Ein graphischer Debugger für Haskells Software Transactional Memory

Fabian Reck

Institut für Informatik

Christian-Albrechts-Universität zu Kiel

fre@informatik.uni-kiel.de

Nebenläufige Programmierung ist ein wichtiges Mittel um verschiedene Probleme in komplexen Anwendungen zu lösen. Durch den expliziten Einsatz von *Locks* zur Gewährleistung des gegenseitigen Ausschlusses beim Zugriff auf gemeinsam genutzte Ressourcen können jedoch Probleme wie zum Beispiel *Deadlocks* auftreten. Wichtig ist auch, dass der Einsatz von Locks dazu führen kann, dass Abstraktionsebenen verloren gehen. Um diesen Problemen zu begegnen wurde das Konzept des Software Transactional Memory (STM) entwickelt. STM erlaubt es, auf einer höheren Ebene einen Block von Aktionen als *atomar* zu kennzeichnen. Das darunterliegende System sorgt dann dafür, dass sich das Programm so verhält, als würden diese Aktionen atomar ausgeführt. In Haskells Implementierung von STM wurde dieses Konzept noch um sequenzielle und alternative Komposition und die Möglichkeit einen Thread zu blockieren, bis eine bestimmte Bedingung erfüllt ist, erweitert.

Doch trotz Software Transactional Memory ist die Entwicklung von nebenläufigen Programmen noch immer schwierig. So kann es für den Entwickler recht schwierig sein, zu verstehen, wie sich komplexe Transaktionen verhalten.

Aus diesem Grund habe ich im Rahmen meiner Diplomarbeit den bereits vorhandenen graphischen Concurrent Haskell Debugger erweitert. In meinem Vortrag werde ich den erweiterten Concurrent Haskell Debugger vorstellen und demonstrieren, wie sich dadurch Transaktionen visualisieren lassen. Zusätzlich werde ich erläutern, wie sich die in den Concurrent Haskell Debugger eingebaute automatische Suche nach Deadlocks auch bei Programmen mit Software Transactional Memory einsetzen lässt.