Typ–Inferenz in Java 5.0

Martin Plümicke
University of Cooperative Education Stuttgart
Department of Information Technology
Florianstraße 15, D–72160 Horb
m.pluemicke@ba-horb.de

1 Einleitung

Das Typsystem von Java 5.0 [GJSB05] ist gegenüber den Vorgängerversionen um parametrisierte Typen, Typvariablen, Typterme und Wildcards erweitert worden. Dies hat zur Folge, dass sehr komplexe Typausdrücke entstehen können. Beispielsweise ist

`Vector<Vector<AbstractList<Integer>>>`

in Java 5.0 ein korrekter Typ. Programmierern fällt es oftmals schwer zu erkennen, dass Typen dieser Art für bestimmte Java 5.0 Methoden eine korrekte Typisierung wären. Desweiteren gibt es Java 5.0 Methoden, die Durchschnittstypen als allgemeinste Typisierung hätten. Durchschnittstypen sind allerdings in Java 5.0 nicht implementiert. Das heißt oftmals haben Java 5.0 Methoden nicht den allgemeinst möglichen Typ, was dem Ziel wiederverwendbaren Code zu schreiben entgegensteht.
Das hat uns veranlasst ein Typinferenz–System für Java 5.0 zu entwickeln, das den Benutzer durch automatische Typberechnungen unterstützt. Typinferenz in Java 5.0 ermöglicht es, Parameter von Methoden und lokale Variablen ungetypt zu deklarieren. Der Typinferenz–Algorithmus berechnet dann jeweils den allgemeinsten Typ.

2 Typsystem

Die Grundlage des Typinferenz-Systems ist eine Typordnung $\leq^*$, die wir aus der Vererbungshierarchie ableiten. Eine ganz wesentliche Eigenschaft der Typordnung ist, dass bei zwei Typtermen $\theta \leq^* \theta_2$ nur die äußersten Klassen voneinander erben dürfen. Die Argumente der Typterme müssen identisch sein. Ohne diese Eigenschaft wäre bereits der Kern von Java 5.0 nicht typsicher. Diese Eigenschaft stellt darüber hinaus sicher, dass die Typordnung $\leq^*$ keine unendlichen Ketten besitzt und dass das Typ–Unifikations–Problem finitär lösbar ist (siehe nächster Abschnitt).
Das Java 5.0 Typsystem hat große Ähnlichkeit mit polymorphic order–sorted Typsystemen von logischen Programmiersprachen (z.B. [Smo89]) und der funktionalen Programmiersprache OBJ–P [Plü99].

3 Typ–Unifikation

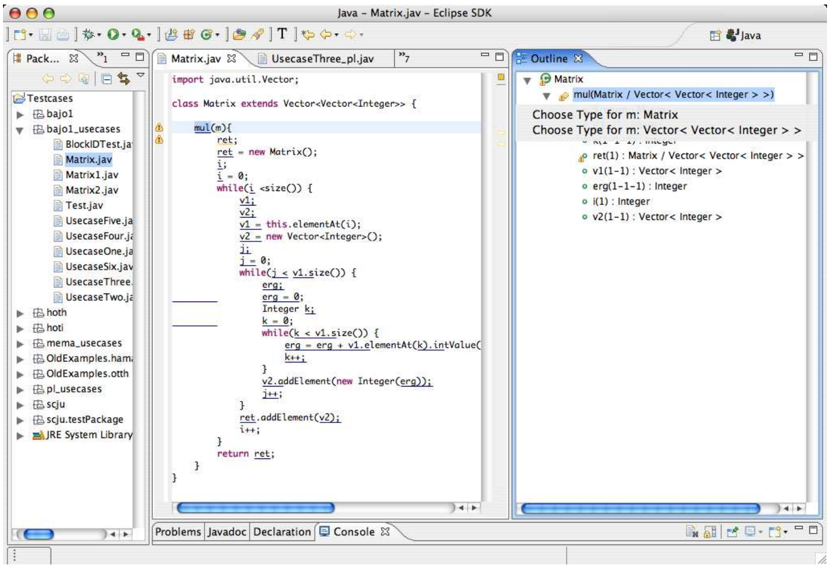
Basis des Typinferenz–Algorithmus ist die Typ–Unifikation. Das Typ–Unifikations–Problem stellt sich wie folgt dar: Für zwei gegebene Typen θ_1 und θ_2 ist eine Substitution gesucht, so dass $\sigma(\theta_1) \leq^* \sigma(\theta_2)$ gilt. Es zeigt sich, dass die Typ–Unifikation für Java 5.0 Typen ohne Wildcards finitär und mit Wildcards nicht finitär ist.
Der Typ–Unifikations–Algorithmus [Plü04] für Typen ohne Wildcards ist eine Erweiterung des Algorithmus aus [MM82]. Die wesentliche Erweiterung liegt darin, dass, wenn man im ursprünglichen Algorithmus $a = \theta$ erhält, so weiss man, dass der Variable a der Term θ zugeordnet wird. Wenn man dagegen in der Erweiterung $a \leq^* \theta$ bzw. $\theta \leq^* a$ erhält, so kann man a alle Terme zuordnen die kleiner bzw. größer als θ sind. Diese Eigenschaft ist auch die Ursache dafür, dass es in bestimmten Fällen mehrere Unifikatoren gibt.

4 Typinferenz in Java 5.0

Zunächst definieren wir Typinferenz–Regeln für die abstrakte Syntax eines Kerns von Java 5.0. Dann zeigen wir die *principal type property*. Der allgemeinste Durchschnittstyp einer Methode ist der Durchschnitt aller verschiedenen allgemeinsten Typen die jeweils einzeln in Java 5.0 für die Methode ausdrückbar sind.
Aus den Typinferenz–Regeln wird dann ein Typinferenz–Algorithmus entwickelt. Der Typinferenz–Algorithmus basiert auf der Typ–Unifikation. Zunächst werden für alle zu berechnenden Typen Typvariablen als Typen angenommen. Dann werden sukzessive durch Unifikation die Typen verfeinert. Wenn die Unifikation mehrere Unifikatoren als Ergebnis liefert, wird für den zugehörigen Typ für jeden Unifikator jeweils eine Annahme gemacht. Schlägt eine Unifikation fehl, wird die zugehörige Annahme gelöscht. Alle am Ende nicht ersetzten Typvariablen werden generalisert. Besteht am Ende das Ergebnis aus verschiedenen Typannahmen, so ist ein Durchschnittstyp inferiert worden.

5 Implementierung

Ein Prototyp eines Java 5.0 Compilers mit dem Typinferenz–Algorithmus ist implementiert. Die Implementierung wurde in ein Eclipse PlugIn (rechts) integriert, so dass eine benutzerfreundliche Bedienbarkeit gegeben ist. Das GUI erlaubt es, sich die inferierten Typen anzeigen zu lassen und im Falle der Inferenz eines Durchschnittstyps, einen davon als gewünschten Typ auszuwählen.



6 Ausblick

Als nächsten Schritt wird der Typ–Unifikations–Algorithmus auf Typen mit Wildcards erweitert. Die Schwierigkeit dabei liegt darin, dass es nun unendliche Ketten in der Typordnung $\leq^*$ geben kann. Daraus folgt, dass das Typ–Unifikations Problem nicht mehr finitär ist. Für den Fall der unendlichen Anzahl von Unifikatoren als Lösung muss eine sinnvolle Auswahlstrategie entwickelt werden.
Eine weitere Überlegung ist, die Codegenerierung so anzupassen, dass es möglich wird, Methoden mit Durchschnittstypen ohne Auswahl eines bestimmten Typs in Bytecode zu übersetzen.

Literatur

[GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java^{TM} Language Specification.* The Java series. Addison-Wesley, 3rd edition, 2005.

[MM82] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.

[Plü99] Martin Plümicke. OBJ–P *The Polymorphic Extension of* OBJ–3. PhD thesis, University of Tuebingen, WSI-99-4, 1999.

[Plü04] Martin Plümicke. Type unification in Generic–Java. In Michael Kohlhase, editor, *Proceedings of 18th International Workshop on Unification (UNIF'04)*, July 2004.

[Smo89] Gert Smolka. *Logic Programming over Polymorphically Order-Sorted Types.* PhD thesis, Department Informatik, University of Kaiserslautern, Kaiserslautern, Germany, May 1989.