Proving the Correctness of Rewrite Rules in LIFT Rewite-Based System

Xueying Qin - 2335466Q

October 3, 2019

1 Introduction

The approach of LIFT is to provide high performance high-level programming with code portability. LIFT achieves this via a rewrite-based system to transform and optimise programs. The aim of this project is using the proof assistant tool Agda to specify and verify a set of rewrite rules in LIFT system.

2 Motivation

The rewrite system of LIFT systematically transforms high-level algorithmic patterns into low-level high performance OpenCL code with equivalent functionality. During this process, a set of rewrite rules are applied. Ensuring the correctness of these rules is important for ensuring the functionality is not altered during the rewrite process. Currently, the correctness of these rules is only proven correct on paper and only for a subset of the used rules. Thus, in this project I will develop mechanical proofs in Agda to show the correctness of the rewrite rules.

3 Project Plan

Firstly, the data types and semantics of LIFT will be specified and a set of primitive types and semantics will be defined. The core part of this project is then to prove the type of a function remains unchanged after being rewritten by applying certain rewrite rule(s).

3.1 LIFT Data Types and Semantics

The six general data type in LIFT `NatData`, `IndexData`, `ScalarData`, `VectorData`, `ArrayData` and `TupleData` are going to be sepcified. Specifically for `ScalarData`, there are four sub-types which are `BoolData`, `IntData`, `FloatData` and `DoubleData`. In this project we are not going distinguish those subtypes in `ScalarData`.

3.2 LIFT Primitive Types and Semantics

There are a set of basic function declarations in LIFT, which define the high-level algorithmic patterns of operations on arrays, such as `map`, `split`, `join`, `reduce`, `zip`, etc. [1] They are the primitives in LIFT that are used in both high-level and low-level expression. Preserving their types to be identical after applying rewrite rules on high-level functions indicating the correctness of rewrite rules. The types and semantics of them will be defined in Agda to assistant developing proofs.

3.3 Proving LIFT Rewrite Rules

The main aim of this project is to prove the correctness of six core algorithmic rules in LIFT, which are iterate decomposition, reorder commutativity, split-join, reduction and partial reduction, simplification rules and fussion rules [2]. Other rules such as OpenCL-specific rules would be an extension to look into.

3.4 Proving ELEVATE Strategies as Extension

ELEVATE is a programming language that combines individual program transformations into programming optimisation strategies. As an extension, we would like to exam the correctness of ELEVATE strategies of combining LIFT rewrite rules.

4 Evaluation

In this project, I would like to design effective semantics for simplification of developing proofs. Moreover, developing effective lemmas to keep the modularity of proofs is also essential for this project. Also, it is important to verify if the rules are correct and potentially propose strategies to fix if there are issues.

References

[1] Robert Atkey et al. *Data Parallel Idealised Algol.*

[2] Michel Steuwer, Christian Fensch, and Christophe Dubach. "Patterns and Rewrite Rules for Systematic Code Generation (From High-Level Functional Patterns to High-Performance OpenCL Code)". In: *arXiv preprint arXiv:1502.02389* (2015).