

Extending CAS with Algebraic Reductions

Zongzhe Yuan
Christ's College



*A dissertation submitted to the University of Cambridge
in partial fulfilment of the requirements for the degree of
Master of Philosophy in Advanced Computer Science*

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
UNITED KINGDOM

Email: zy272@cl.cam.ac.uk

April 17, 2018

Declaration

I Zongzhe Yuan of Christ's College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14,235

Signed:

Date:

This dissertation is copyright ©2018 Zongzhe Yuan.

All trademarks used in this dissertation are hereby acknowledged.

Abstract

This is the abstract. Write a summary of the whole thing. Make sure it fits in one page.

Contents

1	Introduction	1
1.1	Introduction to the Path Problem and Algebraic Solution	1
1.2	Reduction	3
1.3	Motivation	3
2	Background	5
2.1	Basic Definition	5
2.1.1	Properties for the Relationship	6
2.1.2	Properties for the Operator	6
2.2	Direct Product and its Properties	7
2.2.1	Properties for the Relationship	8
2.2.2	Properties for the Operator	8
2.3	Semiring and its Properties	9
2.4	Matrix Semiring and Stability	9
2.5	Distributivity and Lexicographic Product	10
2.6	Global optimality VS Left/Right local optimality	11
2.7	Combinator for Algebraic System	12
3	Design and Implementation	13
3.1	Direct (Traditional) Reduction and Reduction Theory	13
3.1.1	Basic Definition	13
3.1.2	Properties for the Relationship	14
3.1.3	Properties for the Binary Operator	14
3.2	Binary Reduced Relationship	16
3.2.1	Isomorphic to Traditional Reduction	16
3.3	Binary Full Reduced Operator	16
3.3.1	Isomorphic to Traditional Reduction	16
4	Evaluation	17
5	Summary and Conclusions	19

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to the Path Problem and Algebraic Solution

The path problem has always fascinated mathematicians and computer scientists. At the very beginning, programmer and scientists designed algorithms to solve each of the path problem. The most famous path problem is the shortest distance problem and there are several well-known algorithm that can solve such a problem: Dijkstra's algorithm, BellmanFord algorithm, A* search algorithm and FloydWarshall algorithm. People use different primitive metrics and various complicated algorithms to solve different path problems.

However, such an approach has its obvious shortcomings. Even at some point, designing a new algorithm can "steal" the ideas of the original algorithm, people must design a completely new independent algorithm in the face of each new problem (new metric), and this makes it difficult to have a generic (or framework) approach to solve this type of problem. Even if the path problem has minor changes to the problem, it is difficult for people to solve the new problem by slightly modifying existing algorithms.

Hence, lots of predecessors have found the algebraic approaches to work around this kind of problem. Using the knowledge of abstract algebra, people find that the routing problem (path problem) can be represent using a data structure called "semiring" $(S, \oplus, \otimes, \bar{0}, \bar{1})$ [1, 2, 3, 4, 5]. For example, the popular "shortest path problem" can be represented as $(S, \min, +, \infty, 0)$ [4] and the "maximal capacity path problem" can be represented as $(S, \max, \min, 0, \infty)$.

For each path problem that represented as a semiring, we can construct a corresponding matrix semiring that represent the concrete problem (the edges and the distances for the shortest path problem for example). Then using matrix multiplication and stability of the closure (the semiring), we can solve the real problem of each concrete path problem. However, the simple matrix approach can only solve the "trivial" path problem. Those complicated problem, for example the widest shortest path problem that constructed from the shortest path problem semiring and the widest path problem (maximal capacity path problem) semiring using lexicographic product, can't be solved by this "traditional" theory approach. Some times even the method can find an optimal solution, there is no guarantee to find all optimal solutions using the classical method.

Therefore, people have found a non-classical theory of algebraic path finding method, so that algebras that violated the distributive law can be accepted. This non-classical theory can handle the problem the simple classical theory can't handle, such as the problem that can't be solved by Dijkstra or Bellman-Ford. This kind of method is dedicated to finding the local optimal solution at first, and then the local optimal solution is exactly the same as the global optimal solution by some verification or some addition restriction on the computation. For example, the famous protocol, the routing information protocol which is based on distributed Bellman-Ford algorithms is one of the non-traditional theory.

However, even so, RIP will also have a series of problems. For example, when a node does not have edges connected to it, the RIP matrix calculation (without pre-setting the maximum number of calculation steps) will continue infinitely. Even if the maximum number of calculation steps is set in advance, RIP still has some deficiencies in efficiency.

So while we use another protocol BGP (Border Gateway Protocol), we add an annihilator to the entire complex semiring. At the same time, we found that when we represent the path, we may have a loop path (a node in the path has been passed more than once). So we need our problem set to change from the original path to elementary path (A path p is elementary if no node is repeated). In this process, we performed two operations $S \rightarrow S'$, and here we call it reduction in general, which comes to the main problem of our project.

1.2 Reduction

Algebraic reduction, introduced by Ahnont Wongseelashote in 1977[2] is an unary operator for a given set of problem, $reduce : S \longrightarrow S$. It has several properties, satisfying $reduce(\emptyset) = \emptyset$, $\forall A \in S, reduce(reduce(A)) = reduce(A)$ (which is called idempotent property) and $\oplus : S \times S \rightarrow S, \forall A, B \in S, reduce(reduce(A) \oplus B) = reduce(A \oplus B) = reduce(A \oplus reduce(B))$ (which is left and right invariant property) and this paper will discuss these properties in the later section.

It is hard to specify the reduced problem set in the most programming language. However, in the world of logic and those programming language that can be used to prove properties, programmer can represent the reduced problem set 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 $\langle x, Pr(x) \rangle$ where $x \in S$ is the element in the set and $Pr(x) : r(x) = x$ is the proof that the element is in the subset (the element will not change after the reduced function, otherwise it will be reduced).

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. And this reduction is used in the construction of elementary path.

1.3 Motivation

Wongseelashote defined the reduction operator in his paper[2]. However the definition there is not constructive and it is the traditional reduction. Gurney and L11 claimed reductions could be used for "non-traditional" reductions such as elementary paths and combining elementary paths with lexicographic product [6]. However, Gurney and L11 never worked out the details, and the reduction there is still not constructive.

It is worth mentioning that, the algebraic approaches that using the matrix as the computation rely on the property of the operators a lot, for example, the left and right distributivity of the semiring. However, most of the cases mentioned above are aimed at some simple problems, or the ideal situations. In reality, we need to face the problem that, in the most time for the com-

plex path problem (especially for lexicographic product). The CAS system can derive most of the properties for the new semirings from the original "simple" semirings. However, sometimes the problem set is not the original problem set the provided to us, but the subset of it. As I mentioned before, as the problem that consist of the lexicographic product of the shortest path problem semiring and the maximum capacity semiring, there exists some path that have 0 capacity which shouldn't be concerned in the solution of the problem. This kind of reduction can be represent as a unary operation on the original problem set in our paper.

Another 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.

As the method mentioned previously, the algebraic approach to the path problem is depend on the properties of semirings. If one, or some of the properties don't hold for the semiring, the algebraic approach can't guarantee to find the optimal solution (depends on the semiring structure and the operations). Thus, after we applying those reductions on the original problem set, there is no guarantee that the original properties will still hold for the new subset of problem set, and it comes to our point. The existing CAS system doesn't have the functionality to derive and prove the properties for the reduced problem. Hence, I want to figure out the relationship between the reduction and those properties for those operators on the problem set.

Furthermore, in most programming languages, it is extremely difficult to express the reduction properly, on contrast, we can represent the reduction as a proof or proposition in our proof world. The second goal to the project is to figure out the friendly-extraction to those reduction.

Chapter 2

Background

This chapter will introduce several basic concept, and some related ideas that will help us to understand the main idea of the project.

2.1 Basic Definition

In the world of logic, we need to define several basic concepts before we are really getting started. We can simply understand that a Type is a collection of terms which have the same "properties". Under a Type S , we can define a binary relationship (in most cases we will define the binary relationship of equality $=_S$ at first). In mathematics, A binary relation R in an arbitrary Type S (here I restrict the relationship to be inside a single Type, 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, which is regarded as a property in *Coq*.

$\Lambda S.brel : S \rightarrow S \rightarrow bool$.

Next we should define the operators that exist on a given Type:

binary operation : $\Lambda S.binary_op : S \rightarrow S \rightarrow S$

unary operation : $\Lambda S.unary_op : S \rightarrow S$.

2.1.1 Properties for the Relationship

For our definition of binary relationship, we need to focus on a few properties related to it.

Here we have four properties for a binary relationship for type S . As it is mentioned in the previous section, in order to define the proposition in the logic proof, the binary relationship will return a boolean value indicating that the relationship is holding or not, here we provide a arbitrary type variable S .

Then for a given relationship $=_S: S \rightarrow S \rightarrow \text{bool}$, we should define:

reflexivity: $\forall a \in S, a =_S a \equiv \text{true}$,

symmetric : $\forall a, b \in S, a =_S b \equiv b =_S a$,

transitivity: $\forall a, b, c \in S, a =_S b \equiv \text{true} \wedge b =_S c \equiv \text{true} \rightarrow a =_S c \equiv \text{true}$,

congruence : $\forall a, b, c, d \in S, a =_S b \equiv \text{true} \rightarrow c =_S d \equiv \text{true} \rightarrow a =_S c \equiv b =_S d$

2.1.2 Properties for the Operator

Next we need to discuss some of the properties of the unary operator and the binary operator that we have defined.

Properties for Unary Operator

For the unary operator we have defined, we want it to have the following properties, or we will analyse the unary operator we have defined from the following perspectives.

In fact, our unary operator is used to represent reduction, so our discussion of the properties of the unary operator is actually a discussion of some of the properties of reduction.

There are six properties that we need to be discussed. Three of them are of the same properties as the reduction mentioned in the previous section, here we give a arbitrary type variable S , a binary relationship in $S : =_S$ and our unary operation u_S .

congruence : $\forall a, b \in S, a =_S b \rightarrow u_S(a) =_S u_S(b)$,

idempotent : $\forall a \in S, u_S(u_S(a)) =_S u_S(a)$,

left_invariant: $\Lambda \oplus_S : S \rightarrow S \rightarrow S, \forall a, b \in S, u_S(a) \oplus_S b =_S a \oplus_S b$,

right_invariant: $\Lambda \oplus_S : S \rightarrow S \rightarrow S, \forall a, b \in S, a \oplus_S u_S(b) =_S a \oplus_S b$,

preserve_id : $\forall i \in S$, if i is the identity for type S , equality $=_S$ and a binary operator \oplus_S , ($\forall a \in S, a \oplus_S i =_S a =_S i \oplus_S a$) then $u_S(i) =_S i$,

preserve_annihilator : $\forall a \in S$, if i is the annihilator for type S , equality $=_S$ and a binary operator \oplus_S , $(\forall b \in S, a \oplus_S b =_S a =_S b \oplus_S a)$ then $u_S(b) =_S b$.

Properties for Binary Operator

Also for the binary operator we have defined, we hope that we will focus on the following seven properties of it. For a given type variable S , the binary relationship $=_S$ and our pre-defined binary operator $\oplus_S : S \times S \rightarrow S$, we have:

congruence: $\forall s_1, s_2, t_1, t_2 \in S, s_1 \equiv t_1 \wedge s_2 \equiv t_2 \rightarrow s_1 \oplus_S s_2 \equiv t_1 \oplus_S t_2$,

idempotent : $\forall a \in S, a \oplus_S a =_S a$,

associativity: $\forall a, b, c \in S, a \oplus_S (b \oplus_S c) \equiv (a \oplus_S b) \oplus_S c$,

commutativity: $\forall a, b \in S, a \oplus_S b = b \oplus_S a$,

selectivity: $\forall a, b \in S, a \oplus_S b \in \{a, b\}$,

hasId: $\exists \bar{0} \in S, \forall a \in S, a \oplus_S \bar{0} = a = \bar{0} \oplus_S a$, which means $\bar{0}$ is the identity for type S , equality $=_S$ and the binary operator \oplus_S ,

hasAnn: $\exists \bar{1} \in S, \forall a \in S, a \oplus_S \bar{1} = \bar{1} = \bar{1} \oplus_S a$, which means $\bar{1}$ is the identity for type S , equality $=_S$ and the binary operator \oplus_S ,

In this paper, we will mostly concentrate on those properties that are holding or not in the reduced set of problem.

2.2 Direct Product and its Properties

Here we need to mention Product Type and some properties about Product Type.

CAS will use pair to combine two semirings (or just semigroups) to form a new data structure. Therefore we must discuss the relationship between the properties of the single semirings and the properties they have been combined with.

First, as with a given Type and its binary relationship, we need to define a binary relationship to describe the relationship between product type terms.

$\Lambda S, T, =_s, =_t . brel_product : S * T \rightarrow S * T \rightarrow bool$,

$:= \lambda(s_1, t_1)(s_2, t_2). s_1 =_s s_2 \wedge t_1 =_t t_2$, (we can write this binary relationship as $=_S \times =_T$).

At the same time, we should give an expansion on existing unary operators and binary operators for product.

For two given type S and T , unary operators u_S and u_T , binary operators \oplus_S and \oplus_T :

uop_product: $S * T \rightarrow S * T := \lambda(s, t), (u_S(s), u_T(t))$,

bop_product: $S * T \rightarrow S * T \rightarrow S * T := \lambda(s_1, t_1)(s_2, t_2), (s_1 \oplus_S s_2, t_1 \oplus_T t_2)$, (we can write this new binary operator as (\oplus_S, \oplus_T)).

2.2.1 Properties for the Relationship

In fact, we also focus on whether or not some of the properties of the binary relationship previously defined apply to the product type. Although the proof of this part is not the key to our project. Fortunately, the CAS system has systematically proved the product type properties, including binary relationship and binary operator.

For two given type S and T and their corresponding binary relationship $=_S$ and $=_T$,

If both $=_S$ and $=_T$ are reflexivity, then $=_S \times =_T$ is reflexivity,

If both $=_S$ and $=_T$ are symmetric, then $=_S \times =_T$ is symmetric,

If both $=_S$ and $=_T$ are transitivity, then $=_S \times =_T$ is transitivity,

If both $=_S$ and $=_T$ are congruence, then $=_S \times =_T$ is congruence.

2.2.2 Properties for the Operator

As mentioned above, fortunately the CAS system has given us a systematic proof of the binary operator of the product type.

For two given type S and T , binary operators \oplus_S and \oplus_T :

If both \oplus_S and \oplus_T are congruence, then (\oplus_S, \oplus_T) is congruence,

If both \oplus_S and \oplus_T are commutativity, then (\oplus_S, \oplus_T) is commutativity,

If both \oplus_S and \oplus_T are associativity, then (\oplus_S, \oplus_T) is associativity,

If both \oplus_S and \oplus_T have Id $\bar{0}_S$ and $\bar{0}_T$, then (\oplus_S, \oplus_T) has Id $(\bar{0}_S, \bar{0}_T)$,

If both \oplus_S and \oplus_T have Ann $\bar{1}_S$ and $\bar{1}_T$, then (\oplus_S, \oplus_T) has Ann $(\bar{1}_S, \bar{1}_T)$.

It is worth mentioning that although both \oplus_S and \oplus_T may be selective, this does not lead to the conclusion that (\oplus_S, \oplus_T) is a selective. The explanation of selective will be mentioned later in the lexicographic product section.

2.3 Semiring and its Properties

In abstract algebra, a semiring is a data structure $(S, \oplus, \otimes, \bar{0}, \bar{1})$ where S is a set (Type) and \oplus, \otimes are two operators : $S \times S \rightarrow S$.

(S, \oplus) is a commutative semigroup (has associative property) and (S, \otimes) is a semigroup:

$$\forall a, b, c \in S, a \oplus (b \oplus c) = (a \oplus b) \oplus c, a \oplus b = b \oplus a,$$

$$\forall a, b, c \in S, a \otimes (b \otimes c) = (a \otimes b) \otimes c.$$

\oplus, \otimes are also left and right distributive on S :

$$\forall a, b, c \in S : a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c),$$

$$\forall a, b, c \in S : (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c).$$

$\bar{0}$ is the identity of \oplus and $\bar{1}$ is the identity of \otimes :

$$\forall a \in S, a \oplus \bar{0} = a = \bar{0} \oplus a,$$

$$\forall a \in S, a \otimes \bar{1} = a = \bar{1} \otimes a.$$

Finally, $\bar{0}$ is the annihilator of \otimes :

$$\forall a \in S, a \otimes \bar{0} = \bar{0} = \bar{0} \otimes a.$$

Some definition will include that $\bar{1}$ is the annihilator of \oplus :

$$\forall a \in S, a \oplus \bar{1} = \bar{1} = \bar{1} \oplus a.$$

2.4 Matrix Semiring and Stability

For each semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ (that represent the rule of a path problem), we can define a matrix semiring $(M_n(S), \oplus, \otimes, \bar{J}, \bar{I})$ to represent the concrete path problem.

$M_n(S)$ is a $n \times n$ matrices over S ,

$$(A \oplus B)(i, j) = A(i, j) \oplus B(i, j),$$

$$(A \otimes B)(i, j) = \bigoplus_{1 \leq q \leq n} A(i, q) \otimes B(q, j),$$

$$\bar{J}(i, j) = \bar{0},$$

$$\bar{I}(i, j) = \begin{cases} \bar{1} & i = j \\ \bar{0} & otherwise \end{cases}$$

So here we can easily use this matrix to encode any specific path problem and use matrix multiplication to calculate our answer.

For a graph $G = (V, E)$ a directed graph and a weight function $w \in E \rightarrow S$, we can define the weight of a path $p = i_1, i_2, \dots, i_k$ is $w(p) = w(i_1, i_2) \otimes w(i_2, i_3) \otimes \dots \otimes w(i_{k-1}, i_k)$, while the empty path is given the weight of $\bar{1}$. Then we can define the adjacency matrix A as

$$A(i, j) = \begin{cases} w(i, j) & (i, j) \in E \\ \bar{0} & otherwise \end{cases}$$

And our problem will be represent as $A^*(i, j) = \bigoplus_{p \in \pi(i, j)} w(p)$.

However, our calculations depend on the properties of the semiring.

$a \in S$, we define the powers a^k as, $a^0 = \bar{1}$ and $a^{k+1} = a \otimes a^k$.

$a \in S$, we define the closure a^* as, $a^{(k)} = a^0 \oplus a^1 \dots a^k$ and $a^* = a^0 \oplus a^1 \dots a^k \oplus \dots$.

Here we say, if there exists a q such that $a^q = a^{(q+1)}$, then a is q -stable, which means $a^q = a^*$.

And if we know that S is 0-stable, then $M_n(S)$ is $n - 1$ -stable (because we can ignore paths with loops, and that is our reduction!). This allows us to actually calculate at most $n - 1$ steps when calculating path problems.

Therefore we can define the power and closure on the matrix semiring,

$A \in M_n(S)$, we define the powers A^k as, $A^0 = \bar{I}$ and $A^{k+1} = A \otimes A^k$.

$A \in M_n(S)$, we define the closure A^* as, $A^{(k)} = A^0 \oplus A^1 \dots A^k$ and $A^* = A^0 \oplus A^1 \dots A^k \oplus \dots$.

Hence we have $\pi(i, j)$ which is the set of paths from i to j , then $\pi^k(i, j)$ will be the set of paths from i to j with exactly k arcs, then $\pi^{(k)}(i, j)$ will be the set of paths from i to j with at most k arcs.

Then we have

$$A^k(i, j) = \bigoplus_{p \in \pi^k(i, j)} w(p),$$

$$A^{(k)}(i, j) = \bigoplus_{p \in \pi^{(k)}(i, j)} w(p) \text{ and}$$

$$A^*(i, j) = \bigoplus_{p \in \pi^*(i, j)} w(p).$$

It is worth mentioning that $A^*(i, j)$ may not be well defined, but if $M_n(S)$ is k stable, then for $A \in M_n(S)$, $A^*(i, j) = A^{(k)}(i, j)$.

2.5 Distributivity and Lexicographic Product

It is worth mentioning that the properties of distributivity (left and right) play an important role in the computation of routing problem. We've defined $M_n(S)$ the matrix and its related semiring on the given semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$. However we still need to check the properties of $M_n(S)$, to make sure that it is exactly a semiring. Consider about the distributivity properties, $M_n(S)$ is

left/right distributive if S has the distributivity properties.

But some times when we are defining some complex routing problem, such as the widest-shortest path problem, which is constructed by shortest path problem semiring and widest path problem semiring by lexicographic product, it is not guarantee that the new data structure still have the properties of distributivity.

Suppose (S, \oplus_S) is a commutative and selective semigroup and (T, \oplus_T) is a semigroup, then the lexicographic product of two semigroups $(S, \oplus_S) \bar{\times} (T, \oplus_T) = (S \times T, \oplus_{\bar{\times}})$ where

$$(s_1, t_1) \oplus_{\bar{\times}} (s_2, t_2) = \begin{cases} (s_1 \oplus_S s_2, t_1 \oplus_T t_2) & s_1 = s_1 \oplus_S s_2 = s_2 \\ (s_1 \oplus_S s_2, t_1) & s_1 = s_1 \oplus_S s_2 \neq s_2 \\ (s_1 \oplus_S s_2, t_2) & s_1 \neq s_1 \oplus_S s_2 = s_2 \end{cases}$$

As mentioned above, we used the lexicographic product when constructing the widest shortest path problem, and used the elementary path reduction. We also added a distinct annihilator in order to prevent excessive useless calculations.

These constructs make it difficult to speculate about the properties of our semiring (or new data structure), which is what our project is devoted to research that the relationship between reduction and semiring properties.

2.6 Global optimality VS Left/Right local optimality

On the other hand, we also need to discuss how to make our approach embrace the algebras that violate distributivity.

The global optimality: $A^*(i, j) = \bigoplus_{p \in \pi(i, j)} w(p)$,

The left local optimality which is the distributed Bellman-Ford algorithm: $L = (A \otimes L) \oplus \bar{I}$ which is $L(i, j) = \bigoplus_{q \in V} A(i, q) \otimes L(q, j)$,

The right local optimality which is the Dijkstra's algorithm: $R = (R \otimes A) \oplus \bar{I}$ which is $L(i, j) = \bigoplus_{q \in V} R(i, q) \otimes A(q, j)$,

note that with distributivity, $M_n(S)$ is a semiring and the three optimality problems are essentially the same, the local optimal solutions are global solutions : $A^* = L = R$,

However with out distributivity, those three solutions may all exists but all distinct, and that comes to the part of our project – discussing about the relationship between reductions and semirings.

2.7 Combinator for Algebraic System

Combinator for Algebraic system (CAS)[7] is introduced in *L11*. It is a language to design algebraic systems, in which many algebraic properties are automatically received and people can combine different operators to obtain a new semiring[7]. We can also generalize a more complex path problem (in other words, we can abstract a more complex path problem with this new semiring, such as the lexicographic products [5]). CAS can easily return the properties of those combined-operation semirings and it is already defined in Coq[8] (mentioned in *L11* by Dr Timothy Griffin).

Chapter 3

Design and Implementation

3.1 Direct (Traditional) Reduction and Reduction Theory

3.1.1 Basic Definition

Initially, First of all, I first made a detailed analysis (proof) of the reduction (we call it traditional reduction) mentioned by Wongseelashote in his article [2].

For a given type S , a binary relationship $=_S$ on S , a unary operator and a binary operator u_S and \oplus_S , we define the traditional reduction as following:

$Pr(x : S) := u_S(x) =_S x$ which is the proof that, for a given element x of type S , x is in the reduced set.

$red_Type := \{x : S \& Pr(x)\}$, which is a set that all the elements in the set are pairs of the element in S itself together with the proof that the element is in the reduced set.

$red_eq : red_Type \rightarrow red_Type \rightarrow bool := \lambda p_1 p_2, projT_1(p_1) =_S projT_1(p_2)$.

Here $projT_1/projT_2$ are two auxiliary function that map the element/the proof part from the reduced set.

It is worth mentioning that since the definition follows Wongseelashote in his article [2], our unary operator(reduction) needs congruence and idempotent properties. And when we discuss the relation and properties of the unary operator (reduciton) and the binary operator, we need to have two properties left_invariant and right_invariant.

With these definitions we can discuss the nature of the binary relationship of traditional reduction.

3.1.2 Properties for the Relationship

For our given type S and binary relationship $=_S$, we assume that $=_S$ on S has such properties:

- reflexive
- symmetric
- transitive
- congruence

It is easy to prove that red_{eq} on red_Type has those four properties:

- Lemma `red_ref` : reflexive `red_Type red_eq`.
- Lemma `red_sym` : symmetric `red_Type red_eq`.
- Lemma `red_cong` : congruence `red_Type red_eq red_eq`.
- Lemma `red_trans` : transitive `red_Type red_eq`.

3.1.3 Properties for the Binary Operator

Before starting our proof, we need to prove an auxiliary Lemma.

Lemma `Pr_br` : $\forall (p1 p2 : red_Type), Pr(u_S(projT1(p1) \oplus_S projT1(p2)))$.

The significance of this Lemma is that for any two elements that exist in a reduced set, we take out its element separately and execute a binary operation, and finally do a reduction process, and the result is still in the reduced set.

Because our binary operator \oplus_S is working on type S , we need a new reduced binary operator for our red_Type .

$red_bop : binary_op\ red_Type := p1\ p2, existT\ Pr(u_S(projT1(p1) \oplus_S projT1(p2)))(Pr_br\ p1\ p2)$.

Here `existT` is also an auxiliary function that take the element of type S and the proof of the element to form the an element in the reduced set. (which is inverse to `projT1` and `projT2`)

Here we assume that our binary operator \oplus_S has such properties separately on S :

- congruence
- associativity

- commutativity
- selective
- idempotent
- has id $\bar{0}$
- has annihilator $\bar{1}$

It is easy to proof that red_{bop} has such properties on red_{Type} :

- Lemma $red_bop_cong : congruence \rightarrow bop_congruence\ red_Type\ red_eq\ red_bop$.
- Lemma $red_bop_ass : associativity \rightarrow bop_associative\ red_Type\ red_eq\ red_bop$.
- Lemma $red_bop_comm : commutativity \rightarrow bop_commutative\ red_Type\ red_eq\ red_bop$.
- Lemma $red_bop_sel : selective \rightarrow bop_selective\ red_Type\ red_eq\ red_bop$.
- Lemma $red_bop_idem : idempotent \rightarrow bop_idempotent\ red_Type\ red_eq\ red_bop$.
- Lemma $red_bop_id :$
 $uop_preserves_id\ S =_S \oplus_S u_S \rightarrow$
 $bop_exists_id\ S =_S \oplus_S \rightarrow$
 $bop_exists_id\ red_Type\ red_eq\ red_bop$.
- Lemma $red_bop_ann :$
 $uop_preserves_ann\ S =_S \oplus_S u_S \rightarrow$
 $bop_exists_ann\ S =_S \oplus_S \rightarrow$
 $bop_exists_ann\ red_Type\ red_eq\ red_bop$.

It is worth mentioning that although the proof in this direction looks very obvious, those proof are not all sufficient. For example, sometimes red_{bop} can be commutative on red_{Type} even if \oplus_S is not commutative on S .

Fortunately, in our project we only care about whether our reduced set has the same properties when the original problem set has certain properties. So after our construction, we will specify the properties of the original problem set.

However, as we mentioned in the introduction, there are quite a few problems with this traditional reduction.

First of all, the proof part can not be well expressed in the outside world. We must know that

many programming languages do not have the ability to represent and prove proposition.

Second, when we apply a lot of reduction at the same time, this traditional representation will make our reduced set winding and confusing. For example, suppose we have two reductions r_1 and r_2 , then the reduced problem set that apply reduction 2 after reduction 1 on the problem set must be represent as $\{x \in \{y \in S \& r_1(y) = y\} \& r_2(x) = x\}$. At the same time we need to pay attention that the type of r_2 must be $\{y \in S \& r_1(y) = y\} \rightarrow \{y \in S \& r_1(y) = y\}$, but it should be $S \rightarrow S$ by our definition.

Because of the above problems, we have to consider other ways to represent our reduction (this is also the purpose and significance of our project).

3.2 Binary Reduced Relationship

3.2.1 Isomorphic to Traditional Reduction

3.3 Binary Full Reduced Operator

3.3.1 Isomorphic to Traditional Reduction

Chapter 4

Evaluation

Chapter 5

Summary and Conclusions

Bibliography

- [1] B. A. CARR. An Algebra for Network Routing Problems. *IMA Journal of Applied Mathematics*, 7(3):273–294, June 1971.
- [2] Ahnont Wongseelashote. Semirings and path spaces. *Discrete Mathematics*, 26(1):55 – 78, 1979.
- [3] Seweryn Dynierowicz and Timothy G. Griffin. On the forwarding paths produced by internet routing algorithms. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.
- [4] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
- [5] Alexander JT Gurney and Timothy G. Griffin. Lexicographic products in metarouting. In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pages 113–122. IEEE, 2007.
- [6] Alexander James Telford Gurney. Construction and verification of routing algebras.
- [7] Timothy G. Griffin and Joo Lus Sobrinho. Metarouting. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 1–12. ACM, 2005.
- [8] The Coq proof assistant. <https://coq.inria.fr>.