Extending CAS with Algebraic Reductions

An MPhil project proposal

Zongzhe Yuan (*zy272*), Christ's College

Project Supervisor: Dr Timothy Griffin

1 Introduction

1.1 Reduction

Algebraic reduction, introduced by Ahnont Wongseelashote in 1977[**?**] is an unary operator $reduce : S \longrightarrow S$. The reduced problem set can be represented as $\{x|x \in S, reduce(x) = x\}$ which is also a subset of the original problem set. The idea to represent the reduced set explicitly is to form a pair $< x, Pr(x) >$ where $x \in S$ and $Pr(x) : r(x) = x$. The first example of the reduction is id which maps all the stuff to itself. Another example is the min-set where $min_{\leq}(x) = \{x \in S|\forall y \in S, \neg(y < x)\}$. Regarding to $\mathcal{P}(S)$ it works well with $\cup$ that the min-set contain all the elements and remove the element that is non-trivial set. Wongseelashote defined the reduction operator in his paper, however the definition is not constructive. Our purpose is defining the reduction constructive in Coq, and proving the related properties of the reduction operation.

1.2 Why We need Reduction

In fact, the concept of reduction can be applied in lots of places. Here gives an example, as Jeffrey Dean mentioned in his paper "MapReduce: Simplified Data Processing on Large Clusters", the whole process of some data computation can be divided in to map and $reduce$ two phrases. It seems that when we are processing data, or we are doing some calculations, the only interesting objects are the proper subset of the object families. For example, when we are doing path problems, for the most time the path that has negative value or have infinite value is not quite interesting. However, the operation we defined there is on the whole families of object. When we are defining some data structure, like semiring, we want to know that the properties (like commutative, selective and etc..) of the proper subset (the set of objects after reduction) will hold or not, or for some cases we can't do further calculation.

1.3 Why reduction is a hard problem

The problems comes out that how can we express the reduced set properly. There is a method list above that for each element in that proper subset, we make it as a pair: the element itself plus the proof that the element is inside the subset. However those expression can't hold in most of the programming languages since we don't have such proof/proposition in the most programming languages. Thus, we need to construct the proof inside the Coq, and provides friendly extraction to the out side world. The

second problem is, when we are combining reduction operation together, the expression of the element inside the result subset will become nested pairs, which is really hard to reason the correctness and properties.

2 Basic Concept

2.1 Basic Definition

Initially I need to define the basic concept. In mathematics, A binary relation R in an arbitrary sets (or classes) S (here I restrict the relationship to be inside a single set, or say the element from the same type) is a collection of ordered pairs of elements of set (type) S, which is a subset of the Cartesian product $S \times S$. In order to link the mathematical concept with the proposition in logic, I provides each relation a representation (hold or not) as a boolean value. $\Lambda S : Type.brel(\text{Binary relation}) : S \to S \to bool$. Then, I can define the basic operations for a given (but arbitrary) type: binary operation : $\Lambda S.S \to S \to S$ and unary operation : $\Lambda S.S \to S$. And I extend the relation with a product type and a reduced version. I define the composition, identity and production for a unary operation, production and reduction for binary operation. Here I define the reduction in a binary operation in three different version: reduce the result of the operation, only reduce the argument of the operation and reduce both result and argument of the operation. $bop - reduce : \forall S : Type.\forall r : unary - opS.\forall b : binary - opS.\lambda x, y : S.r(bxy)$, $bop - reduce - args : \forall S : Type.\forall r : unary - opS.\forall b : binary - opS.\lambda x, y : S.b(rx)(ry)$, $bop - full - reduce : \forall S : Type.\forall r : unary - opS.\forall b : binary - opS.\lambda x, y : S.r(b(rx)(ry))$,

2.2 Properties that we are interested in

2.2.1 Reflexive

2.2.2 Transitive

2.2.3 Congruence

2.2.4 Associative

2.2.5 Selective

2.2.6 Idempotent

2.2.7 Has Id

2.2.8 Has Annihilator

2.3 Semiring

2.4 Reduced semiring

2.5 Reduction Combinator: Produce New Reduction from Existing One