Fusion on Languages
Extended Abstract

Roland Backhouse*

October 20, 2000

Abstract

Many functions on context-free languages can be expressed in the form of the least fixed point of a function whose definition mimics the grammar of the given language. Examples include the function returning the length of the shortest word in a language, and the function returning the smallest number of edit operations required to transform a given word into a word in a language.

This paper presents the basic theory that explains when a function on a context-free language can be defined in this way. The contributions are: a novel definition of a regular algebra capturing division properties, several theorems showing how complex regular algebras are built from simpler ones, and the application of fixed point theory and Galois connections to practical programming problems.

1 Introduction

A common technique for solving programming problems, particularly the more challenging problems, is to express the problem in terms of solving a system of mutually recursive equations. Having done so, a number of techniques can be used for solving the equations, ranging from simple iterative techniques to more sophisticated but more specialised elimination techniques.

A relatively straightforward and well-known example is the problem of finding shortest paths through a graph. The distances to any one node from the nodes in a graph can be expressed as a set of simultaneous equations, which equations can be solved using for example Dijkstra's shortest path algorithm [Dij59]. Another, similar but much less straightforward, problem is that of finding the *edit distance* between a word and a (context-free) language — the minimum number of edit operations required to edit the given word into a word in the given language. In the case that the language is defined by a context-free grammar, the problem can be solved by constructing a system of equations in the edit distances between each segment of the given word and each nonterminal in the given grammar [AP72]. This set of equations can then be solved using a simple iterative technique or Knuth's generalisation [Knu77] of Dijkstra's shortest path algorithm.

A stumbling block for the use of recursive equations is that there is often a very big leap from a problem's specification to the algorithm for constructing the system of simultaneous equations; the justification for the leap almost invariably involves a *post hoc* verification of the construction. Thus, whereas methods for solving the equations, once constructed, are well-known and understood, the process of constructing the equations is not. The example of edit distances just given is a good example. Indeed, the theory of context-free languages offers many examples — determining whether the language generated by a given grammar is empty or not, determining the length of a shortest word in a context-free language, determining the $FIRST$ set of each of the nonterminals in a grammar, and so on.

*School of Computer Science and Information Technology, University of Nottingham, Nottingham NG8 1BB, England, rcb@cs.nott.ac.uk, fax: +44 115 951 4254

In this paper we present a general theorem which expresses when the solution to a problem can be expressed as solving a system of simultaneous equations. We give several examples of the theorem together with several non-examples (that is, examples where the theorem is not directly applicable). The non-examples serve two functions. They highlight the gap between specification and recursive equations —we show in several cases how a small change in the specification leads to a breakdown in the solution by recursive equations— and they inform the development of a methodology for the construction of recursive equations.

There are three elements to the mathematical theory underlying this paper — the theory of Galois connections, fixed point calculus and regular algebra. A brief introduction to Galois connections and fixed point calculus is given in section 2. Section 2 concludes with the so-called *fusion* theorem; showing that this theorem is fundamental to a large number of examples is the primary novel contribution of the paper.

The novel contribution of the paper begins in section 3. Here we propose a novel definition of a regular algebra; this definition is chosen so that we can encompass within one theorem many examples of programming problems, including ones involving computations on context-free grammars as well as the standard examples on finite graphs. Several theorems are presented showing how complex regular algebras can be constructed from simpler ones. Section 4 introduces the notion of a regular homomorphism and specialises the fusion theorem of section 2 in the context of regular algebras.

Section 5 is about extending "measures" on the elements of a monoid to measures on the elements of a regular algebra. A measure is some function that imposes a (possibly partial) ordering on the elements of its domain. In order to apply the fusion theorem of section 4 we require that measures preserve the regular algebra structure. The main theorem in section 5 provides a simple test for when this is indeed the case for the proposed extension mechanism. The section is concluded by several non-trivial examples of measures on languages.

For space reasons, all proofs have been omitted; they can be found in the full version of the paper at `http://www.cs.nott.ac.uk/~rcb/papers`.

2 Galois Connections and Fixed Points

This section summarises the mathematical theory needed to understand the later sections. References are supplied for further reading. We assume familiarity with the notions of a partial ordering and complete lattice. (See, for example, [GHK$^+$ 80] for an extensive account.)

The notation we use follows the recommendations of Dijkstra and Scholten [DS90]. In particular, angle brackets delimit the scope of bound variables so that $\langle x{:}R{:}E\rangle$ denotes the function that maps a value x in the range given by predicate R to the value denoted by expression E. Also, function application is denoted by an infix dot.

2.1 Galois Connections

Several equivalent definitions can be given. The following was first introduced in [Sch53].

Definition 1 (Galois Connection) Suppose $\mathcal{A}=(A,\sqsubseteq)$ and $\mathcal{B}=(B,\preceq)$ are partially ordered sets and suppose $F\in A\leftarrow B$ and $G\in B\leftarrow A$. Then (F,G) *is a Galois connection between* $\mathcal{A}$ *and* $\mathcal{B}$ iff, for all $x\in B$ and $y\in A$,

$$F.x\sqsubseteq y \equiv x\preceq G.y \quad .$$

We refer to F as the *lower adjoint* and to G as the *upper adjoint.*

□

The concept of a Galois connection was introduced by Oystein Ore in 1944 [Ore44]. Since Ore's introduction of the general notion, Galois connections have been used in many contexts, although often without specific reference to the notion. Examples include [Lam58, HS64, Con71, HH86,

FB90]. Since the 1980's, however, the notion has become part of the everyday vocabulary of many computing scientists and its use is becoming more explicit. Examples are [CC77, MSS86, Lam94, DP90, HJ98, vK00]. Elementary examples of functions having Galois adjoints include negation of boolean values, the floor and ceiling functions, the maximum and minimum operators on sets of numbers, and weakest liberal preconditions.

We assume familiarity with the notion of supremum and infimum of a (monotonic) function. Following the tradition in regular algebra to denote binary suprema by the symbol "+", we use Σf to denote the supremum of f. We also use Πf to denote its infimum.

In many applications, knowing that a function is a lower adjoint in a Galois connection is all that is needed, and the specific definition of its upper adjoint is not relevant. In such circumstances, the following existence theorem is often used to determine that a function is indeed a lower adjoint.

Theorem 2 (Fundamental Theorem) Suppose that $\mathcal{B}$ is a poset and $\mathcal{A}$ is a complete poset. Then a monotonic function $F \in \mathcal{A} \leftarrow \mathcal{B}$ is a lower adjoint in a Galois connection equivales F is supremum-preserving.

□

The next theorem on Galois connections is described by Lambek and Scott [LS86] as "the most interesting consequence of a Galois correspondence". As such it deserves a name. The name "unity-of-opposites theorem" used here was suggested to us by Lambek's discussion of the theorem in [Lam81].

Theorem 3 (Unity of Opposites) Suppose $F \in \mathcal{A} \leftarrow \mathcal{B}$ and $G \in \mathcal{B} \leftarrow \mathcal{A}$ are Galois connected functions, F being the lower adjoint and G being the upper adjoint. Then $F.\mathcal{B}$ and $G.\mathcal{A}$ are isomorphic posets. Moreover, if one of $\mathcal{A}$ or $\mathcal{B}$ is complete then $F.\mathcal{B}$ and $G.\mathcal{A}$ are also complete.

□

2.2 Fixed Points

An immediate issue when confronted with a system of simultaneous equations is whether the system has no solutions, whether it has a unique solution or whether it has many solutions. In the case that the solution space is a complete lattice the issue is reduced to whether or not the system has more than one solution. The relevant theory is the theory of fixed points and prefix points of a monotonic endofunction.

Suppose $\mathcal{A} = (A, \sqsubseteq)$ is a partially ordered set. Assuming $\mathcal{A}$ is complete, we denote the least prefix point of f by μf.

Theorem 4 (μ-fusion) Suppose $f \in A \leftarrow B$ is the lower adjoint in a Galois connection between the complete posets $(A, \sqsubseteq)$ and $(B, \preceq)$. Suppose also that $g \in (B, \preceq) \leftarrow (B, \preceq)$ and $h \in (A, \sqsubseteq) \leftarrow (A, \sqsubseteq)$ are monotonic functions. Then

$$f.\mu g = \mu h \Leftarrow f \circ g = h \circ f \quad .$$

□

We call this theorem μ-"fusion" because it states when application of function, f, can be "fused" with a fixed point, μg, to form a fixed point, μh. (The rule is also used, of course, to "defuse" a fixed point into the application of a function to another fixed point.) The fusion rule is the basis of so-called "loop fusion" techniques in programming: the combination of two loops, one executed after the other, into a single loop. The theorem also plays a central role in the abstract interpretation of programs; Cousot and Cousot [CC77] introduced the theorem in this context in a different form.

2.3 Applying Fusion: Example and Non-Example

This section discusses two related examples. The first is an example of how the fusion theorem is applied; the second illustrates how the fusion theorem need not be *directly* applicable. Later we return to the second example and show how it can be generalised in such a way that the fusion theorem does become applicable.

Both examples are concerned with membership of a set. So, let us consider an arbitrary set $\mathcal{U}$. For each x in $\mathcal{U}$ the predicate $(x\in)$ maps a subset P of $\mathcal{U}$ to the boolean value true if x is an element of P and otherwise to false. The predicate $(x\in)$ preserves set union. That is, for all bags $\mathcal{S}$ of subsets of $\mathcal{U}$,

$$x \in \cup\mathcal{S} \equiv \langle\exists P: P\in\mathcal{S}: x\in P\rangle \quad .$$

According to the fundamental theorem, the predicate $(x\in)$ thus has an upper adjoint. Indeed, we have, for all booleans b,

$$x\in S \Rightarrow b \quad \equiv \quad S \subseteq \mathsf{if}\ b \to \mathcal{U} \,\square\, \neg b \to \mathcal{U}\backslash\{x\}\ \mathsf{fi} \quad .$$

Now suppose f is a monotonic function on sets. Let μf denote its least fixed point. The fact that $(x\in)$ is a lower adjoint means that we may be able to apply the fusion theorem to reduce a test for membership in μf to solving a recursive equation. Specifically

$$(x\in \mu f \ \equiv\ \mu g) \quad \Leftarrow \quad \langle\forall S::\ x \in f.S \ \equiv\ g.(x\in S)\rangle \quad .$$

That is, the recursive equation with underlying endofunction f is replaced by the equation with underlying endofunction g (mapping booleans to booleans) if we can establish the property

$$\langle\forall S::\ x \in f.S \ \equiv\ g.(x\in S)\rangle \quad .$$

An example of where this is always possible is testing whether the empty word is in the language defined by a context-free grammar. For concreteness, consider the grammar with just one nonterminal S and productions

$$S \quad ::= \quad aS \quad | \quad SS \quad | \quad \varepsilon$$

Then the function f maps set X to

$$\{a\}\cdot X \ \cup\ X\cdot X \ \cup\ \{\varepsilon\} \quad .$$

We compute the function g as follows:

$$\begin{array}{ll}
 & \varepsilon \in f.S \\
= & \quad \{ \quad \text{definition of } f \quad \} \\
 & \varepsilon \in (\{a\}\cdot S \ \cup\ S\cdot S \ \cup\ \{\varepsilon\}) \\
= & \quad \{ \quad \text{membership distributes through set union} \quad \} \\
 & \varepsilon \in \{a\}\cdot S \ \vee\ \varepsilon \in S\cdot S \ \vee\ \varepsilon\in\{\varepsilon\} \\
= & \quad \{ \quad \varepsilon \in X\cdot Y \ \equiv\ \varepsilon\in X \wedge \varepsilon\in Y \quad \} \\
 & (\varepsilon\in\{a\} \wedge \varepsilon\in S) \ \vee\ (\varepsilon\in S \wedge \varepsilon\in S) \ \vee\ \varepsilon\in\{\varepsilon\} \\
= & \quad \{ \quad \bullet\ \ g.b \ =\ (\varepsilon\in\{a\}\wedge b) \ \vee\ (b\wedge b) \ \vee\ \varepsilon\in\{\varepsilon\}\ , \\
 & \qquad \text{see below for why the rhs has not been} \\
 & \qquad \text{simplified further} \quad \} \\
 & g.(\varepsilon\in S) \quad .
\end{array}$$

We have thus derived that

$$\varepsilon \in \mu\langle X :: \{a\}\cdot X \ \cup\ X\cdot X \ \cup\ \{\varepsilon\}\rangle \ \equiv\ \mu\langle b :: (\varepsilon \in \{a\} \wedge b) \ \vee\ (b \wedge b) \ \vee\ \varepsilon \in \{\varepsilon\}\rangle \ .$$

Note how the definition of g has the same structure as the definition of f. Effectively set union has been replaced by disjunction and concatenation has been replaced by conjunction. Of course, g can be simplified further (to the constant function true) but that would miss the point of the example.

Now suppose that instead of taking x to be the empty word we consider any word other than the empty word. Then, repeating the above calculation with "$\varepsilon \in$" replaced everywhere by "$x \in$", the calculation breaks down at the second step. This is because the empty word is the only word x that satisfies the property

$$x \in X\cdot Y \ \equiv\ x \in X \wedge x \in Y$$

for all X and Y. Indeed, taking x to be a for illustration purposes, we have

$$a \in \mu\langle X :: \{a\}\cdot X \ \cup\ X\cdot X \ \cup\ \{\varepsilon\}\rangle \ \equiv\ \mathsf{true}$$

but

$$\mu\langle b :: (a \in \{a\} \wedge b) \ \vee\ (b \wedge b) \ \vee\ a \in \{\varepsilon\}\rangle \ \equiv\ \mathsf{false} \ .$$

This second example emphasises that the conclusion of μ-fusion — $\mu h = f.\mu g$ — demands *two* properties of f, g and h, namely that f be a lower adjoint, and that $f \circ g = h \circ f$. The rule is nevertheless very versatile since being a lower adjoint is far from being uncommon, and many algebraic properties take the form $f \circ g = h \circ f$ for some functions f, g and h. In cases when the rule is not immediately applicable we have to seek generalisations of f and/or g that do satisfy both properties. Example 26 shows how this is done in the case of the general membership test for context-free languages.

3 Regular Algebra

In this section we propose a novel definition of a regular algebra, motivated by a desire to exploit to the full the calculational properties of Galois connections. We give several examples of regular algebras and prove several theorems showing how more regular algebras can be built up from simpler algebras, these constructions also being illustrated by examples.

Our view of a regular algebra is that it is the combination of a monoid (the algebra of composition) and a complete poset (the algebra of choice), with an interface between the two structures. The interface is that composition distributes through choice in all circumstances; in other words, the product operator of the monoid structure admits left and right "division" or "factorisation" operators.

Various axiomatisations of regular algebra have been given in the past, in particular several by Conway[Con71]. Conway introduced (left and right) *factors* in the context of regular languages, exploiting implicitly the fact that concatenation functions $(L\cdot)$ and $(\cdot L)$, for given language L, are both lower adjoints. Some remarkable results in his book make significant use of the existence of factors. However, Conway did not base any of his axiomatisations on this fact. Other authors have striven for the weakest possible set of axioms that still permit equality of regular languages to be decided. See for example, Kozen [Koz91]. Such axiomatisations are, however, too weak to encompass all the applications considered in this paper.

3.1 Definition and Examples

Our definition of a regular algebra is a combination of a monoid and a complete, universally distributive lattice.

Definition 5 (Regular Algebra) Suppose $\mathcal{A} = (A, \preceq)$ is a complete lattice. (This means that the suprema and infima of all monotonic functions exist, whatever the shape set.) Denote the binary supremum operator on $\mathcal{A}$ by $\oplus$. Then $\mathcal{A}$ is said to be *universally distributive* if, for all $a \in A$, the endofunction $(\oplus a)$ is the upper adjoint in a Galois connection of the poset $\mathcal{A}$ with itself.

A *regular algebra* is a 5-tuple $(A, \otimes, \oplus, \preceq, \mathbf{0}, \mathbf{1})$ where

(a) $(A, \otimes, \mathbf{1})$ is a monoid,

(b) $(A, \preceq, \oplus, \mathbf{0})$ is a complete, universally distributive lattice with least element $\mathbf{0}$ and binary supremum operator $\oplus$,

(c) for all $a \in A$, the endofunctions $(a\otimes)$ and $(\otimes a)$ are both lower adjoints in Galois connections between $(A, \preceq)$ and itself.

□

The standard example of a complete, universally distributive lattice is a power set lattice — the set of subsets of a given set ordered by set inclusion. The main application of universal distributivity is predicting uniqueness of fixed points [DBvdW97]. This however will not be an issue in this paper.

Where necessary, we use the notation $(a\backslash)$ and $(/a)$ for the upper adjoints of the functions $(a\otimes)$ and $(\otimes a)$, respectively. Thus the rules are: for all x and y,

$$a \otimes x \preceq y \equiv x \preceq a \backslash y$$

and

$$x \otimes a \preceq y \equiv x \preceq y / a \quad .$$

The operators $\backslash$ and $/$ are called *division* operators, and we often paraphrase requirement 5(c) as *product admits (left and right) division.*

Example 6 (Bool) The set $\mathbb{B}$ containing the boolean values true and false is the carrier of a regular algebra. The ordering is implication, summation is disjoint sum and product is conjunction. The zero of product is false and its unit is true. This algebra forms the basis of decision problems.

Although very simple, we shall see in example 14 that this is not a primitive regular algebra.

□

Example 7 (Min Cost Algebra) A regular algebra that occurs frequently in problems involving some cost function has carrier the set of all positive real numbers, $\mathbb{R}^{\geq 0}$, augmented with a largest element, ∞. (That is, the carrier is $\mathbb{R}^{\geq 0} \cup \{\infty\}$.) This set forms a monoid where the product operation is defined by

$$\begin{array}{lll} x \otimes y \;=\; & \mathsf{if} & x = \infty \vee y = \infty \;\rightarrow\; \infty \\ & \Box & x \neq \infty \wedge y \neq \infty \;\rightarrow\; x + y \\ & \mathsf{fi} & \end{array}$$

and the unit of product is 0. Ordering its elements by the at-least relation, where by definition $\infty \geq x$ for all x, the set forms a complete, universally distributive lattice. The supremum is minimum. Henceforth, we denote the minimum of x and y by $x \downarrow y$. Moreover, the product operation admits division. The upper adjoint of $(\otimes y)$ is given by

$$
\begin{array}{lll}
z/y & = & \mathsf{if} \quad y = \infty \ \rightarrow \ 0 \\
 & & \Box \quad y \neq \infty \wedge z \neq \infty \ \rightarrow \ z - y \\
 & & \Box \quad y \neq \infty \wedge z = \infty \ \rightarrow \ \infty \\
 & & \mathsf{fi} \quad .
\end{array}
$$

□

Example 8 (Bottleneck Algebra) Bottleneck problems are problems with a max-min requirement. For example, if it is required to drive a high load under a number of low bridges, we want to find the maximum over all different routes of the minimum height bridge on the route. A regular algebra fundamental to bottleneck problems has carrier the set of all real numbers, augmented with largest and smallest values, ∞ and $-\infty$ respectively. The addition operator is maximum (so that the ordering relation is at-most) and the product operator is minimum. The minimum operator is easily seen to satisfy the property

$$x \downarrow y \leq z \ \equiv \ x \leq z/y$$

where

$$
\begin{array}{lll}
z/y & = & \mathsf{if} \quad y \leq z \ \rightarrow \ \infty \\
 & & \Box \quad y > z \ \rightarrow \ z \\
 & & \mathsf{fi} \quad .
\end{array}
$$

That is, the product operator in the algebra admits division.

□

3.2 All Solutions

Normally, when solving optimisation problems, we are not just interested in the optimum value of the cost function, but in determining some value in the domain of solutions that optimises the cost function. To cater for this, we consider an algebra of pairs, where the first element is the cost and the second element is a set of values that have that cost. Similarly, in decision problems (such as determining whether or not there is a path between two given points in a graph) it is not sufficient to know that the answer is yes, we also want witnessing information justifying the yes answer. Again, the solution is to consider an algebra of pairs where the first element is a boolean and the second element is a set of solutions.

An algebra of pairs is also appropriate when two criteria for optimality are combined. For example, we may wish to determine among all least-cost solutions to a given problem, a solution that optimises some other criterion, like size or weight.

Theorem 10 is the basis for the use of regular algebra in such circumstances. The theorem details the construction of an algebra of pairs in which the pairs are ordered lexicographically. In general, it is not possible to combine arbitrary regular algebras in this way; the theorem assumes conditions on the algebras that are appropriate to optimisation and decision problems. The first algebra in the pair is assumed to be a *cost* algebra, as defined below.

Definition 9 A *cost algebra* is a regular algebra with the property that the ordering on elements is total and, for all $y \neq \mathbf{0}$,

$$(x \cdot y)/y = x/y \cdot y = x = y \cdot y \backslash x = y \backslash (y \cdot x) \quad .$$

□

It is easy to verify that the algebras of examples 6 and 7 are cost algebras.

Theorem 10 (Cost Algebras) Suppose $\mathcal{R}_1$ and $\mathcal{R}_2$ are both regular algebras. Suppose further that $\mathcal{R}_1$ is a cost algebra. Define the set P to be the set of ordered pairs (x, r) where $x \in \mathcal{R}_1$, $r \in \mathcal{R}_2$ and $x = \mathbf{0}_1 \Rightarrow r = \mathbf{0}_2$. Order P lexicographically; specifically, let

$$(x, r) \preceq (y, s) \quad \equiv \quad x < y \ \vee \ (x = y \wedge r \sqsubseteq s)$$

where $\leq$ is the ordering in algebra $\mathcal{R}_1$, $\sqsubseteq$ is the ordering in algebra $\mathcal{R}_2$, and $\preceq$ is the ordering in P. Define the product operation on elements of P coordinatewise:

$$(x, r) \otimes (y, s) \ = \ (x \cdot y \ , \ r \circ s)$$

the products $x \cdot y$ and $r \circ s$ being the products in the appropriate algebra. Define addition by

$$\begin{array}{lll} (x, r) \oplus (y, s) \ = & \textsf{if} & x < y \ \rightarrow \ (x \ , \ r) \\ & \square & x = y \ \rightarrow \ (x \ , \ r + s) \\ & \square & y < x \ \rightarrow \ (y \ , \ s) \\ & \textsf{fi} & . \end{array}$$

Then $(P , \otimes , \oplus , \preceq , (\mathbf{0}_1, \mathbf{0}_2) , (\mathbf{1}_1, \mathbf{1}_2))$ is a regular algebra.

□

Example 11 An instance of theorem 10 is when we combine the min-cost algebra of example 7 with the bottleneck algebra of example 8 *in that order*. The theorem is *not* applicable when the algebras are combined in the opposite order —the bottleneck algebra does not satisfy the conditions of being a cost algebra. This has important implications for the applicability of fixed point algorithms.

Suppose we consider a network of cities connected by a number of roads. Each road has a certain length and along each road there is a low bridge. It is required to drive a high load from one of the cities to another by an "optimum" route (a route being a sequence of connecting roads).

One criterion of optimality is that we choose, among the shortest routes, a route that maximises the minimum height of bridge along the route. A second criterion of optimality is that, among the routes that maximise the minimum height of bridge along the route, we choose a shortest route.

The construction of theorem 10 is applicable to the first criterion of optimality. The elements of the algebra are ordered pairs $(\mathit{distance} , \mathit{height})$. A route with "cost" (d , h) is better than a route with "cost" (e , k) if $d < e$ or $d = e$ and $h \geq k$. As this is a regular algebra, it is possible to apply the standard all-pairs path-finding algorithm (variously known as the (Roy-)Warshall algorithm [Roy59, War62], Floyd's algorithm [Flo62] — see [BC75] for more details) to determine, for all pairs of cities, the cost of a best route between the cities.

The construction of theorem 10 is *not* applicable to the second criterion of optimality; indeed an attempt to embed the lexicographical ordering on $(\mathit{height} , \mathit{distance})$ pairs in a regular algebra fails because product does not distribute through addition.

The problem is demonstrated by the graph shown in figure 1.

Note that the best route from the city numbered 4 to the city numbered 1 is the route 4, 3, 2, 1. However, the best route from the city numbered 3 to the city numbered 1 is the direct route, 3, 1. So the route from 4 follows a suboptimal route from city 3.

□

3.3 Vector Algebras

Recursive equations typically involve several unknowns, possible even an infinite set of unknowns. This situation is modelled in a straightforward and standard way. We consider a collection of equations in a collection of unknowns as a single equation in a single unknown, that unknown

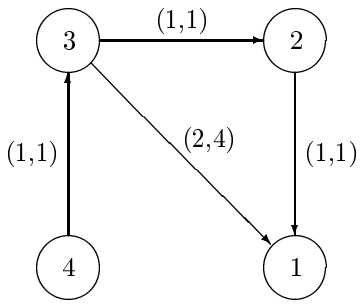

Figure 1: (Height, Distance) Graph

being a vector of values. And a vector is just a function with range the carrier set of a regular algebra. The set of functions with domain some arbitrary, fixed set and range a regular algebra forms a regular algebra if we extend the operators in the range algebra pointwise.

Theorem 12 (Vector Algebras) Suppose $\mathcal{A} = (A, \times, +, \leq, 0, 1)$ is a regular algebra and suppose $\mathcal{B} = (B, \sqsubseteq)$ is an arbitrary poset. Then $\mathcal{A} \leftarrow \mathcal{B} = (A \leftarrow B\ ,\ \dot{\times}\ ,\ \dot{+}\ ,\ \dot{\leq}\ ,\ \dot{0}\ ,\ \dot{1})$ is a regular algebra, where product, addition and ordering are defined pointwise and $\dot{0}$ and $\dot{1}$ are the constant functions returning 0 and 1, respectively.

□

3.4 From Monoids to Regular Algebras

Some important examples of regular algebras are power set algebras. These are introduced in this section.

Lemma 13 Every monoid can be extended to a power set regular algebra.

Specifically, suppose $(A, \cdot, 1)$ is a monoid. We define a regular algebra with carrier set 2^A (the set of all subsets of A) as follows. The partial ordering is the subset relation. Monoid multiplication is extended to sets by

$$X \cdot Y \ = \cup \langle x, y \colon x \in X \wedge y \in Y \colon \{x \cdot y\} \rangle$$

and $\{1\}$ is the unit.

□

Example 14 (Bool) The simplest possible example of a monoid has carrier $\{1\}$. The subsets of this set are the empty set and the set itself. The power set regular algebra is clearly isomorphic to the booleans. Choosing to map the empty set to false and $\{1\}$ to true, the product operation of the regular algebra is conjunction, and the addition operator is disjunction.

This is example 6 discussed earlier.

□

Example 15 The standard example of a regular algebra has carrier set the set of words over some finite alphabet.

A *language* over alphabet T is a set of words, i.e. a subset of T^*. The power set regular algebra constructed from the monoid $(T^*, \cdot, \varepsilon)$, where the infix dot denotes concatention of words and ε denotes the empty word, has carrier the set of all languages over alphabet T.

□

3.5 Graph Algebras

If regular algebra is to be applied to path-finding problems then it is vital that the property of being a regular algebra can be extended to graphs/matrices[1] [BC75].

Often, graphs are supposed to have finite dimensions. In the present circumstances —the assumption of a complete lattice— there is no need to impose this as a requirement. Indeed if we are to include the algebra of binary relations over some given, possibly infinite, set in the class of regular algebras then we certainly should not require graphs to have finite dimension. Other applications demand a very general definition of a graph. In the following, a *binary relation* is just a set of pairs.

Definition 16 Suppose r is a binary relation and suppose A is a set. A *(labelled) graph of dimension r over A* is a function f with domain r and range A. Elements of relation r are called *edges*.

We will use $M_r A$ to denote the class of all labelled graphs of dimension r over A. If f is a graph and the pair (i,j) is an element of r, then $i\langle f\rangle j$ will be used to denote the application of f to the pair (i,j).

□

Addition and product of graphs involves a slight generalisation of the standard definitions of addition and product of real matrices.

Definition 17 (Addition and Product) Suppose $\mathcal{R} = (A, \times, +, \leq, 0, 1)$ is a regular algebra. Then zero and the addition and product operators of $\mathcal{R}$ can be extended to graphs as follows. Two graphs f and g of the same dimension r can be ordered according to the rule: for all pairs (i,j) in r

$$f \mathrel{\dot{\leq}} g \;\equiv\; \forall\langle i,j :: i\langle f\rangle j \leq i\langle g\rangle j\rangle \quad .$$

The supremum ordering is just pointwise. In particular, f and g of the same dimension r are added according to the rule: for all pairs (i,j) in r

$$i\langle f \dot{+} g\rangle j \;=\; i\langle f\rangle j + i\langle g\rangle j \quad .$$

Two graphs f and g of dimensions r and s can be multiplied to form a graph of dimension $r \circ s$ according to the rule: for all pairs (i,j) in $r \circ s$

$$i\langle f \dot{\times} g\rangle j \;=\; \Sigma\langle k:\; (i,k) \in r \wedge (k,j) \in s:\; i\langle f\rangle k \times k\langle g\rangle j\rangle \quad .$$

Finally, the zero graph, denoted by $\mathbf{0}$, is defined by: for all pairs (i,j) in r,

$$i\langle \mathbf{0}\rangle j \;=\; 0 \quad .$$

□

Henceforth, we use $M_r A$ to denote the class of all graphs of dimension r over A.

Theorem 18 (Graph Algebras) Suppose $\mathcal{R} = (A, \times, +, \leq, 0, 1)$ is a regular algebra with carrier A, and suppose r is a reflexive, transitive relation. Define an ordering, addition and product operators as in definition 17. Define the unit graph, denoted by $\mathbf{1}$, by

$$\begin{array}{lll} i\langle \mathbf{1}\rangle j \;=\; & \mathsf{if} & i = j \;\rightarrow\; 1 \\ & \Box & i \neq j \;\rightarrow\; 0 \\ & \mathsf{fi} & . \end{array}$$

[1] For us, the words *graph* and *matrix* are interchangeable. In some applications "graph" is the word that is traditionally used, in others "matrix" is more conventional. For consistency we use "graph" everywhere throughout this paper.

(Note that M_rA is closed under the product operation and contains $\mathbf{1}$ on account of the assumptions that r is transitive and reflexive, respectively.)

Then the algebra $M_r\mathcal{R} = (M_rA, \dot{\times}, \dot{+}, \dot{\leq}, \mathbf{0}, \mathbf{1})$ so defined is a regular algebra.

□

There are several important examples of graph regular algebras. The binary relations on some set A is one example. The underlying regular algebra is the booleans $\mathbb{B}$, and the edge relation is the cartesian product $A \times A$. The relation represented by graph f is the set of pairs (i,k) such that $i\langle f\rangle k$. Graph addition corresponds to the union of relations, and graph product corresponds to the composition of relations. The divisions f/g and $f\backslash g$ are called residuals [Dil39] in the mathematics literature and weakest pre and post specifications [HH86] in the computing science literature.

Path problems on finite graphs provide additional examples of graph algebras. The standard example is shortest paths: the underlying algebra is the minimum cost algebra introduced in definition 7. For further examples see [BC75, Tar81].

4 Regular Homomorphisms and the Main Theorem

In this section we specialise the fusion theorem of section 2.2 to systems of equations with the structure of a context-free grammar.

Definition 19 (Regular Homomorphism) Let $\mathcal{R} = (R, \circ, 1_R)$ and $\mathcal{S} = (S, \cdot, 1_S)$ be monoids. Suppose m is a function with domain M and range R. Then m is said to be *compositional* if, for all x and y in R,

$$m.(x \cdot y) \;=\; m.x \circ m.y \quad .$$

Also, m is said to be a *monoid homomorphism* from $\mathcal{R}$ to $\mathcal{S}$ if m is compositional and preserves units:

$$m.1_R \;=\; 1_S \quad .$$

Now let $\mathcal{R} = (R, \circ, \oplus, \preceq, 0_R, 1_R)$ and $\mathcal{S} = (S, \cdot, +, \leq, 0_S, 1_S)$ be regular algebras. Suppose m is a function with domain R and range S. Then m is said to be a *regular homomorphism* if m is a monoid homomorphism (from $(R, \circ, 1_R)$ to $(S, \cdot, 1_S)$) and it is a lower adjoint in a Galois connection between the two orderings.

□

For m to be a regular homomorphism it must be compositional and preserve the unit of $\mathcal{R}$. However, the latter requirement is redundant if we restrict attention to the values in the image of R under m. After all, we have

$$m.x = m.(x \circ 1_R) \;=\; m.x \cdot m.1_R$$

and, similarly,

$$m.y = m.(1_R \circ y) \;=\; m.1_R \cdot m.y$$

so that $(m.R, \cdot, m.1_R)$ is a monoid, where $m.R$ denotes the image of the set R under m. In more complicated applications this observation becomes important. Formally, we combine the observation with the unity-of-opposites theorem in the following theorem.

Theorem 20 (Range Algebras) Let $\mathcal{R} = (R, \circ, \oplus, \preceq, 0_R, 1_R)$ and $\mathcal{S} = (S, \cdot, +, \leq, 0_S, 1_S)$ be regular algebras. Suppose m is a function with domain R and range S that is compositional and is a lower adjoint in a Galois connection between the orderings. Let $m.R$ be the image of R

under m and let $m^\sharp$ denote its upper adjoint. Then $m.\mathcal{R} = (m.R\ ,\cdot\ ,\ \boxplus\ ,\leq\ ,\ 0_S\ ,\ m.1_R)$ is a regular algebra, where

$$x \boxplus y\ =\ m.(m^\sharp.x \oplus m^\sharp.y)\quad .$$

Moreover, m is a regular homomorphism from $\mathcal{R}$ to $m.\mathcal{R}$.

□

Regular homomorphisms are lower adjoints and are compositional. These are exactly the conditions we need to apply the fusion theorem to systems of equations with the structure of a context-free grammar.

Theorem 21 (Fusion on Languages) Suppose G is a context-free grammar, and $\mathcal{R}$ and $\mathcal{S}$ are regular algebras. Suppose also that m is a regular homomorphism from $\mathcal{R}$ to $\mathcal{S}$. Suppose g is the endofunction obtained in the obvious way from G by interpreting concatenation and choice by the product and addition operators of $\mathcal{R}$, and by giving suitable interpretations to the symbols of the alphabet. Suppose h is obtained in the same way using $\mathcal{S}$ instead of $\mathcal{R}$. (See section 2.3 for an example.) Then

$$m\,.\,\mu g\ =\ \mu h\quad .$$

□

5 Measures

On its own, theorem 21 is difficult to use because the requirement of being a regular homomorphism are quite strong. However, the sort of functions m that we usually consider are defined by extending a function on words to a function on languages. We call such functions "measures". The sort of measures we have in mind are the length of a word, the first k symbols in a word, and the edit distance of a word from some given word. For example the length function on words is extended to the length-of-a-shortest-word function on languages.

In this section we show that the standard mechanism for extending a measure to a set results in a homomorphism of regular algebras, the only condition being that we start with a measure that is compositional. This makes theorem 21 relatively easy to use.

Theorem 22 (Monoidal Extensions) Suppose that $(\,M\,,\cdot\,,1_M\,)$ is a monoid and that $\mathcal{R}\ =\ (R\,,\cdot\,,+\ ,\leq\,,\,0_R\,,\,1_R)$ is a regular algebra. Suppose m is a function with domain M and range R. Consider the power set algebra $(\,2^M\,,\cdot\,,\cup\ ,\ \subseteq\ ,\,\phi\,,\,\{1_M\}\,)$ as defined in theorem 13. Define *measure*, the *extension* of m to subsets of M (elements of 2^M), by

$$measure.S\ =\ \Sigma\langle x{:}\, x{\in}S{:}\, m.x\rangle\quad .$$

Then *measure* is a regular homomorphism if m is compositional.

□

We consider several examples.

Example 23 (Test for Empty) Suppose we wish to determine whether a language is empty or not. Consider the regular algebra $\mathbb{B}$ (definition 6). Define the measure m of a word to be true. Then the extension *measure* of m to sets of words tests whether a language is empty or not. Specifically, by definition

$$measure.S\ \equiv\ \exists\langle u{:}\, u{\in}S{:}\,\mathsf{true}\rangle\quad .$$

That is,

$$measure.S\ \equiv\ S{\neq}\phi\quad .$$

The measure m is clearly compositional and so *measure* is a regular homomorphism, and theorem 21 can be applied.

□

Example 24 (Membership) We return to the membership problem discussed in section 2.3. Consider the regular algebra $\mathbb{B}$ (definition 6). Given a word X, define the measure m of a word u to be $u = X$. Then the extension, *measure*, of m to sets of words tests for membership of x in the set. Specifically, by definition

$$measure.S \equiv \exists\langle u: u \in S: u = X\rangle \quad .$$

That is,

$$measure.S \equiv X \in S \quad .$$

This measure is a regular homomorphism if measure m is compositional. But m is compositional equivales $\varepsilon = X$. So the only example of a membership test on sets of words that is a regular homomorphism is the so-called *nullability* test: the test whether the empty word is in the set. So only in this case can theorem 21 can be applied directly.

□

Example 25 (First sets, continued) Let k be a natural number. The construction of parsers typically involves computing the $FIRST_k$ sets of certain languages. The $FIRST_k$ set of language S is defined by

$$FIRST_k S = \{u: u \in S: F_k u\}$$

where F_k is the function that truncates a word to its first k symbols. (See example **??**.) $FIRST_k$ is the extension to sets of the measure F_k on words.

Defining the product $u \otimes v$ of two strings each of length at most k as we did in example **??**, it is clear that $F_k(x \cdot y) = F_k x \otimes F_k y$. That is, F_k is compositional. Hence $FIRST_k$ is a regular homomorphism, and theorem 21 can be applied.

□

The next two examples involve graph algebras. Note how example 26 generalises example 24.

Example 26 (General Context-Free Parsing) The general parsing algorithm invented by Cocke, Younger and Kasami exploits a regular homomorphism. (See [AU72, p332] for references to the origin of the Cocke-Younger-Kasami parsing algorithm.)

Let X be a given word and let N be the length of X. The motivating problem is to determine whether X —the input string— is in a language L given by some context-free grammar.

We use X to define a measure on words and then we extend the measure to sets and then to vectors of sets. The measure of word u is a graph of Booleans that determines which segments of X are equal to u. Specifically, let us index the symbols of X from 0 onwards. The edge relation of the graph is the set of pairs (i, j) such that $0 \leq i \leq j \leq N$ and will be denoted by seg. Note that this is a reflexive, transitive relation. For brevity we omit this constraint on i and j from now on.

Now, with i and j satisfying $0 \leq i \leq j \leq N$, let $X[i \,..\, j)$ denote the segment of word X beginning at index i and ending at index $j-1$. (So $X[i \,..\, i)$ is the empty word.) Now define[2]

$$m.u = \langle i, j:: X[i..j) = u\rangle \quad .$$

[2] Recall that $\langle i, j:: X[i \,..\, j) = u\rangle$ denotes a function mapping pair (i, j) (implicitly) satisfying $0 \leq i \leq j \leq N$ to the boolean value $X[i \,..\, j) = u$.

This defines $m.u$ to be a boolean graph with dimension the set of pairs (i, j) satisfying $0 \leq i \leq j \leq N$. The extension of the measure m to sets is

$$measure.S \;=\; \langle i, j :: \exists \langle u : u \in S : X[i..j) = u \rangle \rangle$$

so that

$$0 \langle measure.S \rangle N \;\equiv\; X \in S \quad .$$

Crucial to the correctness of the Cocke-Younger-Kasami algorithm is that m is compositional. This is proved as follows.

$$\begin{array}{ll} & m.u \times m.v \\ = & \quad \{ \quad \text{definition of } m \quad \} \\ & \langle i, j :: X[i \,..\, j) = u \rangle \times \langle i, j :: X[i \,..\, j) = v \rangle \\ = & \quad \{ \quad \text{definition of graph product in algebra } M_{seg}\mathbb{B} \quad \} \\ & \langle i, j :: \exists \langle k :: X[i \,..\, k) = u \wedge X[k \,..\, j) = v \rangle \rangle \\ = & \quad \{ \quad \text{word calculus} \quad \} \\ & \langle i, j :: X[i \,..\, j) = u \cdot v \rangle \\ = & \quad \{ \quad \text{definition of } m \quad \} \\ & m.(u \cdot v) \quad . \end{array}$$

We conclude that theorem 21 can be applied. Thus testing whether X is an element of the language generated by a context-free grammar G involves solving a system of order $N^2 \times k$ equations where k is the number of nonterminals in the grammar. (The total number of terms in the equations has order $N^3 \times k$.)

□

The final example is the most complicated, and requires a more complicated justification.

Example 27 (Error Repair) A general technique for error repair when parsing languages is to compute the minimum number of edit operations required to edit the input string into a string in the language being recognised [AP72]. The technique involves a generalisation of the Cocke-Younger-Kasami algorithm, similar to the generalisation that is made when going from Warshall's transitive closure algorithm to Floyd's all-shortest-paths algorithm.

Let X be a given word (the input string) and let N be the length of X. As in example 26, we use X to define a measure on words and then we extend the measure to sets. The measure of word u is a triangular graph of numbers that determines how many edit operations are required to transform each segment of X to the word u.

Transforming one word to another involves a sequence of primitive edit operations. Initially the input index, i, is set to 0; the edit operations scan the input string from left to right, transforming it to the output string. The allowed edit operations and their effect on the input string are

- $Insert(a)$. Insert symbol a after the current symbol in the output string.
- $Delete$. Increment the index i.
- $ChangeTo(a)$. Increment the index i and add symbol a to the end of the output string.
- OK. Copy the symbol at index i of the input to the output. Then increment i.

(We will see that the choice of allowed edit operations is crucial to the correctness of the generalised algorithm.)

Let $dist(u,v)$ denote the minimum number of non-OK edit operations needed to transform word u into word v using a sequence of the above edit operations. Now define

$$m.u \;=\; \langle i, j{::}\; dist(X[i..j)\,,u)\rangle \quad .$$

This defines $m.u$ to be a graph of numbers. The numbers, augmented by ∞, form the min-cost regular algebra discussed in example 7. Thus graphs over numbers also form a regular algebra. Taking this as the range algebra, the extension of the measure m to sets is

$$measure.S \;=\; \langle i, j{::}\; \downarrow\langle u{:}\, u{\in}S{:}\, dist(X[i..j)\,,u)\rangle\rangle$$

so that $0\langle measure.S\rangle N$ is the minimum number of edit operations required to repair the word X to a word in S.

Crucial to the correctness of the generalised Cocke-Younger-Kasami algorithm is that m is compositional. This follows from

$$dist(X[i..j)\;,\; u{\cdot}v) \;=\; \downarrow\langle k{::}\, dist(X[i..k)\,,u) + dist(X[k..j)\,,v)\rangle$$

which is a non-trivial property of the chosen collection of edit operations

To see that the property is non-trivial, suppose we extend the set of edit operations to allow the transposition of two adjacent characters. (Transposing characters is a very common error when using a keyboard.) Then the edit distance function is not compositional as, for example, $dist($ "ab" , "ba" $)$ is 1 —it takes one transposition to transform the word "ab" to the word "ba"— but this is not equal to the minimum of $dist($ "ab" , "b" $) + dist(\varepsilon$, "a" $)$, $dist($ "a" , "b" $) + dist($ "b" , "a" $)$ and $dist(\varepsilon$, "b" $) + dist($ "ab" , "a" $)$ —which it should be if the function m were to be compositional. Indeed, computing minimal edit distances for context-free languages is very difficult if the possibility of transpositions is included in the analysis.

This is the first example where m is compositional but not a monoid homomorphism. In an algebra of graphs with underlying algebra minimum costs the (i,j)th entry in the unit graph is ∞ whenever $i{\neq}j$. The (i,j)th entry in $m.\varepsilon$, on the other hand, is the cost of deleting all the symbols of $X\,[\,i\,..\,j\,)$. This is an instance where theorem 20 is really needed. The extension $measure$ of m is indeed a regular homomorphism; its domain is the algebra over languages and its range is the algebra of graphs in the image set of $measure$.

□

References

[AP72] A. V. Aho and T.G. Peterson. A minimum-distance error-correcting parser for context-free languages. *SIAM J. Computing*, 1:305–312, 1972.

[AU72] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation and compiling*, volume 1 of *Series in Automatic Computation*. Prentice-Hall, 1972.

[BC75] R.C. Backhouse and B.A. Carré. Regular algebra applied to path-finding problems. *Journal of the Institute of Mathematics and its Applications*, 15:161–186, 1975.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unifed lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, January 1977.

[Con71] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[DBvdW97] H. Doornbos, R.C. Backhouse, and J. van der Woude. A calculation approach to mathematical induction. *Theoretical Computer Science*, 179(1–2):103–135, 1997.

[Dij59] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[Dil39] R.P. Dilworth. Non-commutative residuated lattices. *Transactions of the American Mathematical Society*, 46:426–444, 1939.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks. Cambridge University Press, first edition, 1990.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Texts and monographs in Computer Science. Springer-Verlag, 1990.

[FB90] W.H.J. Feijen and Lex Bijlsma. Exercises in formula manipulation. In E.W. Dijkstra, editor, *Formal Development of Programs and Proofs*, pages 139–158. Addison-Wesley Publ. Co., 1990.

[Flo62] R.W. Floyd. Algorithm 97. Shortest Path. *Comm. ACM*, 5(6):345, June 1962.

[GHK+80] G. Gierz, K. H. Hofmann, K. Keimel, J.D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, 1980.

[HH86] C.A.R. Hoare and Jifeng He. The weakest prespecification. *Fundamenta Informaticae*, 9:51–84, 217–252, 1986.

[HJ98] C.A.R. Hoare and He Jifeng. *Unifying Theories of Programming*. Prentice-Hall International, 1998.

[HS64] J. Hartmanis and R.E. Stearns. Pair algebras and their application to automata theory. *Information and Control*, 7(4):485–507, 1964.

[Knu77] D.E. Knuth. A generalization of Dijkstra's shortest path algorithm. *Information Processing Letters*, 6(1):1–5, 1977.

[Koz91] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *Proc. 6th Annual IEEE Symp. on Logic in Computer Science*, pages 214–225. IEEE Society Press, 1991.

[Lam58] J. Lambek. The mathematics of sentence structure. *The American Mathematical Monthly*, 65:154–170, 1958.

[Lam81] J. Lambek. The influence of Heraclitus on modern mathematics. In J. Agassi and R.S.Cohen, editors, *Scientific Philosophy Today*, pages 111–121. D. Reidel Publishing Co., 1981.

[Lam94] J. Lambek. Some Galois connections in elementary number theory. *J. of Number Theory*, 47(3):371–377, June 1994.

[LS86] J. Lambek and P.J. Scott. *Introduction to Higher Order Categorical Logic*, volume 7 of *Studies in Advanced Mathematics*. Cambridge University Press, 1986.

[MSS86] A. Melton, D. A. Schmidt, and G. E. Strecker. Galois connections and computer science applications. In David Pitt, Samson Abramsky, Axel Poigné, and David Rydeheard, editors, *Category Theory and Computer Programming*, number 240 in Lecture Notes in Computer Science, pages 299–312. Springer-Verlag, 1986.

[Ore44] Oystein Ore. Galois connexions. *Transactions of the American Mathematical Society*, 55:493–513, 1944.

[Roy59] B. Roy. Transitivité et connexité. *C.R. Acad. Sci.*, 249:216, 1959.

[Sch53] J. Schmidt. Beitrage für filtertheorie. II. *Math. Nachr.*, 10:197–232, 1953.

[Tar81] Robert Endre Tarjan. A unified approach to path problems. *J. Assoc. Comp. Mach.*, pages 577–593, 1981.

[vK00] Burghard von Karger. A calculational approach to reactive systems. *Science of Computer Programming*, pages 139–161, 2000.

[War62] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9:11–12, 1962.