Nahtlose und effiziente Integration großer Zahlen in eine Programmiersprache

Christian Heinlein
Studiengang Informatik
Fakultät Elektronik und Informatik
Hochschule Aalen − Technik und Wirtschaft

Programmiersprachen wie C, C++ und Java bieten unterschiedlich große Typen für ganze und Gleitkommazahlen an, z. B. `byte`, `short`, `int` und `long` für ganze Zahlen sowie `float`, `double` und eventuell `long double` für Gleitkommazahlen. Außerdem gibt es Bibliotheken wie beispielsweise die GNU Multiple Precision Library für C und C++ sowie die Klassen `BigInteger` und `BigDecimal` für Java, mit denen Berechnungen mit beliebig großen ganzen Zahlen und beliebig genauen Gleitkommazahlen durchgeführt werden können.

Aus konzeptueller Sicht ist die Bereitstellung einer Vielzahl numerischer Typen mit unterschiedlicher Größe bzw. Genauigkeit unnötig. Ein einziger ganzzahliger Typ, dessen Werte prinzipiell beliebig groß sein können, sowie ein einziger Gleitkommatyp, dessen Werte prinzipiell beliebig genau sein können, wäre ausreichend. Allerdings ist die Verwendung der entsprechenden Typen je nach Sprache relativ umständlich − beispielsweise muss man `x.add(new BigInteger("2").multiply(y))` schreiben, um die Berechnung `x + 2*y` mit Java-`BigInteger`-Werten auszudrücken − und darüber hinaus ziemlich ineffizient, sofern die tatsächlich verwendeten Werte „klein“ sind.

Im Vortrag wird gezeigt, wie in einem 32-Bit-`int`-Wert entweder eine „kleine“ ganze Zahl oder aber ein Zeiger auf eine „große“ Zahl gespeichert werden kann und wie sich die beiden Fälle effizient unterscheiden lassen, damit Operationen auf „kleinen“ Zahlen, die normalerweise deutlich überwiegen, mit möglichst geringem Zusatzaufwand ausgeführt werden können. Auf ähnliche Art und Weise kann in einem 64-Bit-`double`-Wert entweder eine Gleitkommazahl gemäß IEEE 754 mit „kleiner“ Genauigkeit oder aber ein Zeiger auf eine Zahl mit „großer“ Genauigkeit gespeichert werden.

Durch Kapselung der jeweiligen Werte in C++-Klassen mit zugehörigen überladenen Operatordefinitionen erhält man Typen zur Repräsentation beliebig großer bzw. genauer Zahlen, die syntaktisch genauso verwendet werden können wie die eingebauten Typen der Sprache und deren Operationen für „kleine“ Werte fast genauso effizient sind. Bei einer direkten Integration dieser Typen in den Compiler einer Programmiersprache sind weitergehende Optimierungsmöglichkeiten denkbar.