Vector<Integer> $\lessdot$?Vector<Integer> $\Rightarrow$ **fail**

$\kappa =$ Vector<Integer>, $\gamma =$ Vector<?Vector<?Integer>>
$\Rightarrow$ Vector<Integer> $\lessdot$?Vector<?Integer> $\Rightarrow$ **fail**

$\kappa =$ Vector<Integer>, $\gamma =$ Vector<?Vector<?Integer>>
$\Rightarrow$ Vector<Integer> $\lessdot$?Vector<?Integer> $\Rightarrow$ **fail**

$\kappa =$ Vector<Integer>, $\gamma =$ Vector<?Vector<Integer>>
$\Rightarrow$ Vector<Integer> $\lessdot$?Vector<Integer> $\Rightarrow$ **OK**

ret.addElement(v2); (II)

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

With $\kappa =$ `Vector<?Integer>` for all assumptions of γ type unification fails.

Result:

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

return ret;

```
{α} mul({β} m) {
    ...
    return {γ} ret;
}
```

Unification: $\gamma \lessdot \alpha$ for

$\gamma =$ `Matrix`

$\gamma =$ `Vector<Vector<Integer>>`

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

Result:

$\alpha =$ `Matrix`

$\alpha =$ `Vector<Vector<Integer>>`

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

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

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

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

Result:

$$\texttt{mul}\colon\ \&_{\alpha,\beta}(\alpha \rightarrow \beta),$$

where

$\alpha \leq^* \texttt{Vector<?Vector<?Integer>>}$,
$\texttt{Matrix} \leq^* \beta$

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`

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

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

$$rty \ \texttt{m(}ty1 \ \texttt{a1 , ... ,} \ tyn \ \texttt{an) \{ ... \}}$$

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

$$\sigma(\theta_i) \leq^* rty, ty1 \leq^* \sigma(\theta_{i,1}), \ldots, tyn \leq^* \sigma(\theta_{i,n})$$

Reduced principal type:

The inferred type of the example

$$\texttt{mul}\colon \&_{\alpha,\beta}(\alpha \longrightarrow \beta),$$

where $\alpha \leq^* \texttt{Vector<?Vector<?Integer>>}$ and $\texttt{Matrix} \leq^* \beta$.

is a principle type.

Reduced principal type:

The inferred type of the example

$$\texttt{mul}\colon \&_{\alpha,\beta}(\alpha \rightarrow \beta),$$

where $\alpha \leq^* \texttt{Vector<?Vector<?Integer>>}$ and $\texttt{Matrix} \leq^* \beta$.

is a principle type.

But there is also a reduced principle type:

$$\texttt{mul}\colon \texttt{Vector<?Vector<?Integer>>} \rightarrow \texttt{Matrix}$$

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 interesection types (adaption of byte-code generation)