Twilight STM in Haskell

Annette Bieniusa (Universität Freiburg)
gemeinsam mit Peter Thiemann (Universität Freiburg)
und Arie Middelkoop (Universiteit Utrecht)

Transactional Memory (TM) hat sich zu einer vielversprechende Alternative bei multi-threaded Anwendungen entwickelt. Im Vergleich zu traditionellen locking-basierten Synchronisationsverfahren verbessert es die Gesamtproduktivität des Programmierers, indem es feingranular Daten vor nebenläufigem Zugriff sichert und zugleich Modularität, Skalierbarkeit und Wiederverwendung von Code in anderem Kontext bietet. Erste industriereife Implementierungen sind mittlerweile verfügbar, die sich kompetitiv zu anderen Synchronisationsparadigmen verhalten.

Programmierer, die Transactional Memory in ihren Anwendungen einsetzen wollen, stehen jedoch weiterhin vor einem Problem: Die Verwendung von bestehenden APIs ist häufig nicht möglich, da diese in der Regel auf Protokollen mit Handshake oder Locking basieren, um I/O und andere Systemservices bzw. nebenläufige Datenstrukturen zu nutzen. Dies widerum steht im Konflikt zur Nutzung von TM Operationen.

Twilight STM ist eine Erweiterung des STM Programmiermodells. Es unterteilt Transaktionen in zwei Phasen: eine Transaktionsphase und eine *Twilight-Phase*, welche durch eine *twilight* Operation voneinander getrennt sind. Zu Begin der Twilight-Phase werden zunächst die transaktionalen Speicheroperationen auf Konsistenz geprüft, bevor die in der Transaktionsphase geschriebenen Werte für andere Transaktionen zum Lesen und Schreiben freigegeben werden. Im Vergleich zu anderen Zweiphasen-Commitprotokollen kann der Programmierer anschliessend flexibel und in Abhängigkeit der jeweiligen Anwendung auf Konflikte reagieren. Durch Korrektur der zu schreibenden Werte kann dann beispielsweise auf ein Rollback verzichtet werden. Eine erweiterte API unterstützt die Inspektionen von Konflikten und deren Korrektur bzw. Reparatur.

Darüberhinaus können in der Twilight Zone I/O Operationen ausgeführt werden. Diese externe Operationen sind direkt global sichtbar und erlauben es auch, dass Transaktionen miteinander direkt kommunizieren.

Der Vortrag stellt eine Implementierung von Twilight für Haskell vor. Durch den Einsatz von parametrisierten Monaden stellt das Typsystem eine korrekte Anwendung der API Operationen sicher. Ein Mechanismus zum Taggen erlaubt das Gruppieren von transaktionalen Variablen und erleichtert das Auffinden und Auflösen von Konflikten.

Wir zeigen ausserdem eine Formalisierung des Systems basierend auf einem einfach getypten Lambda-Kalkül, das um Monaden und Threads erweitert wurde. Das formalisierte System Λ_{Twi} dient als Grundlage zum Beweis diverser semantischer Eigenschaften von Haskell Twilight, wie beispielsweise der Serialisierbarkeit von Transaktionen oder auch der Integrität transaktional verwalteter Daten.