Factor Theory and the Unity of Opposites

Roland Backhouse

School of Computer Science, University of Nottingham, Nottingham NG8 1BB, England

Factor Theory and the Unity of Opposites

Roland Backhouse

School of Computer Science, University of Nottingham, Nottingham NG8 1BB, England

Abstract

The theory of factors of a regular language is used to illustrate the unity-of-opposites theorem of Galois connections. Left and right factors of a language are characterised as unions of right- and left-invariant equivalence classes, respectively, and this characterisation is exploited in the construction of the factor graph. The factor graph is a representation of the poset of left factors and, isomorphically by the unity of opposites, the poset of right factors. Two illustrative examples are given, one of which is the failure function used in the Knuth-Morris-Pratt pattern-matching algorithm.

Keywords: Galois connection, regular algebra, regular language, factor theory, factor matrix, factor graph, pattern matching

Quien sabe por algebra sabe scientificamente. [Who knows by algebra knows scientifically.] Petrus Nonius Salaciensis (Pedro Nunes Salaciense), Libro de Algebra, 1567

1. Introduction

In 1971, I chanced upon a thin book entitled “Regular Algebra and Finite Machines” by J.H. Conway [Con71]. I was attracted by the word “Algebra” in its title and bought it without hesitation because I was convinced that algebra is the key to turning the art of programming into a science. It subsequently formed the basis of much of my PhD thesis [Bac75].

Particularly inspiring for me was Conway’s theory of what he called “factors” of a regular language and its application to the construction of “biregulators” and the derivation of regularity-preserving operations on languages. Compared to some other research papers that I had been wading through, Conway’s book came as a breath of fresh air. A highlight was page 58 which contains a table of biregulators; in this table each line is an algebraic formula —in the form of a “biregulator”— representing a function mapping languages to languages. By applying the theorem that biregulators preserve regularity, he was thus able to prove that all the functions in the table map regular languages to regular languages. Here was evidence indeed of the power of algebra.

Disappointingly, shortly after completing my PhD the relevance of factor theory had not lived up to my expectations. My enthusiasm was reawakened when I spotted a connection between the "failure function" that is a vital element in the Knuth-Morris-Pratt pattern matching algorithm [KMP77] and the "factor graph" of a regular language, which notion I had introduced in my thesis. With the help of an MSc student, Rudi Lutz, its connection with the KMP algorithm and its generalisation to sets of patterns [Wei73, AC75] was formulated and published in [BL77]. However the euphoria at this discovery proved short-lived; I was unable to find further practical applications and —based on the number of citations of Conway's book at the time— nor could others. (Although the book has become more widely cited as its author's fame has grown[1], I still do not know of any publications other than my own or ones written by my colleagues that make use of and/or extend factor theory.)

It wasn't until the late 1980s that I began to look again at factor theory after learning about Galois connections, particularly in the context of relation algebra. I was inspired because I realised that Conway's "factors" could be formulated in terms of Galois connections and that many of his theorems were instances of more general properties of such connections. For my own benefit, I wrote a short technical note on the topic but have never attempted to publish it.

When asked to write an article to celebrate José Nuno Oliveira's 60th birthday, I knew immediately that I should write something about algebra and Galois connections: for the simple reason that I know that José loves both these topics. Since José already knows about basic factor theory, I looked again at my thesis to see whether there is anything "new" I could say about factors and Galois connections. And, indeed, there is! In my thesis, I characterised factors differently from Conway because this made it easier to formulate an algorithm to construct the factor graph of a language. On rereading the thesis, I realised that this alternative characterisation provides a non-trivial illustration of the "unity-of-opposites" theorem which Lambek and Scott [LS86] describe as "the most interesting consequence of a Galois correspondence". ("Unity of opposites" is the name I have given to the theorem; the name is not used by Lambek and Scott.) This then is what this paper is about.

To make the contents more accessible to other readers, the paper begins in section 2 with a short summary of the theory of Galois connections; this is followed in section 3 by a longer calculational presentation of Conway's factor theory that exploits their properties.

The introduction to Galois connections concludes with the unity-of-opposites theorem. Briefly, the theorem asserts an isomorphism between the partial orderings on the image sets of two Galois-connected functions. Its application is

[1] "Conway" is a common English name; "J.H.Conway" is the now-famous mathematician John Horton Conway.

the subject of section 4. A summary of how the existence of factor graphs is established for regular languages is given in section 4.1. This is followed by an algorithm for its construction in section 4.2. It is at this point that we give a concrete illustration of the unity of opposites. Section 4.3 provides a further illustration based on the failure function used in the Knuth-Morris-Pratt pattern matching algorithm.

Apart from illustrating the unity-of-opposites theorem, several elements of the paper are novel in the sense that they have never been published in a journal or conference paper before now. This includes the calculational presentation of Conway's factor theory in section 3, the characterisation of factors as unions of certain (well-known) equivalence classes in section 3.3, and the algorithm to compute factor graphs in section 4.2.

2. Galois Connections

2.1. Definition and Examples

A Galois connection involves two partially ordered sets[2] $(A, \leq)$ and $(B, \preceq)$ and two functions, $F \in A \leftarrow B$ and $G \in B \leftarrow A$. These four components together form a *Galois connection* iff for all $x \in B$ and $y \in A$ the following holds[3]

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

This compact definition of a Galois connection was introduced in [Sch53]. We refer to F as the *lower adjoint* and to G as the *upper adjoint*.

Since the context of the main contribution of this paper is language theory, it seems appropriate to use functions on languages as examples of Galois connections. Suppose T is a finite set. The elements of T are called *symbols* and T itself is called the *alphabet*. A *word* of *length* n, where n is a natural number, is a sequence of symbols of length n. The *empty word*, denoted by ε, is the word of length zero. *Concatenation* is the operation of forming a word of length $m+n$ from two words of lengths m and n by appending the latter after the former. For example, if $T = \{a,b\}$ then ab and bba are both words and their concatenation is the word $abbba$. We use T^* to denote the set of all words. A *language over alphabet* T is a subset of T^*.

Example 1. Below is a list of functions on languages each of which is a lower adjoint in a Galois connection. Each is followed by details of the Galois connection.

[2]Galois connections can be defined for preordered sets but, for our purposes, we restrict the definition to posets.

[3]An infix dot is used to denote function application. The symbol "$\equiv$" denotes equality of booleans (regrettably, often introduced as "if and only if"). We do not use the standard untyped equality symbol here in order to avoid confusion with continued equalities and inequalities, e.g. $m < n = p \leq q$, which are conventionally read conjunctionally. We do use the conventional equality symbol for booleans in calculations when the meaning is indeed conjunctional, i.e. $p = q = r$ means $p = q$ ***and*** $q = r$ (and hence also $p = r$ by transitivity of equality) whatever the type of p, q and r.

(a) The boolean-valued function $X \mapsto (w{\in}X)$ that determines whether a fixed word w is in the given language X.

For all words w, languages X and booleans b,

$$w{\in}X \Rightarrow b \quad \equiv \quad X \subseteq \textsf{if } b \to T^* \,\Box\, \neg b \to \neg\{w\} \textsf{ fi} \quad .$$

($\neg\{w\}$ denotes the set of all words over alphabet T, but excluding the word w.) The upper adjoint is thus the function mapping boolean b to[4]

$$\textsf{if } b \to T^* \,\Box\, \neg b \to \neg\{w\} \textsf{ fi} \quad ,$$

and the ordering on booleans is "only if".

(b) The boolean-valued function $X \mapsto (X \subseteq L)$ that determines whether the given language X is a subset of a fixed language L.

For all languages X and L, and all booleans b,

$$X \subseteq L \Leftarrow b \quad \equiv \quad X \subseteq \textsf{if } b \to L \,\Box\, \neg b \to T^* \textsf{ fi} \quad .$$

The upper adjoint is thus the function mapping boolean b to

$$\textsf{if } b \to L \,\Box\, \neg b \to T^* \textsf{ fi} \quad ,$$

and the ordering on booleans is "if" (i.e. " $\Leftarrow$ ").

(c) The function (from languages to numbers) that determines the length of a shortest word in a given language.

Denoting, the function by the prefix operator " # ", we have, for all languages X and all natural numbers k,

$$\#X \geq k \quad \equiv \quad X \subseteq T^{\geq k} \quad .$$

($T^{\geq k}$ denotes the set of all words over alphabet T that have length at least k.) The upper adjoint is thus the function mapping natural number k to

$$T^{\geq k}$$

and the ordering on numbers is the at-least ordering.

(d) The function *prefix* that extends the prefix function on words to languages; specifically, this is the function from languages to languages defined by, for all languages L, $\mathit{prefix}.L \;=\; \{u,v : uv{\in}L : u\}$ [5].

[4] We use the Guarded Command [Dij76] notation $\textsf{if } b{\to}S \,\Box\, c{\to}T \textsf{ fi}$ to denote conditional statements.

[5] We use the so-called *Eindhoven notation* [Dij76] for quantifiers. In particular, bound variables and their scope are always made explicit. For example, in the expression $\{u,v : uv{\in}L : u\}$ the bound variables are u and v, and their scope is delimited by the

For all languages L and M,

$$\mathit{prefix}.L \subseteq M \quad\equiv\quad L \subseteq \{y \mid \langle\forall u,v : y = uv : u{\in}M\rangle\} \quad .$$

The upper adjoint is the function mapping language M to

$$\{y \mid \langle\forall u,v : y = uv : u{\in}M\rangle\} \quad ,$$

and the ordering is the subset relation.

□

2.2. Commutativity Properties

Inverse functions are Galois connected: if f and g are inverse functions then, for all x and y of appropriate type, $f.x = y \equiv x = g.y$. (Equality is, of course, a pre-order.) A vital property of inverse functions is that they have "inverse" algebraic properties. The exponential function, for instance, has as its inverse the logarithmic function; moreover,

$$\mathsf{exp}(-x) \;=\; \frac{1}{\mathsf{exp}\,x} \qquad \text{and} \qquad \mathsf{exp}(x+y) \;=\; \mathsf{exp}\,x \cdot \mathsf{exp}\,y$$

whereas

$$-\mathsf{ln}\,x \;=\; \mathsf{ln}(\frac{1}{x}) \qquad \text{and} \qquad \mathsf{ln}\,x + \mathsf{ln}\,y \;=\; \mathsf{ln}(x \cdot y) \quad .$$

In general, if θ and ϕ are inverse functions then, for any functions f and g of appropriate type,

$$\langle\forall x :: \theta.(f.x) = g.(\theta.x)\rangle \;\equiv\; \langle\forall y :: f.(\phi.y) = \phi.(g.y)\rangle \quad .$$

More generally, and expressed at function level, if $(\theta_0\,,\,\phi_0)$ and $(\theta_1\,,\,\phi_1)$ are pairs of inverse functions, then for all functions f and g of appropriate type[6],

$$\theta_0{\bullet}f = g{\bullet}\theta_1 \;\equiv\; f{\bullet}\phi_1 = \phi_0{\bullet}g \quad . \tag{1}$$

Algebraic properties like this are the key to efficient computation and a goal of computing science is to try to predict such properties of unfamiliar functions.

accolades; the variable L is free. The subexpression $uv{\in}L$ gives the range of the bound variables and the rightmost occurrence of u is the term of the quantification. The expression $\langle\forall u,v : y = uv : u{\in}M\rangle$ denotes a universal quantification, with bound variables u and v whose scope is delimited by the angle brackets; the range is the subexpression $y = uv$ and the term is $u{\in}M$. Occasionally, if there is exactly one bound variable that is the term of a set-valued quantification, we use the more conventional notation exemplified by $\{y \mid \langle\forall u,v : y = uv : u{\in}M\rangle\}$: importantly, in this example the variable M is free and not bound.

[6]Composition of functions is denoted by a raised, bold infix dot. Specifically, $(\theta{\bullet}\phi).x = \theta.(\phi.x)$ for all functions θ and ϕ and all values x of appropriate type.

Many have the form of what we call "commutativity properties". Among them are, for example, the property $-(2{\cdot}x) = 2{\cdot}(-x)$ which expresses the fact that multiplication by 2 commutes with negation. We also include properties like $\mathsf{ln}\frac{1}{x} = -(\mathsf{ln}\,x)$ in which the order of application of the logarithmic function is "commuted" but in so doing a change occurs in the function with which it is commuted (in this case the reciprocal function becomes negation). In this way, distributivity properties also become commutativity properties. For instance the property that $x{\cdot}(y{+}z) = (x{\cdot}y){+}(x{\cdot}z)$ is a commutativity property of addition: multiplication by x after addition commutes to addition after the function $(y,z) \mapsto (x{\cdot}y\,,\,x{\cdot}z)$.

In general, for a given function F we are interested in discovering functions g and h for which $F\bullet g = h\bullet F$. In the case that F is defined by a Galois connection we can often answer this question most effectively by translating it into a question about the commutativity properties of its adjoint — especially in the case that the adjoint is a known function with known properties, or a "trivial" function whose algebraic properties are easily determined.

The rule that is the key to this strategy is the following. Suppose, for numbers m and n , $0 \le m$ and $0 \le n$, and for all i , $0 \le i < m{+}n$, (F_i, G_i) is a Galois-connected pair of functions. Then, assuming the functions are so typed that the compositions and equalities are meaningful,

$$F_0\bullet\ldots\bullet F_{m-1} = F_m\bullet\ldots\bullet F_{m+n-1} \;\equiv\; G_{m+n-1}\bullet\ldots\bullet G_m = G_{m-1}\bullet\ldots\bullet G_0 \quad . \tag{2}$$

(Note that the cases $m = 0$ and $n = 0$ are included; the composition of zero functions is of course the identity function.) In particular, if for $i = 0..1$, (h_i, k_i) is a Galois-connected pair of functions and so too is (F,G) then

$$h_0\bullet F = F\bullet h_1 \;\equiv\; k_1\bullet G = G\bullet k_0 \quad . \tag{3}$$

The rule (3) captures the strategy we use to discover algebraic properties of an unknown function F , namely to translate to the discovery of properties of the adjoint function.

2.3. Unity of Opposites

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

$$x :: \quad \langle \forall a :: x \sqsupseteq a \equiv \langle \forall b :: f.b \sqsupseteq a \rangle \rangle \quad . \tag{4}$$

Equation (4) need not have a solution. If it does, for a given f , we denote its solution by $\sqcap f$. By definition, then,

$$\langle \forall a :: \sqcap f \sqsupseteq a \equiv \langle \forall b :: f.b \sqsupseteq a \rangle \rangle \quad . \tag{5}$$

The poset $\mathcal{A}$ is $\mathcal{B}$*-complete* if there is a function $\sqcap$ that is the upper adjoint of the constant combinator K of type $(\mathcal{A}{\leftarrow}\mathcal{B}){\leftarrow}\mathcal{A}$ (defined by $\mathsf{K}.a.b = a$ for all a and b). That is, for all $a{\in}A$ and $f \in \mathcal{A}{\leftarrow}\mathcal{B}$,

$$\sqcap f \sqsupseteq a \;\equiv\; f \mathrel{\dot{\sqsupseteq}} \mathsf{K}.a \quad . \tag{6}$$

(The dot above the $\sqsupseteq$ relation on the right signifies the pointwise extension of that relation to functions. We frequently make use of pointwise extensions below, occasionally implicitly.) Dually, a *supremum* of f is a solution of the equation:

$$x :: \quad \langle\forall a :: x \sqsubseteq a \equiv \langle\forall b :: f.b \sqsubseteq a\rangle\rangle \quad . \tag{7}$$

As for infima, equation (7) need not have a solution. If it does, for a given f, we denote its solution by $\sqcup f$. By definition, then,

$$\langle\forall a :: \sqcup f \sqsubseteq a \equiv \langle\forall b :: f.b \sqsubseteq a\rangle\rangle \quad . \tag{8}$$

The poset $\mathcal{A}$ is $\mathcal{B}$*-cocomplete* if there is a function $\sqcup$ that is the lower adjoint of the constant combinator. That is, for all $a{\in}A$ and $f \in \mathcal{A}{\leftarrow}\mathcal{B}$,

$$\sqcup f \sqsubseteq a \;\equiv\; f \mathrel{\dot{\sqsubseteq}} \mathsf{K}.a \quad . \tag{9}$$

In the above definitions, $\mathcal{B}$ is called the *shape* poset. Examples of the shape poset are a 2-element set $\{0{,}1\}$ ordered by equality —used when defining binary suprema or infima— and the natural numbers ordered by the at-most relation —used in defining the notion of *continuity* of a function— .

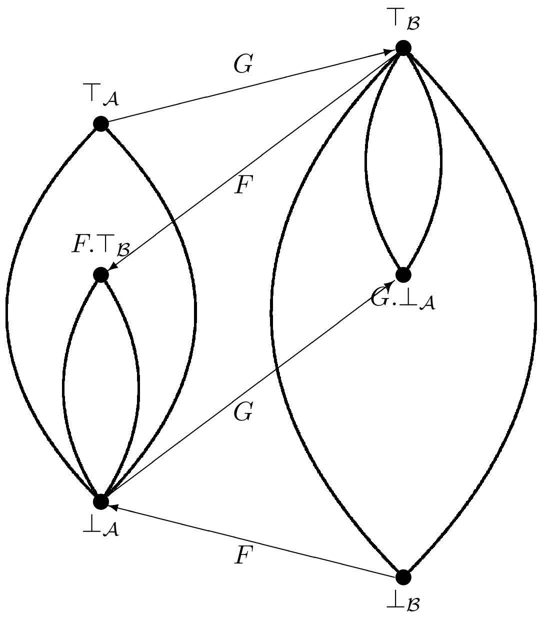
Theorem 1 (Unity of Opposites). Suppose $F \in \mathcal{A}{\leftarrow}\mathcal{B}$ and $G \in \mathcal{B}{\leftarrow}\mathcal{A}$ are Galois connected functions, F being the lower adjoint and G being the upper adjoint. Then $F.\mathcal{B}$ and $G.\mathcal{A}$ are isomorphic posets; in particular, $F{\bullet}G{\bullet}F = F$ and $G{\bullet}F{\bullet}G = G$. 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 and $\mathcal{C}$-cocomplete, the supremum and infimum operators are given by

$$\begin{array}{rcl}
\sqcap_{G.\mathcal{A}}.f & = & \sqcap_{\mathcal{B}}.f \\
\sqcup_{G.\mathcal{A}}.f & = & G.(F.(\sqcup_{\mathcal{B}}.f)) \\
\sqcap_{F.\mathcal{B}}.f & = & F.(\sqcap_{\mathcal{B}}.(G{\bullet}f)) \\
\sqcup_{F.\mathcal{B}}.f & = & F.(\sqcup_{\mathcal{B}}.(G{\bullet}f)) \quad .
\end{array}$$

□

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 $\bot_{\mathcal{B}}$ in the diagram) to the least element of $\mathcal{A}$ (denoted $\bot_{\mathcal{A}}$). Furthermore, G maps the greatest element of $\mathcal{A}$ (denoted $\top_{\mathcal{A}}$ in the diagram) to the greatest element of $\mathcal{B}$ (denoted $\top_{\mathcal{B}}$). 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.



2.4. Existence of Adjoints

We conclude this summary of the properties of Galois connections with a fundamental existence theorem.

Theorem 2 (Fundamental Theorem). Suppose that $(\mathcal{B},\preceq)$ is a poset and $(A,\sqsubseteq)$ is a complete poset. Then a monotonic function $G\in\mathcal{B}\leftarrow A$ is an upper adjoint in a Galois connection equivales G is inf-preserving[7].

Dually, if $(\mathcal{B},\preceq)$ is a co-complete poset and $(A,\sqsubseteq)$ is a poset, a monotonic function $F\in A\leftarrow\mathcal{B}$ is a lower adjoint in a Galois connection equivales F is sup-preserving.

□

[7]That is, for all monotonic functions f with range A, $G.(\sqcap_{(A,\sqsubseteq)}.f)$ is the infimum in $(\mathcal{B},\preceq)$ of $G\bullet f$.

3. Factors

Now that we have prepared the way, we turn to the application we have in mind. In this section we focus on the combination of the monoid structure of concatenation and the subset ordering of languages. We give a calculational proof of Conway's theorem that every regular language defines a so-called "factor matrix".

In more detail, section 3.1 introduces the notions of a "factor" and a "left" and "right" factor of a language via several Galois connections. In this way we can immediately identify several algebraic properties of factorisation (something that Conway did not do). A running example, which we first introduce in section 3.1, illustrates the benefits of the algebraic approach. Section 3.2 introduces Conway's factor matrix in a general algebraic setting, whilst section 3.3 specialises the results to *regular* languages. The calculational presentation of the factor matrix in section 3.2 is very different from Conway's presentation. In particular, the construction of appropriate factorisations is made explicit.

3.1. Product and Factors of Languages

Suppose T is an alphabet (a set of symbols). Then T^* is a set containing a distinguished element ε (the empty word) and which is closed under the binary concatenation operator. Moreover, concatenation is associative ($(uv)w = u(vw)$) and ε is both its left and right unit ($\varepsilon u = u = u\varepsilon$). The set T^* thus forms a monoid.

Recall that a *language* is a subset of T^* . We consider the poset $\mathcal{L}$ consisting of all languages ordered by set inclusion. This is a complete lattice. We extend the monoid structure of T^* to $\mathcal{L}$. We do this by first defining, for each word w and each language M , the *product* $w{\cdot}M$ by:

$$w{\cdot}M \;=\; \langle \cup x : x{\in}M : \{wx\} \rangle \quad .$$

We then define the *product*, $L{\cdot}M$, of two *languages* L and M by:

$$L{\cdot}M \;=\; \langle \cup w : w{\in}L : w{\cdot}M \rangle \quad .$$

For example, taking the alphabet once again to be $\{a,b\}$, the product of word ab and language $\{a,bb\}$ is the language $ab{\cdot}\{a,bb\} = \{aba,abbb\}$. The product of the two languages $\{a,bb\}$ and $\{b,cc\}$ is the language $\{ab,acc,bbb,bbcc\}$.

The use of the same symbol to denote the product of a word with a language and the product of a language with a language is justified by the identity

$$\{w\}{\cdot}M = w{\cdot}M \quad .$$

Indeed, a common convention is to make no distinction between the word w and the set $\{w\}$.

From these few simple definitions and our knowledge of Galois connections we can now enumerate a whole host of consequences. To begin, since the function ($w\cdot$) (which is an endofunction on $\mathcal{L}$) is defined to be universally distributive, we know from the fundamental theorem of Galois connections that it has an upper adjoint. Indeed,

$$
\begin{array}{ll}
 & w{\cdot}L \subseteq M \\
= & \quad \{ \quad \text{definition of concatenation} \quad \} \\
 & \langle \cup x : x{\in}L : \{wx\} \rangle \ \subseteq\ M \\
= & \quad \{ \quad \text{definition of supremum} \quad \} \\
 & \langle \forall x \ :\ x{\in}L \ :\ wx \in M \rangle \\
= & \quad \{ \quad \text{define the } w\text{-}\textit{derivative} \text{ of } M, \\
 & \qquad\quad \text{denoted } \partial_w M, \text{ by } x{\in}\partial_w M \equiv wx{\in}M \quad \} \\
 & \langle \forall x : x{\in}L : x \in \partial_w M \rangle \\
= & \quad \{ \quad \text{definition of infimum} \quad \} \\
 & L \subseteq \partial_w M \quad .
\end{array}
$$

The upper adjoint of ($w\cdot$) is the so-called *word derivative* ∂_w [Brz64] defined by

$$x{\in}\partial_w M \equiv wx{\in}M \quad .$$

The algebraic properties of word concatenation predict algebraic properties of derivatives. Since concatenation is associative it is straightforward to verify that

$$w{\cdot}(x{\cdot}L) = (wx){\cdot}L \quad .$$

Correspondingly —to be precise, by application of (2) with the instantiations $F_0 := (w\cdot)$, $F_1 := (x\cdot)$, $F_2 := (wx\cdot)$ and $m := 2$ — ,

$$\partial_x(\partial_w L) = \partial_{wx} L \quad .$$

(Note that the order of x and w switches.) Also, since ε is the left unit of concatenation, ($\varepsilon\cdot$) is equal to the identity function. It follows that its upper adjoint ∂_ε is also equal to the identity function.

Example 2. We assume that the reader is already familiar with derivatives of regular languages and their use to construct a reduced, deterministic finite-state automaton that recognises a given regular language. We expect, however, that few readers will be familiar with the theory of factors of a language. So let us take the opportunity to introduce a very simple example to which we will continually refer throughout this section and the next. The example we will take is the language $(a{+}b)^*\,a\,(a{+}b)^*$ over the alphabet $\{a,b\}$. Let us denote this language by E and the alphabet by T . Then, using equational properties of regular languages (which we assume the reader is familiar with), we have:

$$E \;=\; (a{+}b)^*\,a\,(a{+}b)^* \;=\; b^*\,a\,(a{+}b)^* \;=\; (a{+}b)^*\,a\,b^* \quad .$$

From these equations, it is easy to calculate that E has just two distinct derivatives: $\partial_\varepsilon E$ and $\partial_a E$. Specifically,

$$\partial_\varepsilon E \;=\; E \quad \text{and } \partial_a E \;=\; (a{+}b)^* \quad .$$

It is also easy to construct the reduced, deterministic finite-state automaton recognising E. It is shown in fig. 1(a). For brevity, we follow Conway and call it the *machine* of E.

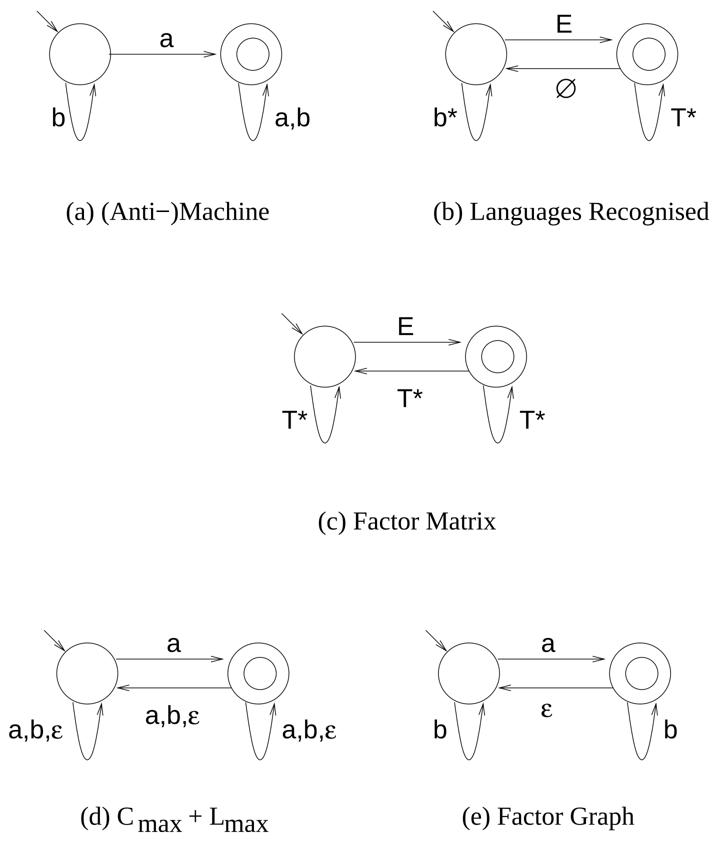


Figure 1: Recognisers of $(a+b)^*\,a\,(a+b)^*$

For greater simplicity, the example language we have chosen is such that it is the reverse of itself. (The reverse of a language is the set consisting of the reverse of all words in the language.) This means that the “anti-machine” of E is identical to its machine, as indicated by the caption of fig. 1(a). The “anti-machine” of a language is the machine of the reverse of the language.

We assume that the reader is familiar with the construction of the machine. So, for example, $\partial_b E = E$, from which equation we construct the edge labelled b from the start node to itself in fig. 1(a). We also assume that the reader is familiar with how the machine is used to recognise words in the language. Fig. 1(b) shows the languages recognised by state transitions; for example, the set of all state transitions from the start state (indicated in the usual way by an unlabelled incoming arrow) to the final state (also indicated in the usual way by two concentric circles) is the language E, and the set of all state transitions from the final state to the empty state is $\emptyset$ (the empty set).

In the following sections, we explain figs. 1(c), (d) and (e). We have grouped them all together so that we can draw parallels at the appropriate time. For the moment, note that fig. 1(e) is also a recogniser of E but is non-deterministic. The edge labelled ε from the final state to the start state of fig. 1(e) can be regarded as a "failure" transition: when in the final state of fig. 1(e), if the next symbol fails to match the symbol b (the only symbol for which there is a transition from the final state), this edge is followed; subsequently this process of choosing a transition is repeated. Fig. 1(c) shows the languages recognised by state transitions in fig. 1(e). Similarly, fig. 1(d) is also a non-deterministic recogniser of E. Comparing figs. 1(d) and 1(e), the set of labels on each edge of fig. 1(d) is a superset of the labels on the corresponding edge of fig. 1(e) but the languages recognised are the same.

The motivation for these observations and the definition and construction of each of these figures will become clearer as we proceed.
□

So far, thus, we assume that there are no surprises for the reader. But suppose we take the process one step further. We observed that the function $(w\cdot)$ has an upper adjoint. By a similar argument, the function $(L\cdot)$ has an upper adjoint. We have, for all languages L, M, and N,

$$
\begin{array}{ll}
 & L{\cdot}M \subseteq N \\
= & \{ \quad \text{definition of concatenation} \quad \} \\
 & \langle \cup w : w{\in}L : w{\cdot}M \rangle \ \subseteq\ N \\
= & \{ \quad \text{definition of supremum} \quad \} \\
 & \langle \forall w : w{\in}L : w{\cdot}M \subseteq N \rangle \\
= & \{ \quad \partial_w \text{ is the upper adjoint of } (w\cdot) \quad \} \\
 & \langle \forall w : w{\in}L : M \subseteq \partial_w N \rangle \\
= & \{ \quad \text{definition of infimum} \quad \} \\
 & M \ \subseteq\ \langle \cap w : w{\in}L : \partial_w N \rangle \quad .
\end{array}
$$

We denote the language $\langle \cap w : w{\in}L : \partial_w N \rangle$ by $L\backslash N$. It is called a *right factor* of the language N. (The notion and terminology, although not the notation, were introduced by J.H. Conway. The terminology is intended to reinforce an analogy with real arithmetic, whereby concatenation is linked to multiplication and factoring to division.) In summary we have:

$$L{\cdot}M \subseteq N \ \equiv\ M \subseteq L\backslash N \quad . \tag{10}$$

As for $(w\cdot)$ we may now conclude that the function $(L\cdot)$ preserves unions and $(L\backslash)$ preserves intersections. In particular, $(L\backslash)$ is monotonic and

$$L\backslash(M{\cap}N) \ = \ (L\backslash M) \cap (L\backslash N) \quad .$$

The next step is to mirror all the above calculations. Instead of defining the function $(\,w\cdot\,)$ we could equally well have defined the function $(\,\cdot w\,)$,

$$L\cdot w \;=\; \langle \cup x : x{\in}L : xw \rangle \quad ,$$

and extended its definition to the definition of a function $(\,\cdot M\,)$ for each language M. Noting that $L\cdot w$ as just defined is equal to $L\cdot\{w\}$ as defined earlier — this requires a short calculation— there is no ambiguity in denoting application of the function $(\,\cdot M\,)$ to L by $L\cdot M$. By a completely dual argument to that above, the function $(\,\cdot w\,)$ has an upper adjoint (which we won't bother supplying a notation for) and the function $(\,\cdot M\,)$ also has an upper adjoint, which we will denote by the postfix operator $(\,/M\,)$. That is,

$$L\cdot M \subseteq N \;\;\equiv\;\; L \subseteq N/M \quad . \tag{11}$$

The function $(\,/M\,)$ preserves intersections. The language N/M is a *left factor* of N.

More interesting is if we combine the Galois connections (10) and (11). By transitivity of equivalence we have:

$$M \subseteq L\backslash N \;\;\equiv\;\; L \subseteq N/M \quad .$$

Rewriting in the form:

$$N/M \supseteq L \;\;\equiv\;\; M \subseteq L\backslash N \quad ,$$

we recognise a Galois connection (actually a family of Galois connections indexed by the variable N). The lower adjoint is the function $(\,N/\,)$ to the poset of languages ordered by set containment $(\,\supseteq\,)$ from the poset of languages ordered by set inclusion $(\,\subseteq\,)$. The upper adjoint is the function $(\,\backslash N\,)$ to the poset of languages ordered by set inclusion $(\,\subseteq\,)$ from the poset of languages ordered by set containment $(\,\supseteq\,)$. By the fundamental theorem it thus follows that $(\,N/\,)$ maps a *union* of languages into an *intersection* of languages. The same is true of the function $(\,\backslash N\,)$. In particular, we have:

$$N/(L{\cup}M) \;=\; (N/L)\cap(N/M) \tag{12}$$

and

$$(L{\cup}M)\backslash N \;=\; (L\backslash N)\cap(M\backslash N) \quad . \tag{13}$$

What other properties do we get for free? We remarked earlier that the set of all words T^* forms a monoid under the concatenation of words, the unit of the monoid being the empty word ε. It is not difficult to prove that as a consequence the set of languages over T^* forms a monoid under the product operation on languages as defined above, and with $\{\varepsilon\}$ as unit. In other words, for all languages L, M, and N,

$$(L\cdot M)\cdot N \;=\; L\cdot(M\cdot N) \tag{14}$$

and

$$L\cdot\{\varepsilon\} \;=\; L \;=\; \{\varepsilon\}\cdot L \quad . \tag{15}$$

(These properties are not for free. They can easily be verified from the definition of product and known properties of set union.) Now, property (14) captures in one equation three identities between functions on languages, and each of these identities can be translated into a property of factors. First, by abstracting on the variable N, (14) asserts the equality of the two functions $((L\cdot M)\cdot)$ and $(L\cdot)\bullet(M\cdot)$. By applying (2) with the instantiations $F_0 := ((L\cdot M)\cdot)$, $F_1 := (L\cdot)$, $F_2 := (M\cdot)$ and $G_0 := ((L\cdot M)\backslash)$, $G_1 := (L\backslash)$, $G_2 := (M\backslash)$, we infer the property:

$$(L\cdot M)\backslash \;=\; (M\backslash)\bullet(L\backslash) \quad .$$

Re-introducing the variable N, this is equivalent to:

$$(L\cdot M)\backslash N \;=\; M\backslash(L\backslash N) \quad , \tag{16}$$

for all N. Second, by abstracting on the variable M, (14) asserts the equality of the two functions $(\cdot N)\bullet(L\cdot)$ and $(L\cdot)\bullet(\cdot N)$. Thus, again applying (2), we infer the property:

$$(/N)\bullet(L\backslash) \;=\; (L\backslash)\bullet(/N) \quad .$$

Re-introducing the variable M, this is equivalent to:

$$(L\backslash M)/N \;=\; L\backslash(M/N) \quad , \tag{17}$$

for all M. Finally, by abstracting on the variable L, applying (2) and reintroducing the variable L, we infer the property:

$$(L/N)/M \;=\; L/(M\cdot N) \quad . \tag{18}$$

Here is evidence indeed of the effectiveness of recognising Galois connections. We can very quickly deduce several properties of the operators just by knowing that they are connected to product and the properties of product (which are well known).

Example 3. The example language introduced earlier (example 2) is too simple to illustrate all the properties we have listed but we can use it to illustrate almost all. The language T^*aT^* where $T = \{a,b\}$, has just two distinct factors: T^*aT^* itself and T^*. For example,

$$T^*aT^* \;=\; \varepsilon\backslash(T^*aT^*) \;=\; (T^*aT^*)/\varepsilon \;=\; b\backslash(T^*aT^*) \;=\; (T^*aT^*)/b \quad .$$

The leftmost of these equalities[8] follows from the equation, for all M, $\varepsilon\cdot M = M$ and hence

$$\varepsilon\cdot M \;\subseteq\; T^*aT^* \;\equiv\; M \;\subseteq\; T^*aT^*$$

[8]The reader familiar with the algebra of derivatives of regular expressions may use the identity $w\backslash E = \partial_w E$, for all E and all words w, to verify this equality and some others that follow. Symmetrically, anti-derivatives are used to calculate E/w for regular expressions E and words w.

and the rightmost from the equation, for all L,

$$L{\cdot}b \subseteq T^*aT^* \;\equiv\; L \subseteq T^*aT^* \quad .$$

(Compare these equations with the definitions of right and left factors, (10) and (11), respectively.) Symmetry justifies the other equalities. We also have

$$T^* \;=\; a\backslash(T^*aT^*) \;=\; (T^*aT^*)/a$$

since, for all M, $M \subseteq T^*$ and $a{\cdot}M \subseteq T^*aT^*$; hence

$$a{\cdot}M \subseteq T^*aT^* \;\equiv\; M \subseteq T^* \quad .$$

The equation

$$T^* \;=\; \emptyset\backslash(T^*aT^*) \;=\; (T^*aT^*)/\emptyset$$

is derived by similar reasoning. Other factorisations can be deduced from the properties listed above. For example, we use (13) to deduce that

$$\{a,b\}\backslash(T^*aT^*) \;=\; a\backslash(T^*aT^*)\cap b\backslash(T^*aT^*) \;=\; T^*aT^*$$

and (18) and induction to deduce that, for all k,

$$(T^*aT^*)\,/\,b^k \;=\; T^*aT^* \quad .$$

Using (12) we conclude that

$$(T^*aT^*)\,/\,b^* \;=\; T^*aT^* \quad .$$

The reader is invited to check instances of (17) as well as to check the general formula: for all L and N,

$$L\backslash(T^*aT^*)/N \;=\; T^* \;\Leftarrow\; L{\cup}N \subseteq T^*aT^* \quad ,$$

and otherwise

$$L\backslash(T^*aT^*)/N \;=\; T^*aT^* \quad .$$

Note that Conway did not introduce the factor operators. This meant that he also did not identify their algebraic properties, and so left the construction of factors to a piecemeal, ad hoc process.
□

3.2. The Factor Matrix

In this section, we introduce Conway's factor matrix. As it turns out, very few of the properties of languages are exploited in the construction of the factor matrix. All that is needed is a partially ordered monoid that admits factorisation. To make this clear, we assume in this section that $(A,\cdot,1)$ is a monoid, $(A,\subseteq)$ is a complete lattice, and A *admits factorisation*: that is, there are operators $\backslash$ and $/$ such that, for all X, Y and Z in A

$$X{\cdot}Y \subseteq Z \equiv X \subseteq Z/Y \quad\text{ and }\quad X{\cdot}Y \subseteq Z \equiv Y \subseteq X\backslash Z \quad .$$

We call such a structure a *regular algebra*. Another name, according to Wikipedia, is a *unital quantale*. (In the literature on quantale theory —which I have only skimmed briefly— a factor is called a *residual*.)

In a regular algebra, the “Kleene star” operator can be defined in several ways. For us, the most convenient definition of X^* is the least fixed-point of the function mapping Y to $1 \cup X \cup Y \cdot Y$, where “$\cup$” denotes the binary supremum operator. In particular, for all X,

$$X = X^* \;\equiv\; X \supseteq 1 \cup X \cdot X \quad . \tag{19}$$

In words, borrowing terminology from relation algebra, X is its own star if it is “reflexive” (i.e. $X \supseteq 1$) and “transitive” (i.e. $X \supseteq X \cdot X$). Equivalent definitions are: X^* is the supremum of finite powers of X, X^* is the least fixed-point of the function mapping Y to $1 \cup X \cdot Y$, and X^* is the least fixed-point of the function mapping Y to $1 \cup Y \cdot X$.

It may surprise some readers that our definition of a regular algebra does not include the so-called “Kleene star” operator as a primitive. Several authors have studied axiom systems for regular languages in which the star operator is indeed primitive: Salomaa [Sal66] may well have been the first to do so; Kozen [Koz94] includes an extensive bibliography. But such axiom systems are typically designed for the sole task of being complete with respect to establishing the equality of regular expressions and, consequently, are inadequate for our purposes here. Conway studies several different “Kleene” algebras [Con71, chapter 4, pp34–40] all of which are derivatives of what he calls a *standard Kleene algebra* or **S**-algebra [Con71, p27]. Such an algebra has just two primitive operators, product and supremum, and, whilst lacking our emphasis on admitting factorisation via explicit naming of the factor operators, is identical to our “regular algebra”.

Since Conway’s factor theory seems to be almost unknown, it may help for us to provide a short summary. Conway does not define factors via a Galois connection as we have done; in particular, as we remarked earlier, he does not introduce the binary operators $/$ and $\backslash$ (although a definition of the ternary operator mapping X, Y and Z to $X \backslash Y / Z$ is implicit in his proof of theorem 1 in chapter 6 on Factors and the Factor Matrix). Instead he assumes a fixed “event” E, he defines a “subfactorization” of E and then defines a factor as an event that is maximal in a subfactorization. (“Events” are elements of an **S**-algebra.) He defines “left factors” and “right factors” and observes a one-to-one correspondence between left and right factors of E. He then assumes that the correspondence is encoded via an indexing of the left and right factors: he names the left factors L_i and the right factors R_i where i ranges over some anonymous index set. He then defines factor E_{ij} effectively to be what we have denoted by $L_i \backslash E / R_j$. He then asserts the existence of a “factor matrix” as follows [Con71, Theorem 4, p.48]:

> Each E_{ij} is a factor, and each factor is one of the E_{ij}. There exist unique indices l, r such that $E = L_r = R_l = E_{lr}$ and $L_i = E_{li}$

and $R_i = E_{ir}$ for each i. Hence the factors naturally form a square matrix among the entries of which is E.

Note that at this stage in the development nothing is stated or assumed about the index set; in particular, it is not assumed that it is finite. That is, Conway's use of the word "square matrix" does not comply with the standard notion where the elements of the matrix are indexed by a finite set. Conway subsequently proves the theorem that an event over a finite input alphabet is a regular language if and only if it has a finitely many factors. So, for regular events, the "factor matrix" is indeed a square matrix in the conventional sense. Several theorems, however, do not rely on the assumption that E is a regular language.

Example 4. Fig. 1(c) displays the factor matrix of $(a{+}b)^*\,a\,(a{+}b)^*$ in the form of a graph. That is, if we use the set of nodes as index set, the ij th factor is the label of the edge from node i to node j. Left factors are the labels of the edges from the start node (i.e. l is the index assigned to the start node) and right factors are the labels of the edges to the final node (i.e. r is the index assigned to the final node).
□

To properly relate our work to Conway's, it will be convenient for us to use the word "matrix" to describe any function that has domain a cartesian product $I{\times}J$ for some sets I and J and range a regular algebra, irrespective of whether or not the sets are finite. A "square matrix" is a function with domain $I{\times}I$ for some (possibly infinite) I. The product of two "matrices" $\mathbf{f}$ and $\mathbf{g}$ with domains $I{\times}J$ and $J{\times}K$, respectively, and the same range (a regular algebra) is the function with domain $I{\times}K$ defined by

$$(\mathbf{f}{\times}\mathbf{g})(i,k) \quad = \quad \langle \cup j :: \mathbf{f}(i,j){\cdot}\mathbf{g}(j,k)\rangle \quad .$$

Matrices with the same domain and range are ordered pointwise: if $\mathbf{f}$ and $\mathbf{h}$ both have domain $I{\times}J$,

$$\mathbf{f} \mathrel{\dot{\subseteq}} \mathbf{h} \;\equiv\; \langle \forall\, i,j :: \mathbf{f}(i,j) \subseteq \mathbf{h}(i,j)\rangle \quad .$$

Extending the supremum and infimum operators pointwise as well, the square matrices with domain $I{\times}I$, for some I, and range a regular algebra themselves form a regular algebra [Bac06, theorem 4.20][9]; the product is matrix product and the unit is the identity matrix, which we denote by $\mathbf{I}$.

An example of "matrix" algebras is afforded by binary relations. The booleans form a regular algebra with conjunction as the product operator and implication as the ordering. "Matrices" of booleans are binary relations and the product operator is relational composition.

[9]The proof is straightforward; Conway [Con71, p.40] states that it is trivial. (Strictly, he only makes this claim for finite-dimensional matrices; however, the finiteness assumption is only relevant for non-standard Kleene algebras.)

Let E denote a fixed element of A. (We use "E" as did Conway to help the reader to relate the properties stated here to those in Conway's book.) A *factor* of E is any element of A that can be expressed in the form $X\backslash E/Y$ for some X and Y. (That parentheses have been omitted here is permitted by virtue of (17).) An element of A is a *left factor* of E if it can be expressed in the form E/Y for some Y and a *right factor* of E if it can be expressed in the form $X\backslash E$ for some X. The function mapping X and Y to $X\backslash E/Y$, where X and Y range over all elements of A, thus forms a "matrix" of factors with index set A but this matrix is *not* Conway's factor matrix. Conway's factor matrix is a matrix indexed by the *left factors* (or equally the right factors) of E, this index set being finite in the case that E is a regular language (as opposed to A which is infinite). In more detail, his theorem states that the factors of E organise themselves into a reflexive, transitive matrix indexed by the left (or right) factors of E. Moreover, E itself and all left and right factors of E are elements of the matrix. Below we give a 4-step proof of the theorem.

Step 1. According to our definition of a matrix, the binary operators $\backslash$ and $/$, with domains restricted to $\mathcal{I}\times\mathcal{I}$ for arbitrary set $\mathcal{I}$, $\mathcal{I}\subseteq A$, are both square matrices over A. Conway's first observation is that the factor matrix is its own star. Indeed, this is true of any square $\backslash$ or $/$ matrix, as we can easily show:

Recall that an event is its own star if it is reflexive and transitive. (See equation (19).) Reflexivity of a $\backslash$ matrix with domain $\mathcal{I}\times\mathcal{I}$ is the property $1\subseteq L\backslash L$ for all $L\in\mathcal{I}$. By the Galois connection between factors and product, this is equivalent to $L\cdot 1\subseteq L$ which follows from the requirement that 1 is a right unit of product. The transitivity of the matrix follows from the following calculation:

$$L\backslash M\cdot M\backslash N\subseteq L\backslash N$$

$=$ { Galois connection defining factors }

$$L\cdot L\backslash M\cdot M\backslash N\subseteq N$$

$\Leftarrow$ { cancellation: $L\cdot L\backslash M\subseteq M$
monotonicity of product and transitivity of $\subseteq$ }

$$M\cdot M\backslash N\subseteq N$$

$=$ { cancellation }

true .

Dually a $/$ matrix is reflexive and transitive and hence its own star.

The clue we obtain from this step to the construction of the factor matrix is that it suffices to construct a suitable index set for the matrix $\backslash$ (or for the matrix $/$).

Step 2. Define the functions $\lhd$ and $\rhd$ by

$$X\lhd \;=\; E/X \quad , \qquad (20)$$

$$X\rhd \;=\; X\backslash E \quad . \qquad (21)$$

By definition, the range of $\lhd$ is the set of *left* factors of E and the range of $\rhd$ is the set of *right* factors of E. It is an easy calculation to derive the Galois connection: for all X and Y,

$$X \subseteq Y\lhd \equiv Y \subseteq X\rhd \ .$$

Note that because of the reversal of the ordering, both operators $\lhd$ and $\rhd$ are anti-monotonic. That is, $X \subseteq Y \Rightarrow X\lhd \supseteq Y\lhd \wedge X\rhd \supseteq Y\rhd$.

From the Galois connection, we deduce that, for all X,

$$X\lhd\rhd\lhd \;=\; X\lhd \ , \tag{22}$$

$$X\rhd\lhd\rhd \;=\; X\rhd \ . \tag{23}$$

Moreover,

$$E\lhd\rhd \;=\; E \;=\; E\rhd\lhd \ , \tag{24}$$

$$X\lhd \backslash Y\lhd \;=\; X\lhd\rhd \,/\, Y\lhd\rhd \ , \tag{25}$$

$$X\rhd / Y\rhd \;=\; X\rhd\lhd \,\backslash\, Y\rhd\lhd \ . \tag{26}$$

Properties (22) and (23) are instances of the unity of opposites, theorem 1. The rightmost equality in (24) is established as follows. The proof of the leftmost equality is similar.

$$
\begin{array}{ll}
 & E\rhd\lhd = E \\
= & \{ \quad \text{antisymmetry} \quad \} \\
 & E\rhd\lhd \subseteq E \ \wedge \ E\rhd\lhd \supseteq E \\
= & \{ \quad X \subseteq Y\lhd \equiv Y \subseteq X\rhd \text{ with } X{,}Y := E\,, E\rhd\,, \\
 & \quad\ \ \text{reflexivity of } \geq \quad \} \\
 & E\rhd\lhd \subseteq E \\
= & \{ \quad \text{definition} \quad \} \\
 & E/(E\backslash E) \subseteq E \\
\Leftarrow & \{ \quad E/ \text{ is an anti-monotonic function,} \quad E/1 = E \quad \} \\
 & E\backslash E \supseteq 1 \\
= & \{ \quad \text{factors, unit} \quad \} \\
 & \mathsf{true} \ .
\end{array}
$$

To establish (25) and (26), and for later use, it is convenient to observe that (17), appropriately instantiated, gives the identity

$$X\backslash(Z\lhd) \;=\; (X\rhd)/Z \ . \tag{27}$$

The calculation of (25) is now easy:

$$\begin{array}{ll} & X{\lhd}\backslash Y{\lhd} \\ = & \{ \quad (22) \quad \} \\ & X{\lhd} \backslash Y{\lhd}{\rhd}{\lhd} \\ = & \{ \quad (27) \quad \} \\ & X{\lhd}{\rhd} \,/\, Y{\lhd}{\rhd} \ . \end{array}$$

Property (26) is calculated in the same way.

This step records a (1-1) correspondence between left and right factors of E (properties (22) and (23)). Also, any matrix indexed by left factors can be mapped directly into a matrix indexed by right factors, and vice-versa (properties (25) and (26)). Moreover —property (24)— E is both a left and right factor of itself.

Example 5. For our running example, where $E = T^*aT^*$, the left and right factors have already been calculated in example 3. Specifically, T^* and T^*aT^* are both left factors and also both right factors. We also have

$$\begin{array}{rcl} (T^*aT^*){\lhd} & = & T^* \ , \\ (T^*){\lhd} & = & T^*aT^* \ , \\ (T^*aT^*){\rhd} & = & T^* \ , \\ (T^*){\rhd} & = & T^*aT^* \ . \end{array}$$

The left factors are totally ordered: $T^*aT^* \subseteq T^*$. Likewise, the right factors are totally ordered: $(T^*aT^*){\rhd} \supseteq (T^*){\rhd}$. Vice-versa, the right factors are totally ordered: $T^*aT^* \subseteq T^*$, and so are the left factors: $(T^*aT^*){\lhd} \supseteq (T^*){\lhd}$. The equations (22) and (23) are easily verified.
□

Let $\mathcal{L}$ denote the set of left factors of E. The conclusion from steps 1 and 2 is that there are only two reasonable candidates for Conway's factor matrix, the matrix $\backslash$ indexed by $\mathcal{L}$ and the matrix $/$ also indexed by $\mathcal{L}$. After a moment's thought it is obvious that the latter matrix is uninteresting, so we consider the former.

Step 3. Define the *factor matrix* of E to be the binary operator $\backslash$ restricted to $\mathcal{L}{\times}\mathcal{L}$. Thus entries in the matrix take the form $L_0\backslash L_1$ where L_0 and L_1 are left factors of E. By step 1, this is a reflexive, transitive matrix. Also, by definition of a left factor and a factor, all entries in the matrix are factors of E.

Suppose that F is a factor, L is a left factor, and R is a right factor of E. We now construct left factors L_0, L_1, L_2, L_3, L_4, L_5 such that

$$F = L_0\backslash L_1 \ , \qquad (28)$$
$$L = L_2\backslash L_3 \ , \qquad (29)$$
$$R = L_4\backslash L_5 \ . \qquad (30)$$

Moreover, L_2 is independent of L and L_5 is independent of R and

$$E \;=\; L_2\backslash L_5 \quad . \tag{31}$$

Suppose F is a factor of E. In particular, suppose that $F = U\backslash E/V$. We construct X and Y such that $F = X\lhd\backslash Y\lhd$ as follows.

$$\begin{array}{ll}
 & X\lhd\backslash Y\lhd = U\backslash E/V \\
= & \qquad \{ \qquad X\backslash Y\lhd = X\rhd/Y \text{ with } X,Y := X\lhd, Y \;,\; U\backslash E = U\rhd \qquad \} \\
 & X\lhd\!\rhd/Y \;=\; U\rhd/V \\
\Leftarrow & \qquad \{ \qquad \textit{Postulate } Y = V \qquad \} \\
 & X\lhd\!\rhd = U\rhd \\
\Leftarrow & \qquad \{ \qquad U\rhd\lhd\!\rhd \;=\; U\rhd \qquad \} \\
 & X = U\rhd \quad .
\end{array}$$

Thus $U\backslash E/V = U\rhd\lhd\backslash V\lhd$.

Now consider the left factor $V\lhd$. This is written in the form $X\lhd\backslash Y\lhd$ as follows.

$$\begin{array}{ll}
 & V\lhd \\
= & \qquad \{ \qquad \text{definition} \qquad \} \\
 & E/V \\
= & \qquad \{ \qquad (24) \qquad \} \\
 & E\lhd\!\rhd/V \\
= & \qquad \{ \qquad X\backslash Y\lhd = X\rhd/Y \text{ with } X,Y := E\lhd, V \qquad \} \\
 & E\lhd\backslash V\lhd \quad .
\end{array}$$

Thus $V\lhd \;=\; E\lhd\backslash V\lhd$. Finally, consider the right factor $U\rhd$. We write this in the form $X\lhd\backslash Y\lhd$ as follows:

$$\begin{array}{ll}
 & U\rhd \\
= & \qquad \{ \qquad \text{definition} \qquad \} \\
 & U\backslash E \\
= & \qquad \{ \qquad (24) \qquad \} \\
 & U\backslash E\rhd\lhd \\
= & \qquad \{ \qquad \text{(27) with } X,Z := U\,, E\rhd \qquad \} \\
 & U\rhd/E\rhd \\
= & \qquad \{ \qquad (23) \qquad \} \\
 & U\rhd\lhd\!\rhd/E\rhd
\end{array}$$

$=$ { (27) with $X,Z := U\triangleright\triangleleft , E\triangleright$ }

$U\triangleright\triangleleft \setminus E\triangleright\triangleleft$.

Thus, $U\triangleright = (U\triangleright)\triangleleft \setminus (E\triangleright)\triangleleft$.

The terms L_2 and L_5 in (29) and (30) are thus $E\triangleleft$ and E . The verification of (31) is then:

$E\triangleleft \setminus E$

$=$ { definition }

$E\triangleleft\triangleright$

$=$ { (24) }

E .

By these explicit constructions, we have established Conway's theorem 4 [Con71, p.48]. Let us do some renaming in order to make it easier to compare our formulation of the theorem with Conway's. As always, E denotes a fixed "event" (thus not necessarily a regular language). Consider the "matrix" defined by the binary operator $\setminus$ indexed by left factors of E . So, each "entry" in the matrix has the form $i\setminus j$ for some left factors i and j of E . Conway uses the notation E_{ij} for such an entry. Then each entry is a factor of E since j is a left factor of E equivales $j = j\triangleright\triangleleft = E \,/\, j\triangleright$ and, hence, $i\setminus j = i \setminus E \,/\, j\triangleright$. Moreover, we have shown that each factor of E is an entry in the matrix, specifically:

$$U\setminus E/V \;=\; U\triangleright\triangleleft \setminus V\triangleleft \quad . \tag{32}$$

In addition, let l denote the left factor $E\triangleleft$ and r denote E (which is a left factor of E since $E = E\triangleright\triangleleft$). Then

$$E = r = l\triangleright = l\setminus r \quad . \tag{33}$$

In words, E is the left factor r , the right factor corresponding to l , and the (l,r) th entry in the matrix. Also, for all left factors i of E

$$i = l\setminus i \tag{34}$$

and

$$i\triangleright = i\setminus r \quad . \tag{35}$$

In words, the left factor i is the (l,i) th entry in the matrix and its corresponding right factor $i\triangleright$ is the (i,r) th entry in the matrix.

We conclude by showing that —in Conway's words— any subfactorisation of E is dominated by a factorisation of E. In particular, we show that:

$$A{\cdot}B \subseteq E \;\equiv\; A \subseteq B\triangleleft \;\wedge\; B \subseteq B\triangleleft\triangleright \quad . \tag{36}$$

More generally, we show that, for all X and Y and all U and V ,

$$X{\cdot}Y \subseteq U\triangleleft \setminus V\triangleleft \;\equiv\; \langle \exists W :: X \subseteq U\triangleleft \setminus W\triangleleft \;\wedge\; Y \subseteq W\triangleleft \setminus V\triangleleft \rangle \quad . \tag{37}$$

The proof of (36) is:

$$
\begin{array}{ll}
 & A{\cdot}B \subseteq E \\
= & \{ \quad \text{factors, } B{\lhd} = E/B \text{ ; cancellation} \quad \} \\
 & A \subseteq B{\lhd} \;\wedge\; B{\lhd}\cdot B \subseteq E \\
= & \{ \quad \text{factors, } B{\lhd\rhd} = B{\lhd}\backslash E \quad \} \\
 & A \subseteq B{\lhd} \;\wedge\; B \subseteq B{\lhd\rhd} \quad .
\end{array}
$$

Whence, we prove (37). First, we determine a specific instance for the existentially quantified variable W :

$$
\begin{array}{ll}
 & X{\cdot}Y \subseteq U{\lhd}\backslash V{\lhd} \\
= & \{ \quad \text{factors, definition of } V{\lhd} \quad \} \\
 & U{\lhd}\cdot X\cdot Y\cdot V \subseteq E \\
= & \{ \quad \text{(36) with } A{,}B := U{\lhd}\cdot X \;,\; Y{\cdot}V \quad \} \\
 & U{\lhd}\cdot X \subseteq (Y{\cdot}V){\lhd} \;\;\wedge\;\; Y{\cdot}V \subseteq (Y{\cdot}V){\lhd\rhd} \\
= & \{ \quad \text{factors} \quad \} \\
 & X \subseteq U{\lhd}\backslash(Y{\cdot}V){\lhd} \;\;\wedge\;\; Y \subseteq (Y{\cdot}V){\lhd\rhd}/V \\
= & \{ \quad (27) \quad \} \\
 & X \subseteq U{\lhd}\backslash(Y{\cdot}V){\lhd} \;\;\wedge\;\; Y \subseteq (Y{\cdot}V){\lhd}\backslash V{\lhd} \quad .
\end{array}
$$

Then, we have:

$$
\begin{array}{ll}
 & X{\cdot}Y \subseteq U{\lhd}\backslash V{\lhd} \\
\Rightarrow & \{ \quad \text{above, } W := Y{\cdot}V \quad \} \\
 & \langle \exists W \;::\; X \subseteq U{\lhd}\backslash W{\lhd} \;\wedge\; Y \subseteq W{\lhd}\backslash V{\lhd} \rangle \\
\Rightarrow & \{ \quad \text{monotonicity of composition} \quad \} \\
 & \langle \exists W \;::\; X{\cdot}Y \subseteq (U{\lhd}\backslash W{\lhd})\cdot(W{\lhd}\backslash V{\lhd}) \rangle \\
\Rightarrow & \{ \quad \text{cancellation} \quad \} \\
 & X{\cdot}Y \subseteq U{\lhd}\backslash V{\lhd} \quad .
\end{array}
$$

This completes the proof of Conway's theorem. A "matrix" has been exhibited containing all factors and only the factors of E , indexed by left factors of E , that is reflexive and transitive and hence equal to its own star. The import of (34) and (35) is that the $E{\lhd}$ "row" of the matrix (the set of entries all having $E{\lhd}$ as first index) contains all (and only) the left factors of E , and the E "column" of the matrix (the set of entries all having E as second index) all (and only) the right factors of E . In addition, from (31) we see that E is the matrix entry at the intersection of this row and column. (Note, however, that factors and left and right factors of E , including E itself, may appear repeatedly in the matrix. Conway's wordy theorems and proofs are confusing on this point and there is one unfortunate misprint that claims exactly the opposite!: "The theorem does prevent E from occurring twice in its factor matrix" [Con71, p.49].)

3.3. Equivalence Relations on Languages

Our calculations in subsection 3.2 assume only an algebra consisting of a partially ordered monoid that admits factorisation. For languages (sets of words) and regular languages in particular more can be said about the factors. Let us recall the basic definitions and properties.

Suppose E is a subset of T^*. (That is, E is a language over the alphabet T.) Then E defines three equivalence relations on T^* — E_l, E_r and E_c — given by, for all x and y in T^*:

$$\begin{array}{rcl} xE_ly & \equiv & \langle\forall z : z\in T^* : zx\in E \equiv zy\in E\rangle \\ xE_ry & \equiv & \langle\forall z : z\in T^* : xz\in E \equiv yz\in E\rangle \\ xE_cy & \equiv & \langle\forall u,v : u\in T^* \wedge v\in T^* : uxv\in E \equiv uyv\in E\rangle \end{array}$$

These are the so-called left-invariant equivalence relation, right-invariant equivalence relation and congruence relation introduced by Rabin and Scott [RS59]. Being equivalence relations, each partitions T^* into equivalence classes. We call an equivalence class modulo E_l an r*-class* of E, an equivalence class modulo E_r an l*-class* of E, and an equivalence class modulo E_c a c*-class* of E. We use $E_r(x)$ to denote the l-class that includes word x. Similarly for $E_l(x)$ and $E_c(x)$.

Note the switch: an equivalence class modulo $E_{\underline{l}}$ is an $\underline{r}$-class. The reason for the switch is the following theorem:

Theorem 3. Each left factor of E is a union of l-classes of E, each right factor of E is a union of r-classes of E, and each factor of E is a union of c-classes of E.

Proof We show that each left factor of E is a union of l-classes of E as follows. First, for all Z, $Z\subseteq T^*$,

$$\begin{array}{ll} & x\in E/Z \\ = & \quad\{ \quad \text{definition of } / \quad \} \\ & \{x\}\cdot Z\subseteq E \\ = & \quad\{ \quad \text{definition of concatenation} \quad \} \\ & \langle\forall z : z\in Z : xz\in E\rangle \quad . \end{array}$$

Hence,

$$\begin{array}{ll} & E/Z \\ = & \quad\{ \quad \text{definition} \quad \} \end{array}$$

$\langle \cup x : x \in E/Z : \{x\} \rangle$

$=$ $\{$ $xE_r x$, idempotency of set union $\}$

$\langle \cup x : x \in E/Z : \langle \cup y : xE_r y : \{x\} \rangle \rangle$

$=$ $\{$ above and nesting $\}$

$\langle \cup x,y : \langle \forall z : z \in Z : xz \in E \rangle \wedge xE_r y : \{x\} \rangle$

$=$ $\{$ definition of E_r and substitution of equals for equals $\}$

$\langle \cup x,y : \langle \forall z : z \in Z : yz \in E \rangle \wedge xE_r y : \{x\} \rangle$

$=$ $\{$ above $\}$

$\langle \cup x,y : y \in E/Z \wedge xE_r y : \{x\} \rangle$

$=$ $\{$ nesting and definition of $E_r(y)$ $\}$

$\langle \cup y : y \in E/Z : E_r(y) \rangle$.

The remaining two properties are proved similarly.
□

Earlier we showed that a right factor of E is an intersection of derivatives of E (specifically, $L \backslash E = \langle \cap w : w \in L : \partial_w E \rangle$). Similarly, a left factor of E is the reverse of an intersection of anti-derivatives of E . (The reverse of E is the set of words obtained by reversing words in E and an anti-derivative of E is a derivative of the reverse of E .) This is the characterisation given by Conway. The above characterisation, introduced in [Bac75], is much more useful when calculating the factors of a language because such calculations use only *representatives* of the three types of equivalence class rather than the full class. We see how this works below.

Now suppose we denote the factor matrix of E by $|\overline{E}|$ (as did Conway). It is a function from pairs of left factors of E to languages. It is also reflexive and transitive. That is

$$|\overline{E}| \ \dot{\supseteq} \ \mathbf{I} \dot{\cup} |\overline{E}| \times |\overline{E}| \ .$$

(As mentioned in the introduction to this section, the operators $\supseteq$ and $\cup$ have here been extended pointwise to functions; the product operator $\times$ is defined as for matrix multiplication. The matrix $\mathbf{I}$ is the "identity" matrix: the matrix with $\{\varepsilon\}$ as entry along the diagonal and the empty set off the diagonal.) Since the set of square matrices with range a regular algebra form a regular algebra under such an extension of the operators, we conclude that $|\overline{E}| \ \dot{\supseteq} \ |\overline{E}|^*$ and hence $|\overline{E}| = |\overline{E}|^*$.

The basic theorem of regular languages [RS59] is that the following statements are all equivalent:

- E is regular (i.e. can be denoted by a regular expression).
- The relation E_l has finite index.

- The relation E_r has finite index.
- The relation E_c has finite index.

It follows that E is regular if and only if it has a finite number of factors. Thus the factor "matrix" exists for all languages E but it is only for regular languages that the use of the word "matrix" complies with standard conventions: entries are indexed by pairs of left factors but, precisely when E is a regular language, the left factors can themselves by indexed by numbers from 1 to n, for some (finite) number n.

4. The Factor Graph

Let us summarise what we have achieved so far. We have shown that every element E of a regular algebra defines a "matrix" indexed by left factors (that is, a function from pairs of left factors of E to languages), called the factor matrix, that is reflexive and transitive. Furthermore, the factor matrix $|\overline{E}|$ is equal to $|\overline{E}|^*$. We now want to show that, if E is a regular language, $|\overline{E}|$ has a unique minimal "starth root", which we call the *factor graph* of E.

Section 4.1 defines what we mean by a "starth root" and then outlines the proof that the factor matrix of a regular language has a starth root — which we call the factor graph of the matrix. Section 4.2 then shows how to construct the factor graph of a given regular language. In principle, this construction involves constructing both the machine and anti-machine of the given language but, in practice, the full details are not required. This is made clear in section 4.3 where we show that the factor graph underlies the well-known Knuth-Morris-Pratt pattern-matching algorithm. This also —at last!— gives us the opportunity to present a significant non-trivial example of the unity of opposites. Specifically, the so-called "failure function" used in the KMP algorithm represents the reflexive-transitive reduction of the subset relation on left factors of a language defined by the pattern, or, equivalently by the unity-of-opposites theorem, the superset relation on right factors of the same language.

For brevity, this section provides outlines only of the algorithms presented and omits proofs.

4.1. Existence Theorem

Suppose U is an element of a regular algebra. A *starth root* of U is any element V that satisfies $V^* = U^*$; it is *minimal* if no smaller element has this property.

By definition, U is always a starth root of itself. So starth roots exist for every element of a regular algebra. In a *free* regular algebra, it is the case that every element has a unique, minimal starth root [Brz67] but in general this is not always the case. Even when minimal starth roots exist, uniqueness is not guaranteed. For example, a minimal starth root of a relation is called a *transitive reduction* of the relation but there may be several different transitive reductions of a given relation. A specific example is the relation $\{(0,1),(1,2),(2,0)\}$ on

$\{0,1,2\}$. It is a minimal starth root of itself but so also is $\{(1,0),(2,1),(0,2)\}$. (Confusingly, the literature sometimes refers to *the* transitive reduction of a relation/graph even though in a case like this one an arbitrary choice must be made.) Unique minimal starth roots (i.e. unique transitive reductions) are guaranteed to exist for well-founded relations on a finite set (equivalently, relations that can be represented by a finite, acyclic graph).

The fact that, nevertheless, the factor matrix of a regular language has a unique minimal starth root was first proved in [Bac75] and later, using an improved argument, in [BL77]. For reasons that will be explained shortly, the name *factor graph* was given to this matrix. Let us summarise those elements of the existence proof relevant to the construction of the factor graph.

The first step is due to Conway [Con71]. He defined a matrix of languages to be a *constant* matrix if each entry is a subset of $\{\varepsilon\}$ and a *linear* matrix if each entry is a subset of T . He then defined the constant matrix $\mathbf{C}_{max}$ to be $|\overline{E}|\dot{\cap}K.\{\varepsilon\}$ (where $K.\{\varepsilon\}$ is a matrix all of whose entries are $\{\varepsilon\}$) and the linear matrix $\mathbf{L}_{max}$ to be $|\overline{E}|\dot{\cap}K.T$ and he showed, (implicitly) by induction on the length of words, that $|\overline{E}|=(\mathbf{C}_{max}\dot{\cup}\mathbf{L}_{max})^*$. Conway calls the matrix $\mathbf{C}_{max}\dot{\cup}\mathbf{L}_{max}$ a *constant+linear* matrix; we call such a matrix a *graph* because it can be represented by a directed graph with a finite number of nodes and edges labelled by ε or an element of T. (There may be more than one edge for each pair of nodes.)

The next step is to reduce the graph $\mathbf{C}_{max}\dot{\cup}\mathbf{L}_{max}$ by eliminating unnecessary edges. Since $|\overline{E}|=(\mathbf{C}_{max}\dot{\cup}\mathbf{L}_{max})^*$, we also have that $|\overline{E}|=(\mathbf{D}\dot{\cup}\mathbf{L}_{max})^*$ where $\mathbf{D}=\mathbf{C}_{max}\dot{\cap}\neg\mathbf{I}$. Now the matrix $\mathbf{D}$ corresponds to the proper subset relation on left factors of E . (There is an edge labelled ε in the graph of $\mathbf{D}$ from the node corresponding to left factor L to the node corresponding to the left factor L' exactly when $L\subseteq L'\wedge L\neq L'$.) That is, $\mathbf{D}$ is an acyclic graph and so has a unique minimal starth root $\mathbf{C}_{min}$ where

$$\mathbf{C}_{min}=\mathbf{D}\dot{\cap}\neg(\mathbf{D}\times\mathbf{D}\times\mathbf{D}^*) \quad .$$

It then follows that $\mathbf{C}_{min}\dot{\cup}\mathbf{L}_{min}$ is the unique minimal starth root of $|\overline{E}|$ where,

$$\mathbf{L}_{min} \;=\; \mathbf{L}_{max} \;\dot{\cap}\; \neg(\mathbf{D}\times\mathbf{L}_{max}) \;\dot{\cap}\; \neg(\mathbf{L}_{max}\times\mathbf{D}) \quad .$$

The matrix $\mathbf{C}_{min}\dot{\cup}\mathbf{L}_{min}$ is what we call the *factor graph* of E . More precisely, it is a (pointwise) union of constant and linear matrices and thus corresponds to a graph with nodes indexed by the left factors of E and edges labelled by the empty word or an element of the alphabet. (There may be multiple edges from one node to another.) The subgraph $\mathbf{C}_{min}$ is the transitive reduction of $\mathbf{C}_{max}$. That is, the relation corresponding to $\mathbf{C}_{min}$ is the transitive reduction of the proper subset relation on left factors of E . Equivalently, by the unity-of-opposites theorem it is transitive reduction of the proper *superset* relation on right factors of E .

Example 6. Figs. 1(d) and 1(e) display the graphs of $\mathbf{C}_{max}\dot{\cup}\mathbf{L}_{max}$ and $\mathbf{C}_{min}\dot{\cup}\mathbf{L}_{min}$, respectively, for the language T^*aT^* , where $T=\{a,b\}$. It is easy to check that

$$(\mathbf{C}_{max} \dot{\cup} \mathbf{L}_{max})^* = (\mathbf{C}_{min} \dot{\cup} \mathbf{L}_{min})^* = |\overline{T^* a T^*}| \ .$$

Recall that the factor matrix $|\overline{T^* a T^*}|$ is diplayed in fig. 1(c).

After reading the next section, the reader should be able to construct these graphs themself.

□

4.2. Construction

Suppose E is a regular language. The construction of the factor graph involves calculating all the left factors of E whilst simultaneously calculating $\mathbf{C}_{min}$ and $\mathbf{L}_{min}$. The definition of $\mathbf{C}_{min}$ and $\mathbf{L}_{min}$ in terms of $\mathbf{C}_{max}$ and $\mathbf{L}_{max}$ is exploited in this process but calculating the full details of the latter matrices is avoided as far as possible. In this section, we show how this is done.

We illustrate the construction using the language $((a{+}b)^* \, c \, a^* \, (a{+}b))^*$ with alphabet $\{a,b,c\}$.

Fig. 2 shows the machine and anti-machine for our example language.

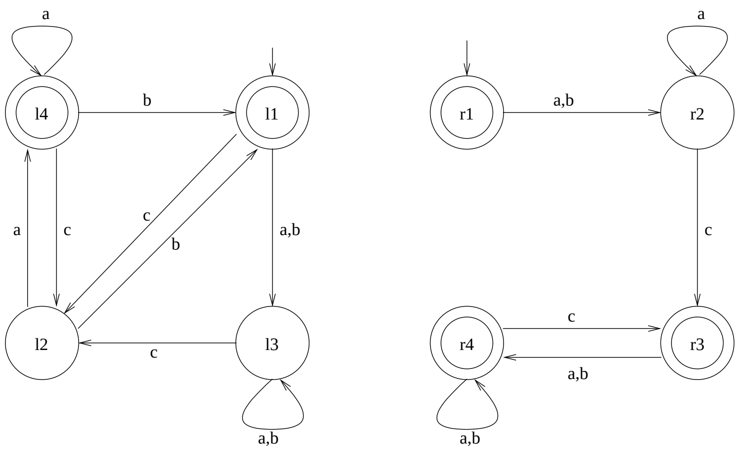


Figure 2: Machine and Anti-Machine of $((a{+}b)^* \, c \, a^* \, (a{+}b))^*$

The machine and anti-machine as shown are "all-admissible". This means that in each a node has been omitted from which there is no path to the final state. Such nodes are said to be *inadmissible.*

We recall that each left factor of E is a union of l-classes of E and each right factor is a union of r-classes of E. Moreover, the l-classes of E are in one-to-one correspondence with the nodes of the machine of E (the reduced, deterministic finite automaton that recognises E) and the r-classes of E are in one-to-one correspondence with the nodes of the anti-machine of E (the reduced, deterministic finite automaton that recognises the reverse of E).

Specifically, for a given state of the machine, the corresponding l-class is the set of all words recognised by that state (i.e. the set of all words spelt out by a path from the start state to the state); for a given state of the anti-machine, the corresponding r-class is the reverse of the set of all words recognised by that state.

The table below names each l-class and each r-class for our example language. The names are those used in fig. 2 with the addition of the names $l5$ and $r5$ of the inadmissible states of machine and anti-machine, respectively. Next to each name of an l-class is an element of the class; next to each name of an r-class is the reverse of an element of the class. We call these *representatives* of the class. For example, ca is a representative of class $l4$ because the word ca spells out a path from the start state $l1$ to the state $l4$. Similarly, ca is a representative of $r3$ because its reverse ac spells out a path from the start state $r1$ to the state $r3$.

l-class	representative	r-class	representative
$l1$	ε	$r1$	ε
$l2$	c	$r2$	a
$l3$	a	$r3$	ca
$l4$	ca	$r4$	aca
$l5$	cc	$r5$	c

After constructing the machine and anti-machine and choosing representatives of the l- and r-classes in this way, the next step is to calculate $l\triangleright$ and $r\triangleleft$ for each of the classes. Each entry $l\triangleright$ is a right factor and thus a union of r-classes, and each entry $r\triangleleft$ is a left factor and thus a union of l-classes. The table below shows the result of the calculation for our example language.

l-class l	$l\triangleright$	r-class r	$r\triangleleft$
$l1$	$r1\cup r3\cup r4$	$r1$	$l1\cup l4$
$l2$	$r2\cup r4$	$r2$	$l2\cup l4$
$l3$	$r3\cup r4$	$r3$	$l1\cup l3\cup l4$
$l4$	$r1\cup r2\cup r3\cup r4$	$r4$	$l1\cup l2\cup l3\cup l4$
$l5$	$\emptyset$	$r5$	$\emptyset$

Calculating the entries is made easy by the use of representatives. Specifically, suppose u is the representative of l and v is the representative of r. Then $r\subseteq l\triangleright$ exactly when $uv\in E$ (which is easily checked using the machine of E). In our example, $l3\triangleright = r3\cup r4$ because aca and $aaca$ are elements of E but no other concatenation of a (the chosen representative of $l3$) and a representative of an r-class is in E. The same process is used to calculate $r\triangleleft$ for each r-class r.

The next step is to determine all the left factors and all the right factors. Simultaneously, we calculate the matrix $\mathbf{C}_{min}$ exploiting the correspondence between $\mathbf{C}_{min}$ and the transitive reduction of the subset ordering on left factors. This is the most laborious process since, for each r-class r, the set $r\lhd$ is a left factor but there will typically be more left factors. It is necessary to consider *all* subsets of the set of l-classes to determine whether or not it is a left factor. Suppose L is a union of l-classes. Then L is a left factor of E equivales $L = L\rhd\lhd$. Fortunately this property is straightforward to check using the information that has already been computed. We have

$$L\rhd \;=\; \langle\cup l : l\subseteq L : l\rangle\rhd \;=\; \langle\cap l : l\subseteq L : l\rhd\rangle$$

and, for any R that is a union of r-classes,

$$R\lhd \;=\; \langle\cup r : r\subseteq R : r\rangle\lhd \;=\; \langle\cap r : r\subseteq R : r\lhd\rangle \quad .$$

(In these equations, the dummies l and r range over l- and r-classes, respectively.) For example,we can compute from the table above that

$$(l2\cup l4)\rhd = l2\rhd\cap l4\rhd = (r2\cup r4)\cap(r1\cup r2\cup r3\cup r4) = r2\cup r4$$

and

$$(r2\cup r4)\lhd = r2\lhd\cap r4\lhd = (l2\cup l4)\cap(l1\cup l2\cup l3\cup l4) = l2\cup l4 \quad .$$

In this way, we have checked that $l2\cup l4$ is indeed a left factor. However, we have that

$$(l1\cup l3)\rhd = l1\rhd\cap l3\rhd = (r1\cup r2\cup r3)\cap(r3\cup r4) = r3$$

and

$$r3\lhd = l1\cup l3\cup l4 \quad .$$

So $(l1\cup l3)\rhd\lhd \neq l1\cup l3$ and, hence, $l1\cup l3$ is not a left factor.

Computing the poset of left factors and simultaneously $\mathbf{C}_{min}$ involves a search of the set of subsets of the l-classes in decreasing order of size. In our example, the sizes of the subsets range from 5 to 0. So 2^5 subsets must be examined. The subsets that are determined to be left factors are accumulated in a set that we call Γ below.

Beginning with T^* (the union of all l-classes) —which is always a left factor— as the only element of Γ, each subset L is checked for the property $L = L\rhd\lhd$; if it does, then the (L, L')th entry in $\mathbf{C}_{min}$ is set to ε for all L' in Γ that satisfy $L\subseteq L'$ and there is no L'' different from L' in Γ such that $L\subseteq L''\subseteq L'$. On completion, all other entries in $\mathbf{C}_{min}$ are set to $\emptyset$.

The table below shows for our example language each left factor as a union of l-classes and each right factor as a union of r-classes. Fig. 3 shows the reflexive-transitive reduction of the subset relation on left factors. The corresponding entries in the matrix $\mathbf{C}_{min}$ are ε where there is an edge and $\emptyset$ where there is no edge.

Left factor		Right factors	
$L0$	T^*	$R0$	$\emptyset$
$L1$	$l1 \cup l2 \cup l3 \cup l4$	$R1$	$r4$
$L2$	$l1 \cup l3 \cup l4$	$R2$	$r3 \cup r4$
$L3$	$l2 \cup l4$	$R3$	$r2 \cup r4$
$L4$	$l1 \cup l4$	$R4$	$r1 \cup r3 \cup r4$
$L5$	$l4$	$R5$	$r1 \cup r2 \cup r3 \cup r4$
$L6$	$\emptyset$	$R6$	T^*

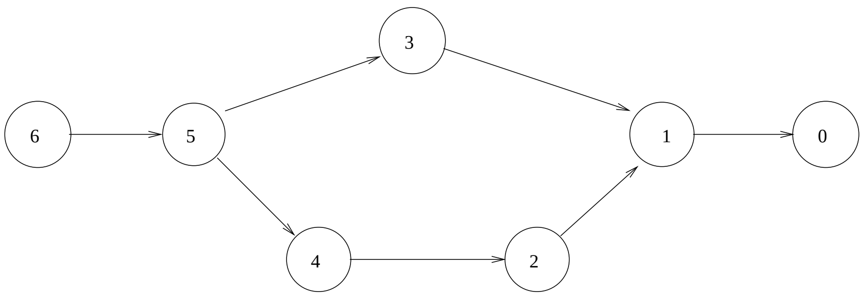


Figure 3: Poset of left factors, Inverted poset of right factors

It is now that we can see the unity of opposites in action. Not only is there the one-to-one correspondence between left and right factors observed by Conway but there is also an isomorphism between the poset of left factors and the poset of right factors. Fig. 3 also shows the reflexive-transitive reduction of the superset ordering on right factors. Moreover, infima and suprema of left and right factors correspond in the way predicted by the unity-of-opposites theorem. For example, the supremum of $L2$ and $L3$ is $L5$; correspondingly, the infimum of $R2$ and $R3$ is $R5$.

The process of constructing the factor graph is completed by constructing the matrices $\mathbf{L}_{max}$ and $\mathbf{L}_{min}$. Suppose L and L' are left factors; let the right factor corresponding to L' be R'. Then the symbol a is an entry in the (L, L') th position in $\mathbf{L}_{max}$ if $L{\cdot}a{\cdot}R' \subseteq E$. The representatives of the l-and r-classes can be used to check this property. We have to check for each representative u of an l-class in L and each representative v of an r-class in R' whether or not $uav{\in}E$. The symbol a is an entry in the (L, L') th position in $\mathbf{L}_{min}$ if L and R' are maximal in $L{\cdot}a{\cdot}R' \subseteq E$. (That is, if left factor L'' is such that $L''{\cdot}a{\cdot}R' \subseteq E$ then $L'' \subseteq L$, and if right factor R'' is such that $L{\cdot}a{\cdot}R'' \subseteq E$ then $R'' \subseteq R'$.) The factor graph of our example language is shown in fig. 4.

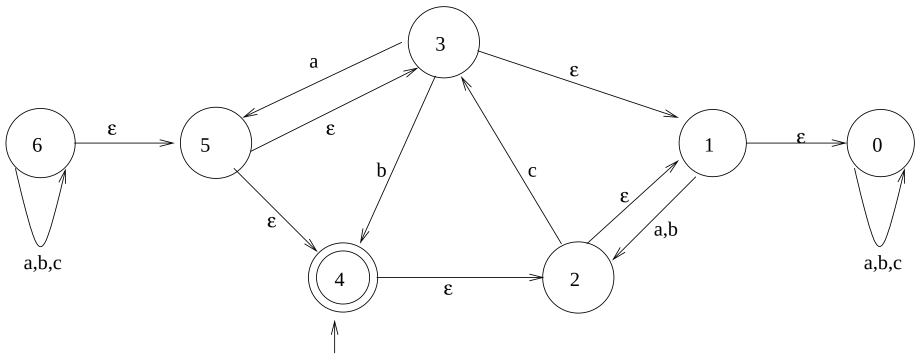


Figure 4: Factor Graph

4.3. Knuth-Morris-Pratt Pattern Matching

The fact that it is theoretically necessary to consider all subsets of the l-classes (2^n if the number of l-classes is n) when computing the factor graph is clearly prohibitive. However, there are circumstances when the computation is relatively straightforward and efficient. The well-known Knuth-Morris-Pratt pattern-matching algorithm [KMP77] (henceforth abbreviated to "KMP algorithm") and its generalisation to a finite set of patterns [AC75] boil down to constructing the factor graph of a language defined by the given patterns [BL77].

Expressed in terms of language recognition, *pattern matching* is the following: given words u and v in T^*, detemine, for each prefix w of u, whether $w \in T^* v$. The word u is called the *text* and v is called the *pattern.* Informally, pattern matching is the task of searching a text u to determine all occurrences of the pattern v.

In this section, we use an example to illustrate how the unity of opposites becomes apparent in a practical application like pattern matching. Underlying our exposition is the fact that the KMP algorithm effectively computes the factor graph of $T^* v$ for a given pattern v. However, our goal is not to reformulate the KMP algorithm. We do show how to construct the factor graph but we undertake more computation than is necessary in order to increase understanding. Our main goal is to provide an appealing and possibly inspiring example.

The section begins with a summary of the construction of the factor graph $T^* v$ for an arbitrary pattern v following which we detail the construction for a specific example. The section ends with a short summary of how the factor graph encodes the failure function used in the KMP algorithm. The KMP algorithm is very much standard textbook material and so we take the liberty of assuming that it needs only a brief explanation.

The key observation is that the construction of a factor graph of a given language does not necessitate computing the language's machine and anti-machine, it is only necessary to find a representative of each l-class and each r-class and, in the case of pattern matching, it is easy to identify such representatives.

To avoid a case analysis with no practical benefit, we assume that the pattern v has length greater than zero and the alphabet T has at least two symbols. The reader may wish to consult fig. 5 whilst reading the following summary.

Given a pattern v of length n (where n is non-zero), the language $T^* v$ has $n+2$ left factors: $n+1$ of the form $T^* w$ where w is a prefix of v and (annoyingly!) the empty set[10]. Thus, the factor graph of $T^* v$ has $n+2$ nodes. One node we call the *inadmissible* node because there is no edge to this node from the start node of the graph. Note that there is a one-to-one correspondence between non-empty left factors and prefixes of v. Let us use the length of the prefix to index the non-empty left factors. (So left factor 0 is T^* and left factor n is $T^* v$.) The edges of the factor graph comprise four sets: The first set we call the *spine* of the factor graph. It has n elements; there is an edge labelled a_i from node $i-1$ to node i if a_i is the ith letter of v. The edges in the second set are from node 0 to itself and each is labelled by a symbol in T different from a_1. The edges in the third set are all labelled by the empty word. These edges represent the (reflexive-)transitive reduction of the subset relation on the left factors of $T^* v$ in the following sense: if L and M are left factors, there is an edge labelled ε from the node corresponding to L to the node corresponding to M if $L \subseteq M$ but L and M are different (the reflexive reduction) and there is no left factor N different from both L and M such that $L \subseteq N \subseteq M$ (the transitive reduction). Finally, there are edges labelled with the symbols in T from the inadmissible node to itself.

The factor graph acts as a recogniser of the language $T^* v$. The start node is node 0 and the final node is node n. When at a node, a transition is made along an edge with label that matches the next symbol in the text; if no such transition is possible the (unique) empty-word edge from the node is followed and then a transition is chosen again. In this way, the empty-word edges act as *failure* transitions; that is, they are followed when it is impossible to follow an edge labelled by a symbol of the alphabet.

The fact that it is easy to compute the factor graph of $T^* v$ is a consequence of the fact that both the machine and anti-machine have exactly the same spine and computing the factor graph requires only knowing a representative of each l-class and each r-class; it is not necessary to know all the details of the machine and anti-machine. To be precise, each l-class of $T^* v$ is represented by a prefix of v and each admissible r-class is represented by a suffix of v. (The anti-machine has one inadmissible r-class whenever the alphabet has at least two symbols.)

The table below names the l- and r-classes for the language $\{a,b\}^* abaab$ together with their representatives. (The inadmissible r-class is named "err", short for "error" in the table. It is this r-class that gives rise to the inadmissible node in the factor graph.)

[10]If $n=0$ or the alphabet has just one element, the empty set is not a left factor. This is the case analysis we want to avoid.

l-class	representative	r-class	representative
$l0$	ε	$r0$	ε
$l1$	a	$r1$	b
$l2$	ab	$r2$	ab
$l3$	aba	$r3$	aab
$l4$	$abaa$	$r4$	$baab$
$l5$	$abaab$	$r5$	$abaab$
		err	a

Using the algorithm detailed in section 4.2, the table below shows the left factor $r\lhd$ for each r-class r and the right factor $l\rhd$ for each l-class l. We include this table for illustration purposes only since it is not necessary to calculate it when calculating the factor graph.

l-class l	$l\rhd$	r-class r	$r\lhd$
$l0$	$r5$	$r0$	$l5$
$l1$	$r4\cup r5$	$r1$	$l4$
$l2$	$r3\cup r5$	$r2$	$l3$
$l3$	$r2\cup r4\cup r5$	$r3$	$l2\cup l5$
$l4$	$r1\cup r4\cup r5$	$r4$	$l1\cup l3\cup l4\cup l5$
$l5$	$r0\cup r3\cup r5$	$r5$	$l0\cup l1\cup l2\cup l3\cup l4\cup l5$
		err	$\emptyset$

Since we know that the language has exactly 6 admissible left/right factors —the first six entries $l\rhd$ and $r\lhd$ in the table above— it is now easy to compute $\mathbf{C}_{min}$, the transitive reduction of the subset ordering on the poset of left factors. Moreover, $\mathbf{L}_{min}$ is already known. Its principal component is what we called the "spine" of the factor graph; other edges in $\mathbf{L}_{min}$ are the edges from node 0 to itself and edges from the inadmissible node. The resulting all-admissible factor graph is shown in fig. 5. (The inadmissible node has been omitted; if included, there would be a dotted edge from it to each of the rightmost three nodes in the graph, and there would be edges from it to itself, one for each symbol of T.) The dotted edges correspond to $\mathbf{C}_{min}$; their labels (not shown) are all the empty word. Below each node of the graph, we also show the l-classes that comprise the corresponding left factor. By the unity of opposites, each node also corresponds to a right factor; the r-classes making up the corresponding right factor are also listed under each node.

Note once more the properties predicted by the unity-of-opposites theorem. Not only do the empty-word edges encode the subset ordering on left factors, they also encode the superset ordering on the right factors. Specifically, from left to right in fig. 5, the empty-word edges encode the following properties:

$$\begin{array}{lll} l0\cup l1\cup l2\cup l3\cup l4\cup l5 \supseteq l1\cup l3\cup l4\cup l5 & \text{and} & r5\subseteq r4\cup r5\ , \\ l0\cup l1\cup l2\cup l3\cup l4\cup l5 \supseteq l2\cup l5 & \text{and} & r5\subseteq r3\cup r5\ , \end{array}$$

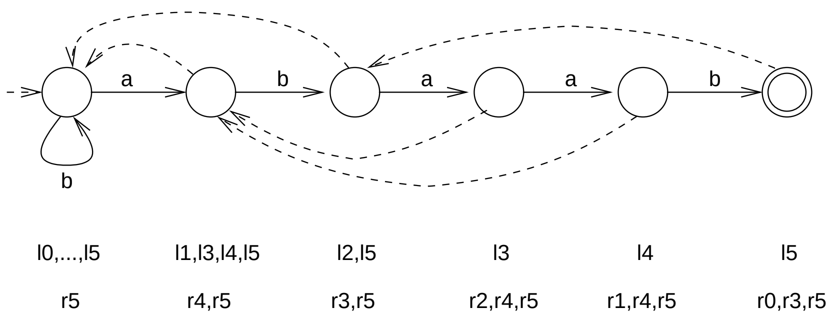


Figure 5: Factor Graph of $T^* abaab$ (omitting inadmissible node).

$$\begin{array}{lll} l1 \cup l3 \cup l4 \cup l5 \supseteq l3 & \text{and} & r4 \cup r5 \subseteq r2 \cup r4 \cup r5 \ , \\ l1 \cup l3 \cup l4 \cup l5 \supseteq l4 & \text{and} & r4 \cup r5 \subseteq r1 \cup r4 \cup r5 \ , \\ l2 \cup l5 \supseteq l5 & \text{and} & r3 \cup r5 \subseteq r0 \cup r3 \cup r5 \ . \end{array}$$

Not listed are the orderings encoded by the three empty-word edges from the inadmissible node. These encode that the left factor $\emptyset$ is the least element in the subset ordering on left factors and the right factor T^* (the union of all r-classes) is the greatest element in the subset ordering on right factors.

A fundamental component of the Knuth-Morris-Pratt algorithm is the computation of a function defined on prefixes of the pattern v different from the empty word [KMP77, p.327]. Specifically, suppose a_i is the ith letter of v. Then, the *failure function* f of type $\{1 \ \dots \ n\} \to \{0 \ \dots \ n{-}1\}$ is defined by

$$f(i) \ = \ \langle MAX\ j \ : \ j < i \ : \ a_1 \ \dots \ a_j \ = \ a_{i-j+1} \ \dots \ a_i \rangle \ .$$

To see the correspondence with the factor graph, it suffices to observe the equivalent definition:

$$f(i) \ = \ \langle MAX\ j \ : \ j < i \ : \ T^* a_1 \ \dots \ a_i \subseteq T^* a_1 \ \dots \ a_j \rangle \ .$$

The equivalence of these two definitions is a straightforward calculation. First, we establish that

$$T^* a_1 \ \dots \ a_i \subseteq T^* a_1 \ \dots \ a_j \quad \equiv \quad a_1 \ \dots \ a_i \in T^* a_1 \ \dots \ a_j$$

by an if-and-only-if argument:

$$\begin{array}{ll} & T^* a_1 \ \dots \ a_i \subseteq T^* a_1 \ \dots \ a_j \\ \Leftarrow & \{ \quad \text{monotonicity of concatenation, } T^* = T^* T^* \quad \} \\ & a_1 \ \dots \ a_i \in T^* a_1 \ \dots \ a_j \\ \Leftarrow & \{ \quad a_1 \ \dots \ a_i \in T^* a_1 \ \dots \ a_i \quad \} \\ & T^* a_1 \ \dots \ a_i \subseteq T^* a_1 \ \dots \ a_j \ . \end{array}$$

Then we simplify the right side:

$$
\begin{array}{ll}
 & a_1 \ \ldots \ a_i \in T^* a_1 \ \ldots \ a_j \\
= & \{ \quad \text{definition of concatenation} \quad \} \\
 & \langle \exists u : u \in T^* : a_1 \ \ldots \ a_i \ = \ u\, a_1 \ \ldots \ a_j \rangle \\
= & \{ \quad \text{equality of words: } u := a_1 \ \ldots \ a_{i-j} \quad \} \\
 & j \leq i \ \wedge \ a_{i-j+1} \ \ldots \ a_i \ = \ a_1 \ \ldots \ a_j \ .
\end{array}
$$

We conclude that

$$T^* a_1 \ \ldots \ a_i \subseteq T^* a_1 \ \ldots \ a_j \ \equiv \ j \leq i \ \wedge \ a_{i-j+1} \ \ldots \ a_i \ = \ a_1 \ \ldots \ a_j$$

and hence that the failure function represents the transitive reduction of the subset relation on the n nonempty left factors of v .

5. Conclusion

In the more than forty years since the publication of Conway's book, much has changed in the science of computing. The demands of reliability of software that affects our livelihoods and our lives on a daily basis has meant that reasoning in many (although not all) computing science publications has become much more formal and calculational. This may mean that the publications are longer but they become clearer by becoming more explicit. The added benefit is that we have become much more aware of the algebraic properties that underpin any computation.

The algebraic properties of Galois connections have now become widely known but it is always useful to have meaningful examples. Such examples are lacking (at least to my knowledge) for one of the most important theorems: the unity of opposites. Because of the decidedly non-trivial nature of factor theory, using the construction of the factor graph of a regular language to exemplify the theorem is quite demanding. However, by relating it to a practical application —the Knuth-Morris-Pratt pattern matching algorithm— we hope that it is both meaningful and worthwhile.

Acknowledgements

Many thanks to the anonymous referees for their thoughtful comments and suggestions for improvements.

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[Bac75] R.C. Backhouse. *Closure algorithms and the star-height problem of regular languages.* PhD thesis, University of London, 1975. Scanned-in copy of the chapters on factor theory available from `www.cs.nott.ac.uk/~rcb/MPC/FactorGraphs.pdf`.

[Bac06] Roland Backhouse. Regular algebra applied to language problems. *Journal of Logic and Algebraic Programming*, 66:71–111, 2006.

[BL77] R.C. Backhouse and R.K. Lutz. Factor graphs, failure functions and bi-trees. In A. Salomaa and M. Steinby, editors, *Fourth Colloquium on Automata, Languages and Programming*, pages 61–75. Springer-Verlag, LNCS 52, July 1977.

[Brz64] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11(4):481–494, October 1964.

[Brz67] J.A. Brzozowski. Roots of star events. *Journal of the ACM*, 14(3):466–477, July 1967.

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

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6:325–350, June 1977.

[Koz94] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 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.

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J.Research and Development*, 1959.

[Sal66] Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. Assoc. Comp. Mach.*, 13(1):158–169, January 1966.

[Sch53] J. Schmidt. Beiträge zur Filtertheorie. II. *Math. Nachr.*, 10:197–232, 1953.

[Wei73] P. Weiner. Linear pattern matching algorithms. In *Conf. Record IEEE 14th Annual Symposium on Switching and Automata*, pages 1–11, 1973.