18th International Workshop on Unification (UNIF'04)
Cork, Ireland, July 5th, 2004

Type Unification in Generic–Java

Martin Plümicke

University of Cooperate Education

Stuttgart, Germany

July 5th, 2004

Overview

1. Motivation: Type–Inference in Java 1.5 (G–JAVA)
2. Inheritance Hierarchy
3. The Type Unification Algorithm
4. Conclusion and Outlook

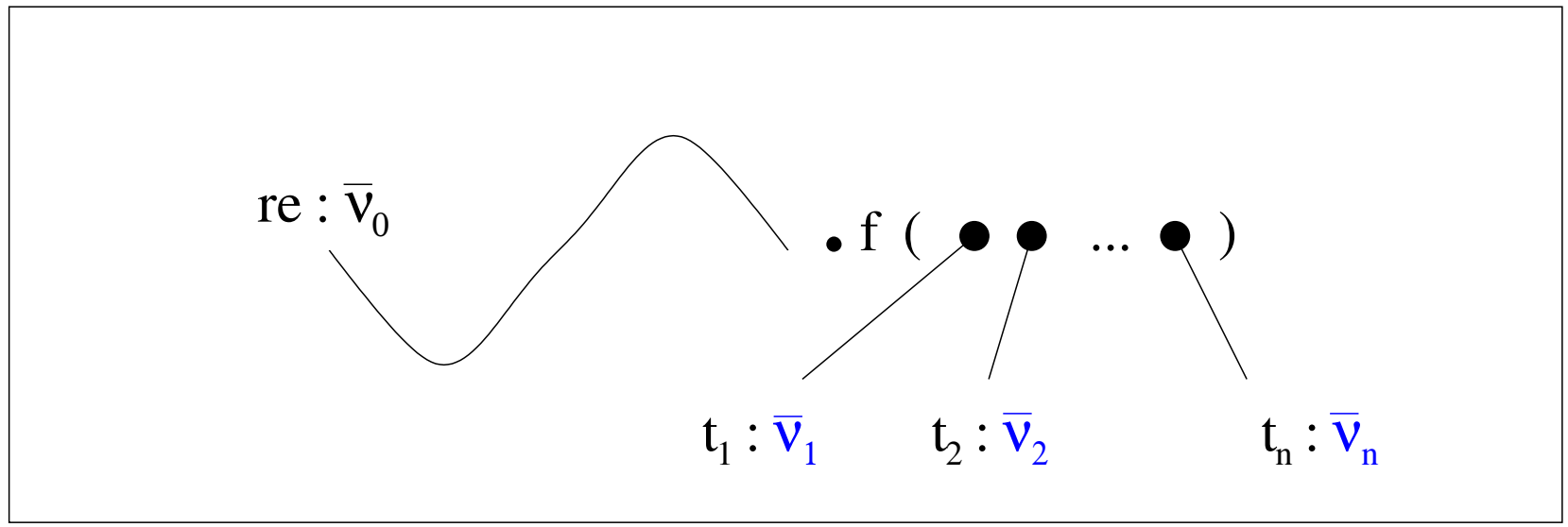
1. Motivation: Type–Inference in G–JAVA

Introduction example

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

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

Type inference for method application



$$f : \nu_1 \times \ldots \times \nu_n \longrightarrow \nu$$

To determine:

$$\textbf{TUnify} \leq^* ((\overline{\nu}_1, \ldots, \overline{\nu}_n), (\nu_1, \ldots, \nu_n))$$

The problem

The type unification problem: For two given type terms θ_1, θ_2 a substitution σ is searched such that

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

The restrictions:

- no interfaces
- no F–bounded parameters
- the number of parameters are equal in all inheritance relations

2. Inheritance Hierarchy

Type terms in G–JAVA

Set of type variables: TV

Rank alphabet: $\Theta = (\Theta^{(n)})_{n \in \mathbb{N}}$

For each class declaration

$$\texttt{class } C\texttt{<}a_1, \ldots, a_n\texttt{>}$$

holds

$$C \in \Theta^{(n)}$$

Examples:

- $\texttt{Seq}, \texttt{Vector} \in \Theta^{(1)}$
- $\texttt{Pair} \in \Theta^{(2)}$

Definition: The *set of types* are the set of terms over Θ: $T_\Theta(TV)$.

Example:

- `Pair<Seq<Integer>, Vector<A>>`

Inheritance hierarchy

Type term ordering $\leq$: The extension declaration

$$\texttt{class } C\texttt{<}a_1, \ldots, a_n\texttt{>} \texttt{ extends } \tau'$$

is described as

$$C\texttt{<}a_1, \ldots, a_n\texttt{>} \leq \tau'.$$

Closure of the type term ordering $\leq^*$: The smallest ordering with the conditions

- if $(\theta, \theta') \in T_\Theta(TV) \times T_\Theta(TV)$ is an element of the reflexive and transitive closure of $\leq$ then $\theta \leq^* \theta'$.
- if $\theta_1 \leq^* \theta_2$ then $\sigma_1(\theta_1) \leq^* \sigma_2(\theta_2)$ for all substitutions σ_1, σ_2, with $\sigma_1(a) = \sigma_2(a)$ for all $a \in \mathsf{TVar}(\theta_2)$.

Contravariance problem

```
class Super { ...}

class Sub extends Super { ...}

class Vector<a> {

    void add (a elem) { ...}
}
```

```
class Main {
  public static void main(String[] args) {
    Vector<Super> v;
    v = new Vector<Sub> ();
    v.add(new Super());
  }
}
```

$$\text{Super} \not\leq^{*} \text{Sub}$$

Finite closure

Example

```
class AbstractList<a> extends List<a> { ... }

class Vector<a> extends AbstractList<a> { ... }

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

class ExtMatrix<a> extends Matrix<a> { ... }
```

Unification of

1. `Matrix<Object>` $\lessdot$ `Vector<Vector<b>>`
   - reduction with `Matrix<a>` $\leq$ `Vector<Vector<a>>`
   - result: `b` $\mapsto$ `Object`.
2. `Matrix<Object>` $\lessdot$ `List<Vector<b>>`
   - there is no `Matrix<a>` $\leq \theta$ for reduction
   - the pair `Matrix<a>` $\leq$ `List<Vector<a>>` would be needed.

$\Longrightarrow$ $\leq$ is extended to the *Finite closure*.

Definition: Finite closure

The *finite closure* $\mathbf{FC}(\leq)$ of $\leq$ is defined as the transitive closure of

$$\{\,(\overline{\theta}, \theta') \in \leq^* \mid \theta \leq \theta', \overline{\theta} \in T_\Theta(TV)\,\} \cup$$
$$\{\,(\theta, \theta'') \in \leq^* \mid \theta \leq \theta', \theta'' \in T_\Theta(TV)\,\}$$

Example

Interesting part of $\mathbf{FC}(\leq)$ (`Matrix<a>` $\leq$ `Vector<Vector<a>>`):

$$\begin{aligned}
fc_1 = \{\,&\texttt{Vector<Vector<a>>} \leq^* \texttt{Vector<Vector<a>>},\\
&\texttt{Matrix<a>} \leq^* \texttt{Vector<Vector<a>>},\\
&\texttt{ExtMatrix<a>} \leq^* \texttt{Vector<Vector<a>>}\,\}\\
fc_2 = \{\,&\texttt{Matrix<a>} \leq^* \texttt{Matrix<a>},\\
&\texttt{Matrix<a>} \leq^* \texttt{Vector<Vector<a>>},\\
&\texttt{Matrix<a>} \leq^* \texttt{AbstractList<Vector<a>>},\\
&\texttt{Matrix<a>} \leq^* \texttt{List<Vector<a>>}\,\}
\end{aligned}$$

$\mathbf{FC}(\leq)$ = transitive and reflexive closure of $(fc_1 \cup fc_2 \cup \leq)$.

3. The Type Unification Algorithm

Type unification problem

For two given type terms θ_1, θ_2 a substitution σ is searched such that

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

Unification Rules *reduce*, *erase*, *swap*

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

adapt rule

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

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

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

subst rule

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

The type unification algorithm

Input: Set of equations $Eq = \{\, \theta_1 \lessdot \theta'_1, \ldots, \theta_n \lessdot \theta'_n \,\}$

Output: Set of most general type unifiers $Uni = \{\, \sigma_1, \ldots, \sigma_m \,\}$

The algorithm itself is given in seven steps:

1. $Eq_1 = \{\, \theta \lessdot \theta' \mid (\theta \lessdot \theta') \in Eq \wedge ((\theta, \theta' \notin TV) \vee (\theta, \theta' \in TV)) \,\}$
2. $Eq_2 = Eq \backslash (Eq_1 \cup \{\, \theta \lessdot \theta \mid (\theta \lessdot \theta) \in Eq \,\})$
3. $Eq_{set} = \{\, Eq_1 \,\} \times (\bigotimes_{(a \lessdot \theta') \in Eq_2} \{\, a \doteq \theta \mid (\theta \leq^* \sigma(\theta')) \in \mathbf{FC}(\leq) \,\}) \times (\bigotimes_{(\theta \lessdot a) \in Eq_2} \{\, a \doteq \theta' \mid \theta \leq^* \theta' \,\})$
4. Repeated application of the rules *reduce1*, *reduce2*, *erase*, *swap*, *adapt* to all elements of Eq_{set}. The end configuration Eq'_{set} is reached if for each element no rule is applicable.
5. Application of the *subst* rule to each element of $Eq' \in Eq'_{set}$ as often as possible. The result is Eq''_{set}.
6. (a) Foreach $Eq'' \in Eq''_{set}$ which has changed in the last step start again with the first step.
   (b) Build the union Eq'''_{set} of all results of (a) and $Eq'' \in Eq''_{set}$ which has not changed in the last step.
7. $Uni = \{\, Eq''' \mid Eq''' \in Eq'''_{set}$ is a unifier (is in solved form) $\,\}$

Example I (1)

$$\Theta^{(0)} = \{\, \texttt{Integer}, \texttt{Object} \,\}, \Theta^{(1)} = \{\, \texttt{Vector}, \texttt{Stack}, \texttt{AbstractList}, \texttt{List} \,\},$$
$$\Theta^{(2)} = \{\, \texttt{Pair} \,\},$$
$$\texttt{Integer} \leq \texttt{Object},$$

and

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

Now, we apply the unification algorithm to

$$Eq = \{\, \texttt{a} \lessdot \texttt{Vector<Integer>},$$
$$\texttt{AbstractList<Object>} \lessdot \texttt{b},$$
$$\texttt{AbstractList<Pair<Stack<Integer>, d>>} \lessdot \texttt{List<Pair<c,List<Integer>>>},$$
$$\texttt{c} \lessdot \texttt{d} \,\}$$

Example I (2)

The result of the first step is

$Eq_1 = \{$ `AbstractList<Pair<Stack<Integer>, d>>` $\lessdot$ `List<Pair<c,List<Integer>>>`,
`c` $\lessdot$ `d` $\}$

The result of the second step is

$Eq_2 = \{$ `a` $\lessdot$ `Vector<Integer>`,
`AbstractList<Object>` $\lessdot$ `b` $\}$

Example I (3)

The third step multiplies the set of pairs:

$$
\begin{aligned}
Eq_{set} = \\
&\{\{\texttt{AbstractList<Pair<Stack<Integer>, d>>} \lessdot \texttt{List<Pair<c,List<Integer>>>}, \\
&\quad \texttt{c} \lessdot \texttt{d}\} \\
&\quad \cup \{\texttt{b} \doteq \texttt{AbstractList<Object>},\ \texttt{a} \doteq \texttt{Vector<Integer>}\}, \\
&\{\texttt{AbstractList<Pair<Stack<Integer>, d>>} \lessdot \texttt{List<Pair<c,List<Integer>>>}, \\
&\quad \texttt{c} \lessdot \texttt{d}\} \\
&\quad \cup \{\texttt{b} \doteq \texttt{List<Object>},\ \texttt{a} \doteq \texttt{Vector<Integer>}\}, \\
&\{\texttt{AbstractList<Pair<Stack<Integer>, d>>} \lessdot \texttt{List<Pair<c,List<Integer>>>}, \\
&\quad \texttt{c} \lessdot \texttt{d}\} \\
&\quad \cup \{\texttt{b} \doteq \texttt{AbstractList<Object>},\ \texttt{a} \doteq \texttt{Stack<Integer>}\}, \\
&\{\texttt{AbstractList<Pair<Stack<Integer>, d>>} \lessdot \texttt{List<Pair<c,List<Integer>>>}, \\
&\quad \texttt{c} \lessdot \texttt{d}\} \\
&\quad \cup \{\texttt{b} \doteq \texttt{List<Object>},\ \texttt{a} \doteq \texttt{Stack<Integer>}\}\}
\end{aligned}
$$

Example I (4)

The result of the fourth step (application of the *reduce* rules and the *swap* rule) is:

$$
\begin{aligned}
Eq'_{set} = \{ \{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \texttt{c} \lessdot \texttt{d}, \\
& \texttt{b} \doteq \texttt{AbstractList<Object>}, \texttt{a} \doteq \texttt{Vector<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \texttt{c} \lessdot \texttt{d}, \\
& \texttt{b} \doteq \texttt{List<Object>}, \texttt{a} \doteq \texttt{Vector<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \texttt{c} \lessdot \texttt{d}, \\
& \texttt{b} \doteq \texttt{AbstractList<Object>}, \texttt{a} \doteq \texttt{Stack<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \texttt{c} \lessdot \texttt{d}, \\
& \texttt{b} \doteq \texttt{List<Object>}, \texttt{a} \doteq \texttt{Stack<Integer>} \} \}
\end{aligned}
$$

Example I (5)

In the fifth step the *subst* rule is applied for `c` and `d`:

$$
\begin{aligned}
Eq''_{set} = \{ \{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{Stack<Integer>} \lessdot \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{AbstractList<Object>}, \ \texttt{a} \doteq \texttt{Vector<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{Stack<Integer>} \lessdot \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{List<Object>}, \ \texttt{a} \doteq \texttt{Vector<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{Stack<Integer>} \lessdot \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{AbstractList<Object>}, \ \texttt{a} \doteq \texttt{Stack<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{Stack<Integer>} \lessdot \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{List<Object>}, \ \texttt{a} \doteq \texttt{Stack<Integer>} \} \}
\end{aligned}
$$

Example I (6)

The sixth step is divided in **(a)** and **(b)**:

(a): In the first three steps nothing is changed, as Eq_2 is empty.

In step four `Stack<Integer>` $\lessdot$ `List<Integer>` is reduced and finally erased.

(b): This leads to the result

$$
\begin{aligned}
Eq'''_{set} = \{ \{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{AbstractList<Object>},\ \texttt{a} \doteq \texttt{Vector<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{List<Object>},\ \texttt{a} \doteq \texttt{Vector<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{AbstractList<Object>},\ \texttt{a} \doteq \texttt{Stack<Integer>} \}, \\
\{ & \texttt{c} \doteq \texttt{Stack<Integer>}, \texttt{d} \doteq \texttt{List<Integer>}, \\
& \texttt{b} \doteq \texttt{List<Object>},\ \texttt{a} \doteq \texttt{Stack<Integer>} \} \}
\end{aligned}
$$

Example I (7)

In the seventh step no mapping is erased, as all pairs are in solved form, such that the result are four most general unifiers:

$$\{\{\texttt{a} \mapsto \texttt{Vector<Integer>}, \texttt{b} \mapsto \texttt{AbstractList<Object>},$$
$$\texttt{c} \mapsto \texttt{Stack<Integer>}, \texttt{d} \mapsto \texttt{List<Integer>}\},$$
$$\{\texttt{a} \mapsto \texttt{Vector<Integer>}, \texttt{b} \mapsto \texttt{List<Object>},$$
$$\texttt{c} \mapsto \texttt{Stack<Integer>}, \texttt{d} \mapsto \texttt{List<Integer>}\},$$
$$\{\texttt{a} \mapsto \texttt{Stack<Integer>}, \texttt{b} \mapsto \texttt{AbstractList<Object>},$$
$$\texttt{c} \mapsto \texttt{Stack<Integer>}, \texttt{d} \mapsto \texttt{List<Integer>}\},$$
$$\{\texttt{a} \mapsto \texttt{Stack<Integer>}, \texttt{b} \mapsto \texttt{List<Object>},$$
$$\texttt{c} \mapsto \texttt{Stack<Integer>}, \texttt{d} \mapsto \texttt{List<Integer>}\}\}$$

Example II (1)

Example with the *adapt* rule:

$$\Theta^{(0)} = \{\, \texttt{Object} \,\}, \Theta^{(1)} = \{\, \texttt{Vector}, \texttt{Matrix}, \texttt{List} \,\}$$

and

$$\texttt{Matrix<a>} \leq \texttt{Vector<Vector<a>>}.$$

Now, we apply the unification algorithm to

$$Eq = \{\, \texttt{Matrix<List<b>>} \lessdot \texttt{Vector<Vector<List<Object>>>} \,\}$$

Example II (2)

- In the first three steps nothing happens.
- In step four neither the *erase*, nore the *swap* rule is applicable.
- The *reduce1* rule is also not applicable, as there is no relation

$$\texttt{Matrix<a>} \leq \texttt{Vector<a>}.$$

$\Rightarrow$ Now the *adapt* rule is necessary.

Example II (3)

As

$$(\texttt{Matrix<a>} \leq^{*} \texttt{Vector<Vector<a>>}) \in \mathbf{FC}(\leq)$$

from

$$Eq = \{\, \texttt{Matrix<List<b>>} \lessdot \texttt{Vector<Vector<List<Object>>>} \,\}$$

by the *adapt* rule follows:

$$Eq_{set} = \{\, \texttt{Vector<Vector<List<b>>>} \doteq \texttt{Vector<Vector<List<Object>>>} \,\}$$

Then the *reduce2* rule leads finally to the result:

$$\{\, \{\, \texttt{b} \mapsto \texttt{Object} \,\} \,\}$$

Soundness and completeness

Theorem: The type unification problem is solved by the type unification algorithm. This means that the algorithm is sound and complete.

4. Conclusion and Outlook

Summary:

- We described the inheritance in G–JAVA by two relations.
- We solved the type unification problem in G–JAVA, with the restrictions:
  - no interfaces
  - no F–bounded parameters
  - the number of parameters are equal in all inheritance relations

To do:

- Extension to interfaces and F–bounded parameters.
- The project is continued by implementing the type inference algorithm.
- Finally the idea is to integrate the type inference algorithm in a programming framework like eclipse or netbeans. The user can then elect one of the determined types for the methods.