technische universität dortmund

Fakultät für Informatik
Lehrstuhl XIV – Software Engineering
Prof.Dr. Falk Howar
Bachelorarbeit

Bachelorarbeit

Implementierung und Verifikation von „Purely Functional Data Structures"

Ansprechpartner: Richard Stewing (richard.stewing@tu-dortmund.de)

Situation

Funktionale Datenstrukturen (DS) haben den Ruf ineffizient zu sein. Okasaki stellte aber 1996 fest, dass Lazy Evaluation erlaubt eine amortisierte Analyse durchzuführen. Dadurch konnten DS entwickelt werden, die ähnliche Laufzeit-Eigenschaften besitzen wie ihre imperativen Entsprechungen. Genauso wie imperative DS haben auch funktionale DS invarianten, die eingehalten werden müssen, um diese Laufzeiten zu erreichen.

LiquidHaskell erweitert Haskell um Refinement Types. Refinement Types erlaubt eine eingeschränkte Form von Dependent Types, die auf Theorien beschränkt sind, die durch SMT-Solver gelöst werden können.

In dieser Arbeit soll eine Implementierung in LiquidHaskell entwickelt werden, für die automatisiert die Invarianten verifiziert werden können.

Aufgabe

Die Abschlussarbeit schließt folgende Aufgaben ein:

Einarbeitung in LiquidHaskell LiquidHaskell ist eine Erweiterung von Haskell und verwendet die gleichen Werkzeuge. Der Studierende soll sich mit den Möglichkeiten und Werkzeugen von LiquidHaskell vertraut machen. Außerdem sollte der Studierende das Tutorial durcharbeiten. (Dieser Schritt erfolgt vor Anmeldung der Arbeit.)

Implementierung Der Studierende entwickelt eine Auswahl der DS in LiquidHaskell und kodiert möglichst viele Invarianten in Refinement Types. Die Auswahl der Datenstrukturen wird zwischen Betreuer und Studierendem getroffen. Invarianten, die nicht durch Refinement Types ausgedruckt werden können, sollen durch Tests belegt werden. Sämtlich beweise sollen entweder durch einen SMT-Solver erfolgen oder durch ihn verifiziert werden.

Evaluation Die Implementierung wird gemäß Implementierungsaufwand und der kodierten Invarianten evaluiert. Insbesondere sollen die Grenzen von Refinement Types, sofern vorhanden, beschrieben werden und an welchen Stellen Beweise händisch geführt werden mussten.

Voraussetzungen

Für die Arbeit sind gute Programmierkenntnisse in Haskell von Nöten. Vorkenntnisse oder Interesse im Bereich der Type-Systeme sind ebenfalls hilfreich.

Links und Literatur

- „The Role of Lazy Evaluation in Amortized Data Structures" von Chris Okasaki
- „Purely Functional Data Structures" von Chris Okasaki
- „Refinement Types For Haskell" von Vazou et. al.
- „Programming with Refinement Types - Workshop" von Jhala et. al. (`http://ucsd-progsys.github.io/lh-workshop/`). Insbesondere Kapitel 9 aus dem Tutorial ist für diese Arbeit interessant (`https://ucsd-progsys.github.io/liquidhaskell-tutorial/Tutorial_09_Case_Study_Lazy_Queues.html`).