Formalization of the Denotation Semantics of Automatic Differentiation in Coq Project Proposal

Curtis Chin Jen Sem
(5601118)

April 2020

1 Introduction

AI and machine learning research has sparked a lot of new interest in recent times due to its many applications and ability to solve very difficult problems. One of the principle techniques in practice is the use of gradient descent, which tries optimize some problem by trying to calculate the local minimum of a differentiable function.

This is regularly done using a technique called Automatic Differentiation. There has been a recent surge of interest in formulating languages for defining automatic differentiable functions. This could have many benefits such as both applying many of the established high and low level optimizations known in programming languages research, ease defining functions for use in a gradient descent optimization through higher order functions and correctness through the use of a possible type system.

We aim to formalize an extendable proof of an implementation of automatic differentiation on a simply typed lambda calculus in the Coq proof assistant, opening up further possibilities for formally proving the correctness of more complex language features in the future. Our formalization is based on a recent proof by Stanton Huot, and Vakär[?] They proved, using a denotational model of diffeological spaces, that their forward mode emulating macro is correct when applied to a simply typed lambda calculus with products, co-products and inductive types.

todopage:1

2 Background

2.1 Automatic Differentiation

One of the principal techniques used in machine learning is back propagation, which calculates the gradient of a function. The idea being to use the gradient in the gradient descent algorithm[?] Automatic is a generalization of backpropogation. Automatic differentiation has a long and rich history, where its purpose is to calculate the derivative of a function, or in other words, calculate the derivative of function described by an arbitrary program[?] So the semantics which

one would normally expect in programming language is extended with relevant concepts such as derivative values and the chain rule.

Automatic or algorithmic differentiation is beneficial over other methods of automatically calculating the derivatives of functions such as numerical differentiation or symbolic differentiation due to its balance between speed and computational complexity. There are two main modes of automatic differentiation which I will both discuss. These are namely forward and reverse mode AD. In this paper we will prove the semantics of a forward mode AD algorithm correct. The algorithm being a very simple macro on the syntax of a simply typed lambda calculus.

todopage:2

2.2 Coq

Coq is a proof assistant created by Thierry Coquand as an implementation of his calculus of constructions type theory[?] In the 30 years since it has been released, research has contributed to extending the proof assistant with additional features such as inductive and co-inductive data types[?] GADTs[?]nd advanced modular constructions for organizing immense proofs[?]

The core of calculus of constructions is based on constructive logic and so many of the laws known in classical logic are not present. Examples include the law of the excluded middle, $\forall A, A \vee \neg A$, or the law of functional extensionality, $\forall x, f(x) = g(x) \to f = g$. They can, however, be safely added to Coq without making its logic inconsistent. Due to its usefulness in proving propositions over functions, we will make use of functional extensionality in Coq.

When defining a simply typed lambda calculus in Coq, one is free to define it using an intrinsic or extrinsic formulation.

todopage:3

2.3 Denotation Semantics

The notion of denotational semantics, created by Dana Scott and Christopher Strachey[?] tries to find underlying mathematical objects able to explain the properties of programming languages. Their original search for a solution for lambda calculi led them to well-known concepts such as partial orderings and least fixed points. In this model, programs are interpreted as partial functions and computation by taking fixpoint of such functions. Non-termination on the other hand is symbolized by a value bottom that is lower in the ordering relation than any other element. This search for an underlying mathematical foundation for languages is also known as domain theory.

In this specific case, we try to find a satisfactory model we can use to show that our implementation of forward mode automatic differentiation is correct when applied to a simply typed lambda calculus.

todopage:4

todopage:5

todopage:6

todopage:7

todopage:8

todopage:9

todopage:10

todopage:11

3 Preliminary Results

3.1 Preliminary Proof

3.2 Proof Extension

4 Timetable and Planning

4.1 Deadlines

4.2 Fill in missing holes

4.3 Generalize prototype for variants and inductive types