

Untyped methods in Generic Java

Martin Plümicke

University of Cooperative Education Stuttgart
Department of Information Technology
Florianstraße 15
D-72160 Horb
tel. +49-7451-521142
fax. +49-7451-521190
m.pluemicke@ba-horb.de

Abstract. In this paper we present the type system of Java version 1.5 and a type inference algorithm for untyped Java programs. By a small example we show that the code is more reusable if a type inference algorithm is used.

1 Generic types in Java

In version 1.5 of sun's java (G-JAVA), generic types are implemented [1]. Generic types are parameterized types. The parameters can be instantiated by other types. For example the type of the vectors has a parameter `Vector<a>`. Then a declaration "`Vector<Integer> var;`" means that the vector `var` contains only `Integers`. The main advantage is, that in each other method, which needs an element of the vector, the type of the element is `Integer`, definitely. This means that on the one hand many type casts are no longer needed. For example "`var.get(5).intValue()`" without type cast is correct. On the other hand some run time errors become compiling errors. For example: `((String)var.get(5)) ++ "hallo"` causes already a compiling error.

Generic types in a java framework are known since PIZZA [2] and Generic Java [3, 4].

2 Type system of G-JAVA

The types in G-JAVA are given as terms over class names and type variables. For example `Vector<a>` and `Vector<Vector<Integer>>` are type terms. The inheritance ordering \leq can be extended to an partial ordering \leq^* on the set of type terms. It holds $C\langle\theta_1, \dots, \theta_n\rangle \leq^* C'\langle\theta'_1, \dots, \theta'_n\rangle$ if the class `C` extends the class `C'` ($C\langle a_1, \dots, a_n\rangle \leq C'\langle a_1, \dots, a_n\rangle$) and $\theta_i = \theta'_i$.

It is possible to assign an expression of a lower type (wrt. \leq^*) as demanded to a variable. For example the declaration "`Matrix extends Vector<Vector<Integer>>`" (`Matrix` \leq^* `Vector<Vector<Integer>>`) allows: If `m` has the type `Matrix` and `var` the type `Vector<Vector<Integer>>` then "`var = m;`" is valid.

3 Type Inference

Type inference is well-known from functional programming languages since [5]. The goal of the type inference algorithm is to determine the result type, and the parameter types for an untyped G-JAVA method. These types can be derived from typed fields and already typed methods. Let us consider the following example.

```
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).intValue()
            * (m.elementAt(k).elementAt(j).intValue());
          k++; }
        v2.addElement(new Integer(erg));
        j++; }
      ret.addElement(v2);
      i++; }
    return ret; } }
```

The method `mul` calculates the matrix multiplication. The types of the variables (printed *italic*) and the result type are determined by the type inference algorithm.

The algorithm: The idea of the algorithm is, that for each method variable a type variable as type is assumed. Then during a visit of the whole abstract syntax tree of the G-JAVA program the assumed type variables are instantiated. The instantiation is determined by unification: If a method with the argument type θ' is applied to an argument with the type θ , then the unifier σ is determined such that $\sigma(\theta) \leq^* \sigma(\theta')$ holds. This unification is not unitary, which means that there are sometimes more than one most general unifier. This means that an untyped method of a G-JAVA program can have more than one type.

The result of the algorithm for the `mul` method of the above `Matrix` example is:

$$\begin{aligned} & (\text{Matrix} \rightarrow \text{Matrix}) \\ & \wedge (\text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle \rightarrow \text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle) \\ & \wedge (\text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle \rightarrow \text{Matrix}) \\ & \wedge (\text{Matrix} \rightarrow \text{Vector}\langle\text{Vector}\langle\text{Integer}\rangle\rangle). \end{aligned}$$

The result is surprising. If a programmer would type the above program, probably only the first typing (`Matrix` \rightarrow `Matrix`) would be identified. The other

types are not obvious. The reasons for the other types are that neither on the parameter `m` the method `mul` is called, nor `mul` is called recursively.

4 Conclusion and future work

We outlined an algorithm to reconstruct generic types in `Java`, which are introduced in version 1.5. The next step is to implement this algorithm by extending a `G-JAVA` compiler. Finally, the goal is to integrate this algorithm in a programming framework. Then the algorithm is applied during program construction and the user can choose the best type for his application.

The example shows that the algorithm can determine the types of `G-JAVA` methods more precisely than the user himself. This means that the type inference algorithm would lead to more reusable `G-JAVA` programs. This is beside the convenience for the programmers the most benefit of using type inference.

References

1. Austin, C.: J2SE 1.5 in a nutshell. java.sun.com/developer/technicalArticles/releases/j2se15 (2004)
2. Odersky, M., Wadler, P.: Pizza into Java: Translating theory into practice. In: Proceedings of the 24th ACM Symposium on Principles of Programming Languages. (1997)
3. Bracha, G., Odersky, M., Stoutamire, D., Wadler, P.: GJ Specification. (1998)
4. Bracha, G., Cohen, N., Kemper, C., Marx, S., Odersky, M., Panitz, S.E., Thorup, D.S.K., Wadler, P.: Adding Generics to the Java Programming Language: Participant Draft Specification. <http://java.sun.com> (2001)
5. Damas, L., Milner, R.: Principal type-schemes for functional programs. Proc. 9th Symposium on Principles of Programming Languages (1982)