Wer A sagt, muss auch B sagen: Konsequente Anwendung benutzerdefinierbarer impliziter Typumwandlungen

Christian Heinlein
Studiengang Informatik
Hochschule Aalen − Technik und Wirtschaft

Implizite Typumwandlungen werden von nahezu jeder Programmiersprache in irgendeiner Form angeboten. Beispielsweise werden ganze Zahlen normalerweise bei Bedarf implizit in Gleitkommawerte umgewandelt, um „gemischte“ Arithmetik zu erlauben oder Gleitkommafunktionen wie `sqrt` direkt auf ganze Zahlen anwenden zu können. Ebenso impliziert das Prinzip der Untertyppolymorphie in objektorientierten Sprachen, dass ein Objekt einer bestimmten Klasse überall eingesetzt werden kann, wo ein Objekt einer (direkten oder indirekten) Oberklasse benötigt wird.

Neben diesen vordefinierten und unveränderlichen impliziten Typumwandlungen, bieten manche Sprachen auch die Möglichkeit, benutzerdefinierte Umwandlungen zu vereinbaren. In C++ kann man beispielsweise beliebige implizite Typumwandlungen definieren, sofern mindestens einer der beteiligten Typen benutzerdefiniert ist. Durch Kombination mit Templates lassen sich auch generische Umwandlungen definieren, zum Beispiel:

- Ein Wert eines beliebigen Typs `T` kann automatisch in eine Liste des Typs `list<T>` umgewandelt werden, die genau diesen Wert als Element enthält.
- Eine Liste eines beliebigen Typs `list<T>` kann automatisch in einen Booleschen Wert umgewandelt werden, der genau dann `true` ist, wenn die Liste nicht leer ist.

Eine wesentliche Einschränkung impliziter Typumwandlungen in C++ und anderen Sprachen ist jedoch, dass sie normalerweise nicht transitiv angewandt werden, d. h. selbst wenn es benutzerdefinierte Umwandlungen von `A` nach `B` und von `B` nach `C` gibt, wird ein Wert des Typs `A` nicht automatisch durch eine Hintereinanderausführung dieser Umwandlungen in einen Wert des Typs `C` umgewandelt. Hinzu kommt, dass für vordefinierte und benutzerdefinierte Umwandlungen unterschiedliche Regeln gelten, was die Gesamtheit der Regeln letztlich erheblich verkompliziert.

Im Vortrag soll daher ein Ansatz vorgestellt werden, bei dem vordefinierte und benutzerdefinierte Umwandlungen absolut gleich behandelt werden; insbesondere werden beide bei Bedarf transitiv angewandt. In Kombination mit generischen Umwandlungen ergeben sich daraus jedoch interessante algorithmische Probleme. Beispielsweise kann die oben erwähnte Umwandlung von `T` nach `list<T>` prinzipiell beliebig oft nacheinander angewandt werden: Da `list<T>` selbst wieder ein beliebiger Typ `T′` ist, der in `list<T′>` umgewandelt werden kann, kann `T` damit indirekt auch in `list<list<T>>` umgewandelt werden usw. Wenn es weitere Umwandlungen von `int` nach `float` und beispielsweise von `list<list<list<float>>>` nach `bool` gäbe, dann wäre `int` → `float` → `list<float>` → `list<list<float>>` → `list<list<list<float>>` → `bool` eine korrekte − endliche − Umwandlungskette, während die Kette `bool` → `list<bool>` → `list<list<bool>>` → … endlos lang „sinnlos“ fortgesetzt werden könnte. Allerdings lässt sich die allgemeine Frage, wann eine wiederholte Anwendung einer generischen Umwandlung noch „sinnvoll“ ist, nicht ganz einfach beantworten.

Eine weitere interessante Frage ist, wann eine Umwandlungskette von `A` nach `B` „besser“ ist als eine andere und daher im Zweifelsfall bevorzugt werden soll. Beispielsweise ist eine Kette, die einen Zyklus enthält − z. B. von `int` nach `float` und wieder zurück, falls beide Umwandlungen definiert wurden −, intuitiv schlechter als die entsprechende Kette ohne Zyklus. Schließlich ergeben sich interessante und schwierige Probleme, wenn implizite Umwandlungen auch bei der Ableitung aktueller Typparameter (template argument deduction in C++) berücksichtigt werden sollen: Wenn sich für einen Typparameter, der mehrmals verwendet wird, mehrere unterschiedliche Belegungen ergeben, ist es in manchen Fällen sinnvoll, den „kleinsten gemeinsamen Obertyp“ zu verwenden, d. h. denjenigen Typ, in den sich alle Belegungen durch möglichst kurze Umwandlungsketten umwandeln lassen. Diesen zu finden − sofern er überhaupt existiert und eindeutig ist −, ist wiederum schwierig.