Fusion on Languages

Roland Backhouse*

February 20, 2001

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. It is shown how the theory can be applied in a methodology for programming the evaluation of such functions.

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

*School of Computer Science and Information Technology, University of Nottingham, Nottingham NG8 1BB, England

the given grammar. 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.

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. This theorem has been used earlier by Jeuring and Swierstra [JS94, JS95] in the case of some of the simpler examples discussed in this 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 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 discussed in section 2 we require that measures preserve the regular algebra structure. The main theorem in section 4 shows that this is indeed the case for the proposed extension mechanism. The section is concluded by several non-trivial examples of measures on languages.

The penultimate section, section 5 applies the fusion theorem of section 2 to the regular homomorphisms of section 4 and concludes with several examples. The paper is concluded by a discussion of the relevance of the results obtained here to programming methodology.

2 Basic Mathematics

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

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. In the field of abstract interpretation Cousot and Cousot [CC77, CC79] have done much to make the notion widely known. Examples are [CC77, MSS86, Lam94, DP90, HJ98, vK00]. 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*.

□

Many examples of Galois connections exist. Inverse functions (for example the exponential and logarithm functions) are special cases, the sets $\mathcal{A}$ and $\mathcal{B}$ being ordered by equality. More interestingly, negation of boolean values is both the lower and upper adjoint in a Galois connection between the booleans ordered by "if" and the booleans ordered by "only if", and for each boolean b, the functions $(b\wedge)$ and $(b\Rightarrow)$ are the lower and upper adjoints, respectively, in a Galois connection on the booleans ordered by "if". Other examples are the definitions of the floor and ceiling functions, the maximum and minimum operators on finite sets of numbers, weakest liberal preconditions and weakest prespecifications.

Suppose $(A,\sqsubseteq)$ and $(B,\preceq)$ are partially ordered sets and $f \in A \leftarrow B$ is a monotonic function. Then a *supremum* of f is a solution of the equation:

$$(2) \qquad x :: \quad \forall\langle a{::}\ x \sqsubseteq a \equiv \forall\langle b{::}\ f.b \sqsubseteq a\rangle\rangle \quad .$$

Equation (2) need not have a solution. However, by definition, $(A,\sqsubseteq)$ is a *complete lattice* if, for all so-called *shape* posets $(B,\preceq)$ and all monotonic functions $f \in A \leftarrow B$, (2) has a solution. If it does, for a given f, we denote its solution by Σf. By definition, then,

$$(3) \qquad \forall\langle a{::}\ \Sigma f \sqsubseteq a \equiv \forall\langle b{::}\ f.b \sqsubseteq a\rangle\rangle \quad .$$

Dually, the *infimum* of f, assuming it exists, is denoted by Πf and satisfies

$$(4) \qquad \forall\langle a{::}\ a \sqsubseteq \Pi f \equiv \forall\langle b{::}\ a \sqsubseteq f.b\rangle\rangle \quad .$$

It is well-known that the existence of all infima in a poset is guaranteed by the existence of all suprema in the poset, and vice-versa. So the definition of a complete lattice can be expressed in terms of the existence of all infima or the existence of all suprema.

Definition 5 Suppose $(A,\sqsubseteq)$ and $(B,\preceq)$ are complete lattices. Function $f \in A \leftarrow B$ is said to be *inf-preserving* if, for all posets C and all monotonic functions $g \in B \leftarrow C$,

$$f.(\Pi g) \;=\; \Pi(f \circ g) \quad .$$

Dually, $f \in A \leftarrow B$ is said to be *sup-preserving* if, for all posets C and all functions $g \in B \leftarrow C$,

$$f.(\Sigma g) \;=\; \Sigma(f \circ g) \quad .$$

□

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 6 (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 sup-preserving.

Dually, a monotonic function $G \in \mathcal{A} \leftarrow \mathcal{B}$ is an upper adjoint in a Galois connection equivales G is inf-preserving.

□

A couple of instances may help the reader to relate the theorem 6 to their own understanding. We mentioned earlier that, for all booleans p, the function $(p\wedge)$ is a lower adjoint. Specifically,

$$(p \wedge q \Rightarrow r) \;\equiv\; (p \Rightarrow (q \Rightarrow r)) \quad .$$

The supremum operator in the lattice of booleans ordered by implication is existential quantification since

$$(\exists\langle x{::}\, p.x\rangle \Rightarrow q) \;\equiv\; \forall\langle x{::}\; p.x \Rightarrow q\rangle \quad .$$

Instantiating theorem 6 we thus conclude that $(p\wedge)$ commutes with existential quantification. That is,

$$p \wedge \exists\langle x{::}\, q.x\rangle \equiv \exists\langle x{::}\; p \wedge q.x\rangle \quad .$$

The dual theorem is that $(p\Rightarrow)$ commutes with universal quantification:

$$(p \Rightarrow \forall\langle x{::}\, q.x\rangle) \;\equiv\; \forall\langle x{::}\; p \Rightarrow q.x\rangle \quad .$$

Another example is negation. Recall that

$$(\neg p \Rightarrow q) \;\equiv\; (p \Leftarrow \neg q) \quad .$$

Now, in order to instantiate theorem 6 we need to be very clear about the partially ordered sets involved: instantiate $\mathcal{A}$ to the booleans ordered by implication and $\mathcal{B}$ to the booleans ordered by follows-from. Observe that the supremum operator in $\mathcal{A}$ is existential quantification, and in $\mathcal{B}$ is universal quantification. Thus by application of the theorem:

$$\neg\forall\langle x{::}\, p.x\rangle \equiv \exists\langle x{::}\, \neg(p.x)\rangle \quad .$$

Switching $\mathcal{A}$ and $\mathcal{B}$ around we get the dual theorem:

$$\neg\exists\langle x{::}\, p.x\rangle \equiv \forall\langle x{::}\, p.x\rangle \quad .$$

This example illustrates why in general it is necessary to take great care with the precise definitions of the orderings on the posets $\mathcal{A}$ and $\mathcal{B}$ when applying theorem 6.

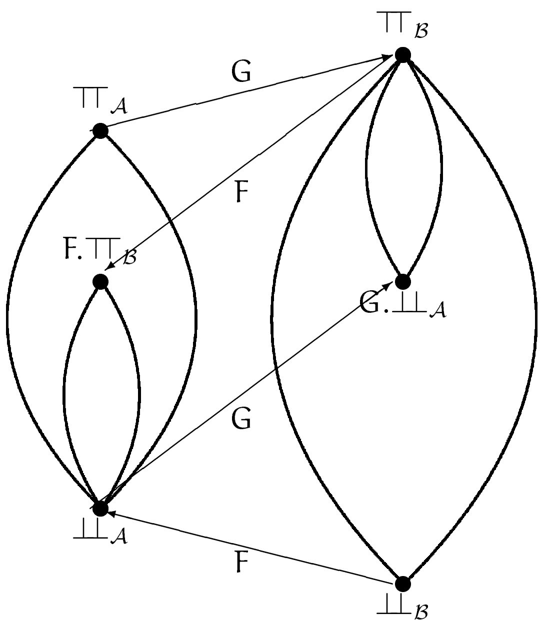
The final theorem on Galois connections is described by Lambek and Scott [LS86] as "the most interesting consequence of a Galois correspondence". As such it deserves a name. The name "unity of opposites theorem" used here was suggested to us by Lambek's discussion of the theorem in [Lam81].

Theorem 7 (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 $\mathcal{C}$-complete, for some shape poset $\mathcal{C}$, then $F.\mathcal{B}$ and $G.\mathcal{A}$ are also $\mathcal{C}$-complete. Assuming that $\mathcal{B}$ is $\mathcal{C}$-complete the supremum and infimum operators are given by

$$\begin{aligned}
\Pi_{G.\mathcal{A}}.f &= \Pi_{\mathcal{B}}.f \\
\Sigma_{G.\mathcal{A}}.f &= G.(F.(\Sigma_{\mathcal{B}}.f)) \\
\Pi_{F.\mathcal{B}}.f &= F.(\Pi_{\mathcal{B}}.(G \bullet f)) \\
\Sigma_{F.\mathcal{B}}.f &= F.(\Sigma_{\mathcal{B}}.(G \bullet f)) \quad .
\end{aligned}$$

□

Picturing the posets $\mathcal{A}$ and $\mathcal{B}$ as sets in which larger elements are above smaller elements, the unity-of-opposites theorem is itself summarised in the following diagram. The two larger lenses picture the sets $\mathcal{A}$ (on the left) and $\mathcal{B}$ (on the right); the bottom-left lens pictures $F.\mathcal{B}$ and the top-right lens $G.\mathcal{A}$. The latter two sets are pictured as having the same size because they are isomorphic, whereas $\mathcal{A}$ and $\mathcal{B}$ are pictured as having different size because they will not be isomorphic in general. Note that F maps the least element of $\mathcal{B}$ (denoted $\perp\!\!\perp_{\mathcal{B}}$ in the diagram) to the least element of $\mathcal{A}$ (denoted $\perp\!\!\perp_{\mathcal{A}}$). Further, G maps the greatest element of $\mathcal{A}$ (denoted $\top\!\!\top_{\mathcal{A}}$ in the diagram) to the greatest element of $\mathcal{B}$ (denoted $\top\!\!\top_{\mathcal{B}}$). The posets $F.\mathcal{B}$ and $G.\mathcal{A}$ are “opposites” in the sense that the former contains small elements whereas the latter contains large elements. In particular, $F.\mathcal{B}$ includes the least element of $\mathcal{A}$ and $G.\mathcal{A}$ includes the greatest element of $\mathcal{B}$. They are unified, however, by the fact that they are isomorphic.



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 and suppose f is a monotonic endofunction on $\mathcal{A}$. Then a *prefix point* of f is an element x of the carrier set A such that $f.x \sqsubseteq x$. We use $\mathrm{Pre}.f$ to denote the set of prefix points of f. A *least prefix point* of f is a solution of the equation

$$x{::}\ f.x \sqsubseteq x \wedge \forall\langle y{:}\ f.y \sqsubseteq y{:}\ x \sqsubseteq y\rangle\quad .$$

A least prefix point of f is thus a prefix point of f that is smaller than all other prefix points of f. A *least fixed point* of f is a solution of the equation

(8) $$x{::}\ f.x = x \wedge \forall\langle y{:}\ f.y = y{:}\ x \sqsubseteq y\rangle\quad .$$

Theorem 9 (Least Prefix Point) Suppose $\mathcal{A} = (A, \sqsubseteq)$ is an ordered set and $f \in A \leftarrow A$ is monotonic. Then f has at most one least prefix point, μf, characterised by the two properties:

(10) $$\mu f = f.\mu f$$

and, for all $x \in A$,

$$(11) \quad \mu f \sqsubseteq x \;\Leftarrow\; f.x \sqsubseteq x \quad .$$

□

Rule (10) states that the least prefix point of f is a fixed point of f. We call it the *computation* rule because it is often used as a left-to-right rewrite rule in the computation of μf. Rule (11) is called the *induction* rule because it is the basis for inductive proofs on sets, such as the natural numbers, defined by a fixed point equation.

In general, as suggested by the wording of theorem 9, least fixed points need not exist. A well-known theorem is that a monotonic function on a complete poset is guaranteed to have a least fixed point. (The theorem is known sometimes as Tarski's theorem and sometimes as the Knaster-Tarski theorem — see [LNS82] for historical information.) The posets we consider in this paper are always complete, allowing us to ignore the niceties of existence problems in the statement of theorems and lemmas.

The most powerful of the two rules characterising least prefix points is the induction rule. Its power is, however, somewhat limited because it only allows one to calculate with orderings in which the μ operator is the *principal* operator on the *lower* side of the ordering (i.e. orderings of the form $\mu f \sqsubseteq \cdots$). A frequently used rule, which overcomes this restriction on the induction rule, can be obtained by combining the calculation properties of Galois connections with those of fixed points.

Theorem 12 (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

$$f.\mu g = \mu h \;\Leftarrow\; f \bullet g = h \bullet f \quad .$$

More generally, if the condition

$$f \bullet g = h \bullet f$$

holds, f is the lower adjoint in a Galois connection between the posets $(\mathsf{Pre}.h\,,\sqsubseteq)$ and $(\mathsf{Pre}.g\,,\preceq)$.

□

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.

In order to apply the μ-fusion theorem we only need to know that f has an upper adjoint, we do not need to know the details of its definition. This is why theorem 6 is important as a way of determining whether a given function does have an upper adjoint.

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 and set intersection. That is, for all bags $\mathcal{S}$ of subsets of $\mathcal{U}$,

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

and

$$x\in\cap\mathcal{S} \equiv \forall\langle P{:}\, P\in\mathcal{S}{:}\, x\in P\rangle \quad .$$

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

$$x\in S \Rightarrow b \quad\equiv\quad S \subseteq \text{ if } b \rightarrow \mathcal{U} \,\Box\, \neg b \rightarrow \mathcal{U}\backslash\{x\} \text{ fi}$$

and

$$\text{if } b \rightarrow \{x\} \,\Box\, \neg b \rightarrow \phi \text{ fi} \subseteq S \quad\equiv\quad b \Rightarrow x\in S \quad .$$

(The complicated definitions of the adjoint functions illustrate why we often don't wish to know the specific details of a function's lower or upper adjoint!)

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

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

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

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

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:

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

We have thus derived that

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

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 do 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 $f{\bullet}g = h{\bullet}f$. The rule is nevertheless very versatile since being a lower adjoint is far from being uncommon, and many algebraic properties take the form $f{\bullet}g = h{\bullet}f$ 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 46 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 of the most 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 (definition 13) and a complete, universally distributive lattice (definition 17).

Definition 13 A *monoid* is a triple $(A, \cdot, 1)$, where A is a set, $\cdot$ is a binary operator and 1 is an element of A, satisfying the properties:

(14) $1{\cdot}x = x = x{\cdot}1$ for all $x{\in}A$,

and

(15) $x{\cdot}(y{\cdot}z) = (x{\cdot}y){\cdot}z$ for all $x,y,z \in A$.

The element 1 is called the *unit* of the monoid, and the binary operator is called the *product* operator.

□

There are many examples of monoids. Numbers (real, integer or natural) form monoids under addition (with unit 0) and under multiplication (with unit 1). The booleans also form a monoid under disjunction with unit false and under conjunction with unit true. Word algebras (defined below) are particularly fundamental because they form the free monoids generated by a given set. Other examples will be discussed later.

Example 16 (Word Algebra) Let T be a finite set (the *alphabet*). A *word* is a finite string formed from elements of T. The *empty word* is the string of length zero, and will be denoted by ε . The *concatenation* of words x and y, denoted by $x{\cdot}y$, is the word formed from the string x immediately followed by the string y. The set of all words over the alphabet T is denoted by T^*.

With these definitions, $(T^*, \cdot, \varepsilon)$ is a monoid.

□

Definition 17 (Universally Distributive Lattice) 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.

□

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. In general, a boolean lattice is complete and universally distributive but the converse is not true. We are not aware of any application that makes use of the additional generality of the distributivity properties of a boolean lattice. The main application of universal distributivity is predicting uniqueness of fixed points [DBvdW97]. This however will not be an issue in this paper.

Normally, the knowledge that $(\oplus a)$ is an upper adjoint is sufficient and specific details about its lower adjoint are not needed. Where we do need to denote the lower adjoint we will use $(\ominus a)$. Thus the rule is:

$$c \ominus b \preceq a \;\equiv\; c \preceq b \oplus a \quad .$$

The notation is deliberately suggestive of addition and subtraction in ordinary arithmetic in order to make the rule look familiar but, of course, care should be taken not to assume that other laws of arithmetic hold.

Definition 18 (Regular Algebra) A *regular algebra* is a 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.

□

As for addition, we try to avoid introducing specific notation for the upper adjoints of the functions $(a\otimes)$ and $(\otimes a)$. When unavoidable, we use the notation $(a\backslash)$ and $(/a)$, 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 18(c) as *product admits (left and right) division*. Again, the notation (and terminology) is suggestive of ordinary arithmetic. And, again, the same word of caution against assuming other laws of arithmetic applies.

By the fundamental theorem of Galois connections, the existence of the upper adjoints $(a\backslash)$ and $(/a)$ implies that multiplication distributes through addition. That is,

$$x \otimes (y \oplus z) \;=\; (x \otimes y) \oplus (x \otimes z)$$

and

$$(y \oplus z) \otimes x \;=\; (y \otimes x) \oplus (z \otimes x)$$

for all x, y and z. It also implies that $\mathbf{0}$ is a zero of multiplication. That is,

$$x \otimes \mathbf{0} \;=\; \mathbf{0} \;=\; \mathbf{0} \otimes x$$

for all x.

Example 19 (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 30 that this is not a primitive regular algebra.

□

Example 20 (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

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

and the unit of product is 0. (The "$+$" operation on the right side of the definition of product is ordinary addition of real numbers.) 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. To check this we have to define the upper adjoint of $(\otimes y)$ for $y \in \mathbb{R}^{\geq 0}$ and $y = \infty$. (Product is clearly symmetric. So the upper adjoint of $(y \otimes)$ is the same as the upper adjoint of $(\otimes y)$.) We do this by an exhaustive examination of the requirements. First, for all x, y and z in $\mathbb{R}^{\geq 0}$,

$$\begin{array}{ll} & x \otimes y \geq z \\ = & \{ \quad \text{definition of product} \quad \} \end{array}$$

$$\begin{array}{ll}
 & x+y \geq z \\
= & \{ \quad \text{arithmetic} \quad \} \\
 & x \geq z-y \\
= & \{ \quad x\in\mathbb{R}^{\geq 0}\text{, i.e. } x\geq 0 \quad \} \\
 & x \geq (z-y)\uparrow 0 \\
= & \{ \quad \text{for all } y\in\mathbb{R}^{\geq 0}\text{, define } (\otimes y)^{\sharp} \text{ by} \\
 & \qquad \text{for all } z\in\mathbb{R}^{\geq 0},\ (\otimes y)^{\sharp}z=(z-y)\uparrow 0 \quad \} \\
 & x \geq (\otimes y)^{\sharp}z \quad .
\end{array}$$

Second, for all y and z in $\mathbb{R}^{\geq 0}\cup\{\infty\}$,

$$\begin{array}{ll}
 & \infty\otimes y \geq z \\
= & \{ \quad \text{definition of product} \quad \} \\
 & \infty \geq z \\
= & \{ \quad \infty \text{ is by definition the largest element of } \mathbb{R}^{\geq 0}\cup\{\infty\}, \\
 & \qquad \text{so the definition of } (\otimes y)^{\sharp}z \text{ is irrelevant.} \quad \} \\
 & \infty \geq (\otimes y)^{\sharp}z \quad .
\end{array}$$

Third, for all x and z in $\mathbb{R}^{\geq 0}\cup\{\infty\}$,

$$\begin{array}{ll}
 & x\otimes\infty \geq z \\
= & \{ \quad \text{definition of product} \quad \} \\
 & \infty \geq z \\
= & \{ \quad \infty \text{ is by definition the largest element of } \mathbb{R}^{\geq 0}\cup\{\infty\} \quad \} \\
 & \mathsf{true} \\
= & \{ \quad 0 \text{ is by definition the smallest element of } \mathbb{R}^{\geq 0}\cup\{\infty\} \quad \} \\
 & x \geq 0 \\
= & \{ \quad \text{define } (\otimes\infty)^{\sharp} \text{ by} \\
 & \qquad \text{for all } z\in\mathbb{R}^{\geq 0}\cup\{\infty\},\ (\otimes\infty)^{\sharp}z=0 \quad \} \\
 & x \geq (\otimes\infty)^{\sharp}z \quad .
\end{array}$$

Finally, for all x and y in $\mathbb{R}^{\geq 0}$.

$$x\otimes y \geq \infty$$

$= \quad \{ \quad$ definition of product $\}$

$\quad \mathsf{false}$

$= \quad \{ \quad$ for all $y \in \mathbb{R}^{\geq 0}$, define $(\otimes y)^{\sharp}\infty$
by $(\otimes y)^{\sharp}\infty = \infty \quad \}$

$\quad x \geq (\otimes y)^{\sharp}\infty$.

We conclude that, for all y, $(\otimes y)$ is a lower adjoint, with upper adjoint given by

$$
\begin{array}{lll}
(\otimes y)^{\sharp}z & = & \mathsf{if} \quad y = \infty \rightarrow 0 \\
 & & \square \quad y \neq \infty \wedge z \neq \infty \rightarrow (z - y)\uparrow 0 \\
 & & \square \quad y \neq \infty \wedge z = \infty \rightarrow \infty \\
 & & \mathsf{fi} \quad .
\end{array}
$$

$\square$

Example 21 (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

$$x{\downarrow}y \leq z \equiv x \leq ({\downarrow}y)^{\sharp}z$$

where

$$
\begin{array}{lll}
({\downarrow}y)^{\sharp}z & = & \mathsf{if} \quad y \leq z \rightarrow \infty \\
 & & \square \quad y > z \rightarrow z \\
 & & \mathsf{fi} \quad .
\end{array}
$$

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

$\square$

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 23 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 22 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 Min Cost algebra defined in lemma 20 is a cost algebra since product of values different from $\mathbf{0}$ is addition of real numbers, which is clearly invertible. It is even simpler to verify that the booleans form a cost algebra. The zero is false and the only other value is true. Product is conjunction, so true is the unit of product; thus product with true is the identity function, which is invertible.

Lemma 23 In a cost algebra, for all x and y, and all $z \neq \mathbf{0}$, the following are equivalent,

$$x = y \quad , \quad x{\cdot}z = y{\cdot}z \quad , \quad z{\cdot}x = z{\cdot}y \quad , \quad x/z = y/z \quad , \quad z\backslash x = z\backslash y \quad .$$

Thus, so too are

$$x \neq y \quad , \quad x{\cdot}z \neq y{\cdot}z \quad , \quad z{\cdot}x \neq z{\cdot}y \quad , \quad x/z \neq y/z \quad , \quad z\backslash x \neq z\backslash y \quad .$$

Also, for all x and y, and all $z \neq \mathbf{0}$, the following are equivalent,

$$x \leq y \quad , \quad x{\cdot}z \leq y{\cdot}z \quad , \quad z{\cdot}x \leq z{\cdot}y \quad , \quad x/z \leq y/z \quad , \quad z\backslash x \leq z\backslash y \quad .$$

Thus, so too are

$$x < y \quad , \quad x{\cdot}z < y{\cdot}z \quad , \quad z{\cdot}x < z{\cdot}y \quad , \quad x/z < y/z \quad , \quad z\backslash x < z\backslash y \quad .$$

Proof The proof is a straightforward application of the definition of a cost algebra. Here is a part of the proof as an illustration. Assume $z \neq \mathbf{0}$. Then

$$\begin{array}{ll} & x = y \\ \Rightarrow & \{ \quad \text{Leibniz} \quad \} \\ & x{\cdot}z = y{\cdot}z \\ \Rightarrow & \{ \quad \text{Leibniz} \quad \} \\ & (x{\cdot}z)/z = (y{\cdot}z)/z \\ = & \{ \quad \bullet \quad z \neq \mathbf{0} \text{ , definition of a cost algebra} \quad \} \\ & x = y \quad . \end{array}$$

□

Theorem 24 (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:

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

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

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

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

Proof It is easily shown that P is a monoid — the product of (x,r) and (y,s) is well-defined because if x or y is $\mathbf{0}_1$ then r or s is $\mathbf{0}_2$, and $\mathbf{0}_i$ is the zero of product in the regular algebra $\mathcal{R}_i$; also, the unit of multiplication is the pair $(\mathbf{1}_1\,,\,\mathbf{1}_2)$, where $\mathbf{1}_i$ is the unit of $\mathcal{R}_i$. Second, P is complete and universally distributive — these are known properties of lexicographic orderings. Finally, it remains to show that factors of the product operation exist. Suppose w/v denotes left factoring in $\mathcal{R}_i$. That is,

$$r{\circ}s \sqsubseteq t \;\equiv\; r \sqsubseteq t/s \quad \text{and} \quad x{\cdot}y \le z \;\equiv\; y \le z/x \quad .$$

(The overloading of the symbol “ / ” is due to a lack of different symbols for division. We apologise for any confusion caused.) An obvious guess at the definition of $(\cdot(y,s))^\sharp(z,t)$ is

$$(\cdot(y,s))^\sharp(z,t) \;=\; (z/y\;,\;t/s) \quad .$$

So the proof takes the following form, where only the middle step needs to be filled in:

$$
\begin{array}{ll}
 & (x,r) \otimes (y,s) \;\preceq\; (z,t) \\
= & \qquad \{ \qquad \text{definitions of } \otimes \text{ and } \preceq \quad \} \\
 & x{\cdot}y < z \;\vee\; (x{\cdot}y = z \;\wedge\; r{\circ}s \sqsubseteq t) \\
= & \qquad \{ \qquad \text{see the case analysis below} \quad \} \\
 & x < z/y \;\vee\; (x = z/y \wedge r \sqsubseteq t/s) \\
= & \qquad \{ \qquad \text{definition of } (\cdot(y,s))^\sharp(z,t) \text{ and } \preceq \quad \} \\
 & (x,r) \preceq (\cdot(y,s))^\sharp(z,t) \quad .
\end{array}
$$

We split the verification into two cases: $y = \mathbf{0}_1$ and $y > \mathbf{0}_1$. First assume that $y = \mathbf{0}_1$. Then

$$
\begin{array}{ll}
 & x{\cdot}y < z \;\vee\; (x{\cdot}y = z \;\wedge\; r{\circ}s \sqsubseteq t) \\
= & \qquad \{ \qquad \bullet \quad y = \mathbf{0}_1\;, \qquad y = \mathbf{0}_1 \Rightarrow s = \mathbf{0}_2 \quad \} \\
 & x{\cdot}\mathbf{0}_1 < z \;\vee\; (x{\cdot}\mathbf{0}_1 = z \;\wedge\; r{\cdot}\mathbf{0}_2 \sqsubseteq t) \\
= & \qquad \{ \qquad \mathbf{0}_1 \text{ is the smallest value and is the zero of product} \quad \} \\
 & \mathsf{true} \\
= & \qquad \{ \qquad \text{for all } a,\; a/\mathbf{0} \text{ is the largest element of the algebra.} \quad \} \\
 & x < z/\mathbf{0}_1 \;\vee\; (x = z/\mathbf{0}_1 \wedge r \sqsubseteq t/\mathbf{0}_2)
\end{array}
$$

$$= \quad \{ \quad \bullet \quad y = \mathbf{0}_1 \ , \quad y = \mathbf{0}_1 \Rightarrow s = \mathbf{0}_2 \quad \}$$

$$x < z/y \ \vee \ (x = z/y \wedge r \sqsubseteq t/s) \ .$$

Now consider the case $y > \mathbf{0}_1$.

$$x{\cdot}y < z \ \vee \ (x{\cdot}y = z \ \wedge \ r{\circ}s \sqsubseteq t)$$

$$= \quad \{ \quad \bullet \quad y > \mathbf{0}_1 \ ,$$

lemma 23 applied to the term $x{\cdot}y < z$,

definition 22 applied to the term $x{\cdot}y = z$ }

$$x < z/y \ \vee \ (x = z/y \ \wedge \ r \sqsubseteq t/s) \ .$$

□

Example 25 Examples that demonstrate when a theorem is not applicable add to understanding. This is one such.

An instance of theorem 24 is when we combine the min-cost algebra of example 20 with the bottleneck algebra of example 21 *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 24 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 . Assuming the cities are numbered from 1 to N and that the distance and height of the road connecting city i to city j is recorded in the $N \times N$ graphs d and h, the graphs are updated by the following code. On termination d_{ij} is the length of a shortest route from i to j, and h_{ij} is the maximum over all routes with length d_{ij} of the minimum height of a bridge on the route.

```
for  each  k ,  1 ≤ k ≤ N
do   for  each pair ( i,j ),  1 ≤ i,j ≤ N
     do  if   d_ij < d_ik + d_kj  →  skip
         □  d_ij = d_ik + d_kj  →  h_ij  :=  h_ij ↑ (h_ik ↓ h_kj)
         □  d_ij > d_ik + d_kj  →  h_ij  :=  h_ik ↓ h_kj
         fi  ;
         d_ij  :=  d_ij ↓ (d_ik + d_kj)
     end_for
end_for
```

Note that the sequence of two assignment statements is equivalent to the single assignment

$$(h_{ij}\,,d_{ij}) \;:=\; (h_{ij}\,,d_{ij}) \oplus ((h_{ik}\,,d_{ik}) \otimes (h_{kj}\,,d_{kj}))$$

where $\oplus$ and $\otimes$ are the addition and product operators as defined in theorem 24. So the algorithm has exactly the same shape as the Roy-Warshall-Floyd algorithm.

The construction of theorem 24 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. As a consequence *the following code does **not** determine the cost of an optimal route*, although a naive comparison of this code with that above might lead one to suspect that that is the case.

```
for  each  k ,  1 ≤ k ≤ N
do   for  each pair ( i,j ),  1 ≤ i,j ≤ N
     do  if   h_ij > h_ik ↓ h_kj  →  skip
         □  h_ij = h_ik ↓ h_kj  →  d_ij  :=  d_ij ↓ (d_ik + d_kj)
         □  h_ij < h_ik ↓ h_kj  →  d_ij  :=  d_ik + d_kj
         fi  ;
         h_ij  :=  h_ij ↑ (h_ik ↓ h_kj)
     end_for
end_for   .
```

The incorrectness of the code 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. The code above correctly determines the minimum bridge height on the route from 4 to 1 but incorrectly determines the distance to be 4 rather than the correct distance 3.

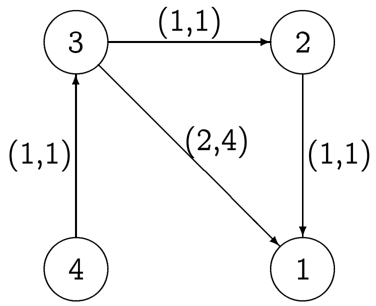


Figure 1: (Height, Distance) Graph

□

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 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. Here are the details.

Lemma 26 Suppose $\mathcal{A}=(A,\leq)$ is a complete, universally distributive lattice, and $\mathcal{B}=(B,\sqsubseteq)$ is an arbitrary poset. Then $\mathcal{A}\leftarrow\mathcal{B}$, the set of all monotonic functions with range A and domain B ordered by

$$f \mathrel{\dot{\leq}} g \;\equiv\; \forall\langle x:: f.x \leq g.x\rangle$$

is a complete, universally distributive lattice.

Proof We omit the proof that $\mathcal{A}\leftarrow\mathcal{B}$ is complete as this is a standard result of ordered set theory. (For later use, we remark that if 0 is the least element in A then $\dot{0}$ is the least element in $\mathcal{A}\leftarrow\mathcal{B}$, where $\dot{0}.x=0$ for all x.)

The binary supremum operator in the lattice $\mathcal{A}\leftarrow\mathcal{B}$, which we denote by $\dot{+}$, is defined by

$$(f\dot{+}g).x \;=\; f.x+g.x$$

where $+$ is the binary supremum operator in the lattice $\mathcal{A}$. (This is a so-called *pointwise* extension of $+$.) We have to show that $\mathcal{A}\leftarrow\mathcal{B}$ is universally distributive. That is, we have to show that $(f\dot{+})$ is an upper adjoint for all functions $f\in\mathcal{A}\leftarrow\mathcal{B}$.

We use the minus symbol to denote the lower adjoint of addition. That is, we assume that for all x, y and z in $\mathcal{A}$,

$$x\leq y+z \equiv x-y\leq z \quad .$$

(This assumption can be made because $\mathcal{A}$ is assumed to be universally distributive.) Then, for all g and h in $\mathcal{A}\leftarrow\mathcal{B}$,

$$
\begin{array}{ll}
 & g \;\dot{\leq}\; f\dot{+}h \\
\equiv & \qquad \{ \qquad \text{definition of pointwise ordering} \quad \} \\
 & \forall\langle x\!:: g.x \leq (f\dot{+}h).x\rangle \\
\equiv & \qquad \{ \qquad \text{above} \quad \} \\
 & \forall\langle x\!::\; g.x \leq f.x+h.x\rangle \\
\equiv & \qquad \{ \qquad \mathcal{A} \text{ is universally distributive. That is,} \\
 & \qquad\qquad\quad \text{for all } x,\ y \text{ and } z,\ x\leq y+z \equiv x-y\leq z \ . \quad \} \\
 & \forall\langle x\!::\; g.x-f.x \;\leq\; h.x\rangle \\
\equiv & \qquad \{ \qquad \bullet \quad \text{define } (f\dot{+})^\flat \text{ by } ((f\dot{+})^\flat g).x = g.x-f.x \quad \} \\
 & \forall\langle x\!:: ((f\dot{+})^\flat g).x \leq h.x\rangle \\
\equiv & \qquad \{ \qquad \text{definition of pointwise ordering} \quad \} \\
 & (f\dot{+})^\flat g \;\dot{\leq}\; h \quad .
\end{array}
$$

Thus the function $(f\dot{+})$ has lower adjoint $(f\dot{+})^\flat$ defined by

$$((f\dot{+})^\flat g).x \;=\; g.x-f.x \quad .$$

That is, $\mathcal{A}\leftarrow\mathcal{B}$ is a universally distributive lattice.

□

Lemma 27 Suppose $\mathcal{A} = (A, \times, +, \leq, 0, 1)$ is a regular algebra and suppose $\mathcal{B} = (B, \sqsubseteq)$ is an arbitrary poset. Define the product of monotonic functions f and g in $\mathcal{A} \leftarrow \mathcal{B}$ by

$$(f \dot{\times} g).x = f.x \times g.x$$

Then $f \dot{\times} g$ is a monotonic function in $A \leftarrow B$. Moreover, $\dot{\times}$ is associative, has unit $\dot{1}$, the constant function that always returns 1, and admits left and right division operators. That is, there are operators $\backslash$ and $/$ such that, for all monotonic functions f, g and h,

$$f \dot{\times} g \mathrel{\dot{\leq}} h \equiv f \mathrel{\dot{\leq}} h/g$$

and

$$f \dot{\times} g \mathrel{\dot{\leq}} h \equiv g \mathrel{\dot{\leq}} f \backslash h \quad .$$

Proof The proof is straightforward. For the final part it is easy to check that the pointwise definitions of h/g and $f \backslash h$ suffice:

$$(f \backslash h).x = f.x \backslash h.x \quad ,$$

and

$$(h/g).x = h.x \,/\, g.x \quad .$$

□

The two lemmas 26 and 27 have the immediate corollary that vector algebras are regular.

Theorem 28 (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 as in lemmas 26 and 27 and $\dot{0}$ and $\dot{1}$ are the constant functions returning 0 and 1, respectively.

□

Vector algebras are used when a problem involves a number of unknowns (which, of course, is the most usual case). A context-free grammar, for example, defines a language for each nonterminal in the grammar. In other words, the grammar defines a function from the set of nonterminals to languages. The appropriate is thus a vector algebra with $\mathcal{A}$ instantiated to the algebra of languages and poset $\mathcal{B}$ to the set of nonterminals ordered by equality.

3.4 From Monoids to Regular Algebras

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

Lemma 29 Every monoid can be extended to a power set regular algebra.

Proof 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

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

The easy proof that $(2^A, \cdot, \{1\})$ is a monoid is omitted, as is the proof that $(2^A, \subseteq)$ is a complete, universally distributive lattice. The proof that product admits division is straightforward:

$$
\begin{array}{ll}
 & X{\cdot}Y \subseteq Z \\
= & \quad\{ \quad \text{definition of multiplication} \quad \} \\
 & \forall\langle x,y\colon x{\in}X \wedge y{\in}Y\colon x{\cdot}y \in Z\rangle \\
= & \quad\{ \quad \text{nesting} \quad \} \\
 & \forall\langle x\colon x{\in}X\colon \forall\langle y\colon y{\in}Y\colon x{\cdot}y \in Z\rangle\rangle \\
= & \quad\{ \quad \text{set comprehension} \quad \} \\
 & \forall\langle x\colon x{\in}X\colon x \in \{x \mid \forall\langle y\colon y{\in}Y\colon x{\cdot}y \in Z\rangle\}\rangle \\
= & \quad\{ \quad \text{set inclusion} \quad \} \\
 & X \subseteq \{x \mid \forall\langle y\colon y{\in}Y\colon x{\cdot}y \in Z\rangle\} \quad .
\end{array}
$$

□

Example 30 (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 19 discussed earlier.

□

Example 31 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 $(T^*, \cdot, \varepsilon)$ (recall example 16) has carrier the set of all languages over alphabet T.

□

Example 32 (First sets) If we are only interested in the words in a language up to a certain length then a slight variation on the definition of concatenation of words is appropriate. Suppose k is a natural number. Let $T^{\leq k}$ denote the set of all words whose length is at most k. Suppose F_k is the function that truncates a word of length at least k to its first k symbols. Define the concatenation $x{\otimes}y$ of words x and y by

$$\begin{array}{rcl} x{\otimes}y & = & \textsf{if}\;\; \#(x{\cdot}y) \leq k \;\rightarrow\; x{\cdot}y \\ & & \Box\;\; \#(x{\cdot}y) \geq k \;\rightarrow\; F_k(x{\cdot}y) \\ & & \textsf{fi}\quad . \end{array}$$

(Here $x{\cdot}y$ denotes ordinary concatenation of words x and y and $\#(x{\cdot}y)$ denotes its length.) Then $(T^{\leq k}, \otimes, \varepsilon)$ is a monoid. (The straightforward verification is left to the reader.)

The power set algebra has carrier the set of all languages whose words have at length at most k. Concatenation of languages involves truncating words down to length at most k.

□

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

[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.

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 33 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).

□

Our definition of a graph is similar to but subtley different from the standard definition of a labelled graph. The standard definition is a pair of sets, the *nodes* and the *edges*, and three functions from the edges to the nodes, $from$, to and $label$. By naming the sets of nodes, this definition allows multiple edges between any pair of nodes, whereas our definition, by not naming the set of nodes, does not. Using our definition, there are two distinct sets of nodes, the *from* nodes and the *to* nodes, the *from* nodes being given by the left domain of the edge relation, and the *to* nodes by its right domain.

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

Definition 34 (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

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

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

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

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 $r{\circ}s$

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

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

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

It is straightforward to check that $\mathbf{0}$ is a unit of addition and a zero of product, as the name suggests.

□

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

Theorem 35 (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 34. Finally, define the unit graph, denoted by $\mathbf{1}$, by

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

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

□

There are a number of parts to the proof of theorem 35, some of which are standard and are omitted here (for example, $\mathbf{0}$ is the zero of product, $\mathbf{1}$ is its unit, and addition and product are associative). Since graphs are functions and the supremum operator is defined pointwise we can use lemma 26 to assert that $(M_r A, \dot{\leq})$ forms a complete, universally distributive lattice. (The fact that the domain of a graph is a relation is not relevant to this part of the proof.) Graph product is not defined pointwise, however, so we need a separate lemma to establish the existence of an upper adjoint for graph product.

Lemma 36 The product operation on graphs admits left and right division operators. That is, there are operators $\backslash$ and $/$ such that, for all graphs f, g and h,

$$f \dot{\times} g \;\dot{\leq}\; h \equiv f \;\dot{\leq}\; h/g$$

and

$$f \dot{\times} g \;\dot{\leq}\; h \equiv g \;\dot{\leq}\; f \backslash h \quad .$$

Proof We have to construct the upper adjoint of $(f\dot{\times})$ for any graph f with range A. In the following calculation the range of the dummies i, j, and k is constrained by the requirements that the pairs (i,j), (j,k) and (i,k) are all in the relation r.

$$f \dot{\times} g \;\dot{\leq}\; h$$

$$\equiv \qquad \{ \qquad \text{definition of pointwise ordering} \qquad \}$$

$$\forall\langle i,k:: i\langle f\dot{\times}g\rangle k \leq i\langle h\rangle k\rangle$$

$\equiv$ { definition of composition }

$$\forall\langle i,k:: \Sigma\langle j:: (i\langle f\rangle j)\times(j\langle g\rangle k)\rangle \leq i\langle h\rangle k\rangle$$

$\equiv$ { supremum and nesting }

$$\forall\langle i,j,k:: (i\langle f\rangle j)\times(j\langle g\rangle k) \leq i\langle h\rangle k\rangle$$

$\equiv$ { $\mathcal{A}$ is a regular algebra.
Thus, $((i\langle f\rangle j)\times)$ has upper adjoint $(i\langle f\rangle j)\backslash$, say }

$$\forall\langle i,j,k:: j\langle g\rangle k \leq (i\langle f\rangle j)\backslash(i\langle h\rangle k)\rangle$$

$\equiv$ { infima }

$$\forall\langle j,k:: j\langle g\rangle k \leq \Pi\langle i:: (i\langle f\rangle j)\backslash(i\langle h\rangle k)\rangle\rangle$$

$\equiv$ { definition of pointwise ordering }

$$g \mathrel{\dot{\leq}} \langle j,k:: \Pi\langle i:: (i\langle f\rangle j)\backslash(i\langle h\rangle k)\rangle\rangle \quad .$$

Dually, it is possible to construct the upper adjoint of $(\dot{\times}g)$ for any graph g.
□

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 20. For further examples see [BC75, Tar81].

4 Measures

This section is about extending measures on the elements of an ordered monoid to measures on the power set regular algebra induced by that monoid. 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. We show how to extend the measure in such a way that it becomes a homomorphism of regular algebras. For example the length function on words is extended to the length-of-a-shortest-word function on languages.

Definition 37 (Monoidal Measures) 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 a *monoid homomorphism* from $\mathcal{R}$ to $\mathcal{S}$ if m preserves units:

$$m.1_R = 1_S$$

and preserves composition: for all x and y in M,

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

□

Definition 38 (Regular Homomorphism) 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.

□

Definition 37 imposes two requirements on the function m. Note, however, that the requirement that it preserve the unit of $\mathcal{R}$ 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.

Definition 39 (Compositional Measure) Let $\mathcal{R} = (R, \circ, 1_R)$ and $\mathcal{S} = (S, \cdot, 1_S)$ be monoids. Suppose m is a function with domain R and range S. Then m is said to be *compositional* if, for all x and y in R,

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

□

Theorem 40 (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 (see definition 39) 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 $m.\mathcal{R}$.

Proof Much of the proof of this theorem is covered by the unity-of-opposites theorem — the theorem tells us that $(m.R, \leq)$ is a complete lattice with binary supremum operator $\boxplus$ as defined above and least element 0_S. To show that $m.\mathcal{R}$ is a regular algebra it thus suffices to show that $m.\mathcal{R}$ admits left and right division operators.

Suppose $a \backslash b$ denotes right division in $\mathcal{S}$. Note that $m.x \backslash m.y$ is not necessarily in $m.R$. However, for all x, y and z,

$$m.y \;\leq\; m.x \backslash m.z \;\;\equiv\;\; m.y \leq m.(m^\sharp.(m.x \backslash m.z))$$

as

$$\begin{array}{ll}
 & m.y \leq m.x \backslash m.z \\
\Leftarrow & \{ \qquad \text{cancellation: } m.(m^\sharp.s) \leq s \quad \} \\
 & m.y \leq m.(m^\sharp.(m.x \backslash m.z)) \\
\Leftarrow & \{ \qquad \text{monotonicity of } m \quad \} \\
 & y \;\leq\; m^\sharp.(m.x \backslash m.z) \\
= & \{ \qquad \text{Galois connection} \quad \} \\
 & m.y \leq m.x \backslash m.z \quad .
\end{array}$$

Thus, in $m.R$ right division is given by the rule

$$m.x \cdot m.y \;\leq\; m.z \;\;\equiv\;\; m.y \leq m.(m^\sharp.(m.x \backslash m.z)) \quad .$$

Left division is defined similarly.

It only remains to observe that m preserves product by assumption and trivially maps the unit of $\mathcal{R}$ to the unit of $m.\mathcal{R}$. That is, m is a regular homomorphism.
□

Theorem 41 (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 29. 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 equivales m is a monoid homomorphism.

Proof There are two parts to the proof: showing that if $measure$ is a regular homomorphism then m is a monoid homomorphism and its converse. That m is a monoid homomorphism if $measure$ is a regular homomorphism follows from the fact that $measure$ is (by definition) a monoid homomorphism and instantiating the requirements on its being a monoid homomorphism in the case that the sets are singleton sets (i.e. have exactly one element).

For the converse, we have to show that the extended function $measure$ is monoidal and that it is a lower adjoint. It is easy to see that $measure.\{1_M\} = 1_R$, i.e. $measure$ preserves units. To see that it preserves products we calculate as follows:

$$
\begin{array}{ll}
 & measure.(S{\cdot}T) \\
= & \{ \quad \text{definition of } measure \quad \} \\
 & \Sigma\langle x{:}\; x \in S{\cdot}T{:}\; m.x\rangle \\
= & \{ \quad \text{definition of } S{\cdot}T \quad \} \\
 & \Sigma\langle u,v{:}\; u{\in}S \wedge v{\in}T{:}\; m.(u{\circ}v)\rangle \\
= & \{ \quad m \text{ is monoidal} \quad \} \\
 & \Sigma\langle u,v{:}\; u{\in}S \wedge v{\in}T{:}\; m.u \cdot m.v\rangle \\
= & \{ \quad \text{nesting} \quad \} \\
 & \Sigma\langle u{:}\, u{\in}S{:}\, \Sigma\langle v{:}\; v{\in}T{:}\; m.u \cdot m.v\rangle\rangle \\
= & \{ \quad \text{product in a regular algebra is} \\
 & \quad\quad \text{universally distributive} \quad \} \\
 & \Sigma\langle u{:}\; u{\in}S{:}\; m.u \cdot \Sigma\langle v{:}\, v{\in}T{:}\, m.v\rangle\rangle \\
= & \{ \quad \text{same again} \quad \} \\
 & \Sigma\langle u{:}\, u{\in}S{:}\, m.u\rangle \cdot \Sigma\langle v{:}\, v{\in}T{:}\, m.v\rangle \\
= & \{ \quad \text{definition of } measure \quad \} \\
 & measure.S \;\cdot\; measure.T \quad .
\end{array}
$$

Finally, it is easy to see that $measure$ is a lower adjoint:

$$
\begin{array}{ll}
 & measure.S \le k \\
= & \{ \quad \text{definition of } measure \quad \} \\
 & \Sigma\langle u{:}\, u{\in}S{:}\, m.u\rangle \le k \\
= & \{ \quad \text{suprema} \quad \} \\
 & \forall\langle u{:}\ u{\in}S{:}\ m.u \le k\rangle \\
= & \{ \quad \text{define } m^\sharp \in 2^M{\leftarrow}R \text{ by} \\
 & \qquad\quad u \in m^\sharp k \equiv m.u \le k \quad \} \\
 & \forall\langle u{:}\ u{\in}S{:}\ u \in m^\sharp k\rangle \\
= & \{ \quad \text{set comprehension} \quad \} \\
 & S \subseteq m^\sharp k \ \ .
\end{array}
$$

□

We consider several examples.

Example 42 (Test for Empty) Suppose we wish to determine whether a language is empty or not. Consider the regular algebra $\mathbb{B}$ (definition 19). 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

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

That is,

$$measure.S \ \equiv\ S \neq \phi \ \ .$$

The measue m is clearly monoidal and so $measure$ is a regular homomorphism.

□

Example 43 (Membership) We return to the membership problem discussed in section 2.3. Consider the regular algebra $\mathbb{B}$ (definition 19). 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

$$measure.S \ \equiv\ \exists\langle u{:}\, u{\in}S{:}\, u{=}X\rangle \ \ .$$

That is,

$$measure.S \ \equiv\ X{\in}S \ \ .$$

This measure is a regular homomorphism equivales measure m is a monoidal homomorphism. But

m is monoidal

$=\qquad \{ \quad \text{definition} \quad \}$

$\varepsilon = X \ \wedge \ \forall\langle u, v{::}\ u{\cdot}v = X \ \equiv \ u = X \wedge v = X\rangle$

$=\qquad \{ \quad \text{properties of strings} \quad \}$

$\varepsilon = X$.

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.

□

Example 44 (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 32.) $FIRST_k$ 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 32, it is clear that $F_k(x{\cdot}y) = F_k x \otimes F_k y$. That is, F_k is monoidal. Hence $FIRST_k$ is a regular homomorphism.

□

Theorem 45 (Vectorial Extensions) Let $\mathcal{R} = (R, \circ, \oplus, \preceq, 0_R, 1_R)$ and $\mathcal{S} = (S, \cdot, +, \leq, 0_S, 1_S)$ be regular algebras and suppose I is a set. Suppose $m \in R{\leftarrow}S$ is a regular homomorphism. Define the extension $measure$ of m to the vector algebras $\mathcal{R}{\leftarrow}I$ and $\mathcal{S}{\leftarrow}I$ by, for all $i{\in}I$ and $f \in S{\leftarrow}I$,

$$(measure.f).i \ = \ m.(f.i) \ .$$

Then $measure$ is a regular homomorphism.

Proof As with all pointwise definitions, the proof is a straightforward extension of the definitions.

□

The next two examples involve graph algebras. It is convenient to use the notation

$$\langle i,j{::}\ E\rangle$$

for a graph whose (i,j)th element is given by the expression E. (The range of the dummies i and j will be given by the context.) Note how example 46 generalises example 43.

Example 46 (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

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

This defines $m.u$ to be a boolean graph. 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 monoidal. This is proved as follows. Clearly $m.\varepsilon$ is the unit graph $\mathbf{1}$. Also,

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

$$\langle i,j{::}\ \exists\langle k{::}\ X[\,i\,..\,k\,) = u \wedge X[\,k\,..\,j\,) = v\rangle\rangle$$

$$= \qquad \{ \qquad \text{word calculus} \quad \}$$

$$\langle i,j{::}\ X[\,i\,..\,j\,) = u{\cdot}v\rangle$$

$$= \qquad \{ \qquad \text{definition of } m \quad \}$$

$$m.(u{\cdot}v) \quad .$$

□

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

Example 47 (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 46, 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 20. 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 is proved as follows.

$$
\begin{array}{ll}
 & m.(u{\cdot}v) \\
= & \quad \{ \quad \text{definition of } m \quad \} \\
 & \langle i,j{::}\ dist(X[i..j)\ ,u{\cdot}v)\rangle \\
= & \quad \{ \quad \bullet \quad \text{property of } dist \quad \} \\
 & \langle i,j{::}\ \downarrow\langle k{::}\ dist(X[i..k)\ ,u)+dist(X[k..j)\ ,v)\rangle\rangle \\
= & \quad \{ \quad \text{definition of graph product} \quad \} \\
 & \langle i,j{::}\ dist(X[i..j)\ ,u)\rangle \times \langle i,j{::}\ dist(X[i..j)\ ,v)\rangle \\
= & \quad \{ \quad \text{definition of } m \quad \} \\
 & m.u \times m.v \quad .
\end{array}
$$

Note that a crucial step in this calculation is the second step

$$
\begin{array}{ll}
 & \langle i,j{::}\ dist(X[i..j)\ ,u{\cdot}v)\rangle \quad . \\
= & \quad \{ \quad \bullet \quad \text{property of } dist \quad \} \\
 & \langle i,j{::}\ \downarrow\langle k{::}\ dist(X[i..k)\ ,u)+dist(X[k..j)\ ,v)\rangle\rangle
\end{array}
$$

This 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 monoidal 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 monoidal. Indeed, computing minimal edit distances for context-free languages is very difficult if the possibility of transpositions is included in the analysis.

Note that m is compositional but not monoidal. 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 40 is 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 $measure$.

□

5 Fusion for Languages

6 Programming Methodology

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.

[CC79] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages*, pages 269–282, San Antonio, Texas, January 1979.

[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.

[JS94] J. Jeuring and S.D. Swierstra. Bottom-up grammar analysis — a functional formulation —. In Donald Sannella, editor, *Proceedings Programming Languages and Systems, ESOP '94*, volume 788 of *LNCS*, pages 317–332, 1994.

[JS95] J. Jeuring and S.D. Swierstra. Constructing functional programs for grammar analysis problems. In S. Peyton Jones, editor, *Proceedings Functional Programming Languages and Computer Architecture, FPCA '95*, June 1995.

[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.

[LNS82] J.-L. Lassez, V.L. Nguyen, and E.A. Sonenburg. Fixed point theorems and semantics: a folk tale. *Information Processing Letters*, 14(3):112–116, 1982.

[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.