Ein Debugger für Concurrent Haskell

Thomas Böttcher
RWTH Aachen
boettch@informatik.rwth-aachen.de

Frank Huch
Christian-Albrechts-Universität zu Kiel
fhu@informatik.uni-kiel.de

Nebenläufigkeit ist ein gängiges Konzept moderner Programmiersprachen. Sie ermöglicht es unterschiedliche Aufgaben eines Programms in miteinander kommunizierenden Prozessen „parallel“ auszuführen. Dies ist nicht zuletzt bei der Programmierung von graphischen Benutzeroberflächen hilfreich, bei denen meist nebenläufig unterschiedliche Events ausgeführt werden können. Die meisten Implementierungen von Nebenläufigkeit (auf sequentiellen Maschinen) verwenden einen Scheduler, welcher die zur Verfügung stehende Rechenzeit auf die einzelnen Prozesse verteilt.

Den zusätzlichen Programmierkomfort erkauft man sich jedoch in der Regel mit neue Probleme: neben den aus der sequentiellen Programmierung bekannten Fehlern, können auch Fehler zwischen den nebenläufigen Prozessen auftreten, wie beispielsweise Deadlocks oder Lifelocks. Zusätzlich können diese Fehler nichtdeterministisch auftreten, da nur einzelne Schedules in eine Fehlersituation führen können. So ist es beispielsweise möglich, daß ein funktionierendes System nach der Hinzunahme eines weiteren unabhängigen Prozesses (wegen des veränderten Schedulings) in einen Deadlock läuft.

Zur Analyse von Programmfehler werden in der Regel Debugger eingesetzt. Da ein nebenläufiges Programm in mehrere Prozesse aufgeteilt ist, muß auch ein Debugger für nebenläufige Programme diese Sichtweise unterstützen und die einzelnen Prozesse und ihr Kommunikationsverhalten darstellen. Es genügt nicht einzelne Werte der einzelnen Prozesse zu beobachten. Außerdem ist auch ein Eingreifen in das Scheduling der einzelnen Prozesse zu ermöglichen, um mögliche Schedule-abhängige Fehlerfälle zu erreichen.

Wir präsentieren einen Debugger für die nebenläufige Haskell-Erweiterung Concurrent Haskell. Concurrent Haskell stellt eine Bibliothek von Funktionen zur Generierung von Prozessen und Abstraktionen zur Kommunikation (z.B. `Channels`) zur Verfügung. Die Grundidee des Debuggers ist ein Überladen dieser Bibliotheksfunktionen, so daß zusätzlich zur eigentlichen Aktion Debug-Informationen ausgegeben werden. Der Debugger stellt Prozesse, Kommunikationsabstraktionen und das Lesen (incl. Suspendieren) und Schreiben dieser Abstraktionen graphisch dar. Über die graphische Darstellung ist auch ein Eingreifen in das Scheduling der Prozesse möglich um beispielsweise Deadlocks im System zu finden und ihre Ursache zu verstehen. Der Debugger kann einfach durch die Verwendung des entwickelten Moduls `ConcurrentDebug` anstelle des Standardmoduls `Concurrent` gestartet werden.

Wir skizzieren die Grundideen der Implementierung des Debuggers und präsentieren seine Verwendung zur Analyse einer Implementierung der dinierenden Philosophen.