CDM

Algebra of Regular Languages

Klaus Sutner

Carnegie Mellon University

Hermann Weyl

Our mathematics of the last decades has wallowed in generalizations and formalizations. But one misunderstands this tendency if one thinks that generality was sought merely for generality's sake. The real aim is simplicity: natural generalization simplifies since it reduces the assumptions that have to be taken into account.

Axiomatic versus Constructive Procedures in Mathematics, 1953

Definition

Definition

A Kleene algebra is a structure

$$\langle A; +, \cdot, {}^*, 0, 1\rangle$$

where

- $\langle A, +, 0\rangle$ is a commutative monoid
- $\langle A, \cdot, 1\rangle$ is a monoid (usually not commutative)
- 0 is a multiplicative null: $0 \cdot x = x \cdot 0 = 0$
- distributivity: $x \cdot (y + z) = x \cdot y + x \cdot z$ and $(y + z) \cdot x = y \cdot x + z \cdot x$
- sumstar identity: $(x + y)^\star = (x^\star y)^\star x^\star$
- prodstar identity: $(x \cdot y)^\star = 1 + x \cdot (y \cdot x)^\star \cdot y$
- starstar identity: $(x^\star)^\star = x^\star$
- powerstar identity: $x^\star = (x^n)^\star x^{<n}$

The powerstar axiom holds for all $n \geq 1$ where $x^{<n} = 1 + x + x^2 + \ldots + x^{n-1}$

Quoi?

The star axioms may seem strange, but the main goal is fairly simple: we would like to axiomatize the language semiring over Σ:

$$\mathcal{L}(\Sigma) = \langle \mathfrak{P}(\Sigma^\star); \cup, \cdot, {}^\star, \emptyset, \varepsilon \rangle$$

It is straightforward to verify that the language semiring (and, more importantly, certain subsemirings such as the regular languages over Σ) is indeed a Kleene algebra.

As usual, we will write xy rather than $x \cdot y$. The Kleene star or asterate operation is often used to denote the transitive closure; we will never ever do this and write something like tcl instead.

The Difference

The major difference between Kleene algebras and more familiar structures such as groups, fields or semirings is that we are dealing with in infinitary operation. For intuition, think of the star operation as

$$x^\star = 1 + x + x^2 + \dots$$

some sort of power series.

More precisely, we can easily generalize the usual binary addition operation to a mulitadic operation Σ:

$$\begin{aligned} \Sigma(\mathsf{nil}) &= 0 \\ \Sigma(x) &= x \\ \Sigma(x_1, \dots, x_k) &= \Sigma(x_1 + x_2, \dots, x_k) \qquad k \geq 2 \end{aligned}$$

This is the left associative version, but since $\langle A, +, 0 \rangle$ is associative any other definition would produce the same result.

Going Infinite

Burning Question: What about $\Sigma(x_0, x_1, x_2, \ldots)$?

Note that we are not interested in analysis here, we do not want to deal with limits and the like. As it turns out, we need to explain how our Σ operator behaves with respect to slightly more general index sets.

- $\Sigma_\emptyset = 0$
- $\Sigma_I \, \Sigma_{J_i} \, x_j = \Sigma_J \, x_j$ where $J = \bigcup_{i \in I} J_i$
- $\Sigma_I \, x_i \Sigma_J \, y_j = \Sigma_{I \times J} \, x_i y_j$
- $x^\star = \Sigma_\mathbb{N} \, x^n$

Order

It can be surprisingly useful to think of a partial order associated with any Kleene algebra:

$$x \leq y :\Leftrightarrow x + y = y$$

For language semirings this is just ordinary set inclusion.
For example, the equation $x = ux + v$ has $u^\star v$ as its least solution:

$$uu^\star v + v = (uu^\star + 1)v = u^\star v$$

and the solution is unique when $\varepsilon \notin u$.

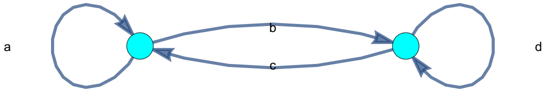
Matrices

Consider the collection $\mathcal{K}^{n\times n}$ of all $n \times n$ matrices over some Kleene algebra $\mathcal{K}$. We can add and multiply them in the usual way, and define the asterate via the infinite sum.

Claim: $\mathcal{K}^{n\times n}$ is again a Kleene algebra.

There is a "closed form" description of the star operation. For simplicity let $n = 2$.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} (a + bd^\star c)^\star & a^\star b(d + ca^\star b)^\star \\ d^\star c(a + bd^\star c)^\star & (d + ca^\star b)^\star \end{pmatrix}$$



Application: Regular Languages

Let $\mathcal{M}$ be a finite state machine over state set Q, not necessarily deterministic. Define a $Q \times Q$ matrix A by setting

$$A(p,q) = \sum (a \mid p \overset{a}{\to} q)$$

Let I and F be 0/1 vectors indicating the initial and final states in $\mathcal{M}$.

Proposition

$\mathcal{L}(\mathcal{M}) = I \cdot A^{\star} \cdot F.$

Language Semiring

From now on, we will focus on one particular Kleene algebra, the language semiring

$$\mathcal{L}(\Sigma) = \langle \mathfrak{P}(\Sigma^\star), \cup, \cdot, {}^\star, \emptyset, \{\varepsilon\} \rangle$$

As defined, this is an uncountable structure, but we will be mostly interested in the case where the carrier set is just the regular languages over Σ.

In order to emphasize the algebraic angle, we will often write $+$ instead of $\cup$, 1 instead of $\{\varepsilon\}$, and so on.

Atoms

Note that, strictly speaking, a word w is not an element of $\mathcal{L}(\Sigma)$. But the singleton $\{w\}$ is, and so it makes sense to be sloppy with notation and identify the two.

If that sends shivers up and down your type-theoretic spine note that we can filter out singletons using only algebra.

In any Kleene algebra, define x to be an atom if $x \neq 0$ but $y \leq x$ implies $y = 0$ or $y = x$.

For example, 1 is an atom.

In the language semiring, atoms are exactly the singletons.

Quotients

How about the missing operations, subtraction and division?

For subtraction we would need an additive cancellation monoid: $x + y = x + z$ implies $y = z$. This is hopelessly false in our setting: $x + x = x = x + 0$.

So how about some operation resembling division? Since our multiplication is not commutative, let's focus on left division for the time being. Here is a plausible approach.

Definition

Let $L \subseteq \Sigma^\star$ be a language and $x \in \Sigma^\star$. The left quotient of L by x is

$$x^{-1} L = \{ y \in \Sigma^\star \mid xy \in L \}.$$

So we are simply removing a prefix x from all words in the language that start with this prefix. If there is no such prefix we get an empty quotient.

Rant on Notation

It is standard to write left quotients as

$$x^{-1}\, L$$

Here is the bad news: left quotients are actually a right action of $\Sigma^\star$ on $\mathcal{L}(\Sigma)$.

As a consequence, the first law of left quotients below looks backward at first sight.

We could fix the problem by writing something like L/x but that's awkward since it seems to suggest that we are removing a suffix.

Algebra of Quotients

Lemma

Let $a \in \Sigma$, $x, y \in \Sigma^\star$ and $L, K \subseteq \Sigma^\star$. Then the following hold:

- $(xy)^{-1}L = y^{-1}x^{-1}L$,
- $x^{-1}(L \odot K) = x^{-1}L \odot x^{-1}K$ *where* $\odot$ *is one of* $\cup$, $\cap$ *or* $-$,
- $a^{-1}(LK) = (a^{-1}L)K + \chi_L\, a^{-1}K$,
- $a^{-1}L^\star = (a^{-1}L)\, L^\star$.

Here we have used the abbreviation χ_L to simplify notation:

$$\chi_L = \begin{cases} 1 & \text{if } \varepsilon \in L, \\ 0 & \text{otherwise.} \end{cases}$$

So χ_L is either zero or one in the language semiring and simulates an if-then-else.

Comments

Note that $(xy)^{-1}L = y^{-1}x^{-1}L$ and NOT $x^{-1}y^{-1}L$. As already mentioned, the problem is that algebraically left quotients are a right action.

Quotients coexist peacefully with Boolean operations, we can just push the quotients inside.

But for concatenation and Kleene star things are a bit more involved; the lemma makes no claims about the general case where we divide by a word rather than a single letter.

Exercise

Prove the last lemma.

Exercise

Generalize the rules for concatenation and Kleene star to words.

Killer App

The ultimate reason we are interested in quotients is that they provide an elegant tool to construct the minimal automaton for a regular language. And the associated algorithms can be made very efficient.

For the time being, though, let us focus on the algebra. We write $\mathcal{Q}(L)$ for the set of all quotients of a language L.

How would we go about computing $\mathcal{Q}(L)$?

In general this will be difficult, but for languages described in terms of Kleene's operations we can use algebra (there is a little glitch, though).

YAFP

Abstractly, this is yet another fixed point problem: we need to compute the least set $\mathcal{X} \subseteq \mathcal{L}(\Sigma)$ such that

- $L \in \mathcal{X}$ and
- $\mathcal{X}$ is closed under a^{-1} for all $a \in \Sigma$.

The corresponding monotonic operation $F : \mathfrak{P}(\mathcal{L}(\Sigma)) \to \mathfrak{P}(\mathcal{L}(\Sigma))$ mapping families of languages to families of languages is

$$F(\mathcal{X}) = \mathcal{X} \cup \{\, a^{-1}X \mid X \in \mathcal{X}, a \in \Sigma \,\}$$

and we are looking for the least fixed point of $\{L\}$ under F. The fixed point exists by Knaster-Tarski.

Quotients Example 1

Using the lemma, we can compute the quotients of a^*b.

$$\begin{aligned} a^{-1}\, a^*b &= a^*b \\ b^{-1}\, a^*b &= \varepsilon \\ a^{-1}\, \varepsilon &= \emptyset \\ b^{-1}\, \varepsilon &= \emptyset \\ a^{-1}\, \emptyset &= \emptyset \\ b^{-1}\, \emptyset &= \emptyset \end{aligned}$$

Thus $\mathcal{Q}(a^*b)$ consists of: a^*b, ε and $\emptyset$.

Quotients Example 1, Contd.

Note that these equations between quotients really determine the transitions in a finite state machine:

$$\begin{array}{cc}
a^{-1}\, a^*b = a^*b & a^*b \xrightarrow{a} a^*b \\
b^{-1}\, a^*b = \varepsilon & a^*b \xrightarrow{b} \varepsilon \\
a^{-1}\, \varepsilon = \emptyset & \varepsilon \xrightarrow{a} \emptyset \\
b^{-1}\, \varepsilon = \emptyset & \varepsilon \xrightarrow{b} \emptyset \\
a^{-1}\, \emptyset = \emptyset & \emptyset \xrightarrow{a} \emptyset \\
b^{-1}\, \emptyset = \emptyset & \emptyset \xrightarrow{b} \emptyset
\end{array}$$

Quotients Example 2

Sometimes it is important to keep track of the words that produce a particular quotient. E.g., let L be the finite language $\{a, aab, bbb\}$.

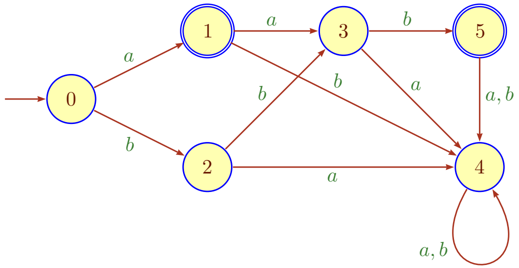
This time $\mathcal{Q}(L)$ has size 6, with witnesses as follows:

x	$x^{-1}\,L$
ε	$\{a, aab, bbb\}$
a	$\{\epsilon, ab\}$
b	$\{bb\}$
bb	$\{b\}$
aab	$\{\epsilon\}$
ab	$\emptyset$

Of course the witness x is not uniquely determined, for example $(abz)^{-1}\,L = (baz)^{-1}\,L = \emptyset$ for any z. The table lists the length-lex minimal witness in each case (which is the appropriate order for many algorithms).

Quotients Example 2.5

Moreover, there happens to be a “natural” DFA for L that has six states.



Could this be coincidence? Nah, more later . . .

Quotients Example 3

A slightly larger example, $L = L_1 = a^*b^* \cup bab$.

$a^{-1}L_1$	a^*b^*	L_2
$b^{-1}L_1$	$b^* \cup ab$	L_3
$a^{-1}L_2$	L_2	
$b^{-1}L_2$	b^*	L_4
$a^{-1}L_3$	b	L_5
$b^{-1}L_3$	L_4	
$a^{-1}L_4$	$\emptyset$	L_6
$b^{-1}L_4$	L_4	
$a^{-1}L_5$	L_6	
$b^{-1}L_5$	ε	L_7
$a^{-1}L_{6/7}$	L_6	
$b^{-1}L_{6/7}$	L_6	

Exercise

Verify this table.

Quotients Example 4

An even larger example, $L = L_1 = a^*ba^* \cup b^*ab^*$.

$a^{-1}L_1$	$a^*ba^* + b^*$	L_2	$b^{-1}L_5$	b^*	L_8
$b^{-1}L_1$	$b^*ab^* + a^*$	L_3	$a^{-1}L_6$	b^*	
$a^{-1}L_2$	a^*ba^*	L_4	$b^{-1}L_6$	b^*ab^*	
$b^{-1}L_2$	$a^* + b^*$	L_5	$a^{-1}L_7$	b^*	
$a^{-1}L_3$	$a^* + b^*$		$b^{-1}L_7$	$\emptyset$	L_9
$b^{-1}L_3$	b^*ab^*	L_6	$a^{-1}L_8$	$\emptyset$	
$a^{-1}L_4$	a^*ba^*		$b^{-1}L_8$	b^*	
$b^{-1}L_4$	a^*	L_7	$a^{-1}L_9$	$\emptyset$	
$a^{-1}L_5$	a^*		$b^{-1}L_9$	$\emptyset$	

Exercise

Verify this table.

Quotients Example 5

Here is a very different scenario:

$$L = \{\, a^i b^i \mid i \geq 0 \,\} = \{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

This time there are infinitely many quotients.

$$\begin{aligned}
(a^k)^{-1}L &= \{\, a^i b^{i+k} \mid i \geq 0 \,\} && \\
(a^k b^l)^{-1}L &= \{b^{k-l}\} && 1 \leq l \leq k \\
(a^k b^l)^{-1}L &= \emptyset && l > k
\end{aligned}$$

This is no coincidence: the language L fails to be regular, unlike all the previous examples.

An Algorithm?

The tables from above suggest that for regular languages we can actually compute the quotients in a purely algebraic manner. We simple apply the algebraic rules from the lemma over and over, until no new quotients appear.

Is this really true?

Yes and no. In order for this to work, we need to be able to test whether two algebraic expressions are equivalent, whether they denote the same language.

This turns out to be decidable, but it is quite difficult: the problem is PSPACE-complete in general. We will find better algorithms based on finite state machines in a while.

Quotients and Machines

Suppose $\mathcal{A}$ is a DFA for a regular language L. Define the behavior of a state p to be

$$[\![p]\!] = \mathcal{L}(\mathcal{A}(p, F))$$

In other words, move the initial state to p but leave the automaton unchanged otherwise. In particular $[\![q_0]\!] = L$.

Proposition

For any word w, $[\![q_0 \cdot w]\!] = w^{-1}L$.

It follows immediately that every regular language has only finitely many quotients. In fact, the size of any DFA for the language is a bound on this number. The next result establishes the opposite direction: finitely many quotients implies regular.

The Quotient Machine

Suppose L is some language with a finite set of quotients. We can exploit $\mathcal{Q} = \mathcal{Q}(L)$ as the state set of a DFA for L.

$$\mathfrak{Q}_L = \langle \mathcal{Q}, \Sigma, \delta; q_0, F \rangle$$

where

$$\delta(K, a) = a^{-1}\, K$$

$$q_0 = L$$

$$F = \{\, K \in Q \mid \varepsilon \in K \,\}$$

A simple induction shows that $\delta(q_0, w) = w^{-1}L$, so this works as advertised. Since every quotient occurs only once in $\mathfrak{Q}_L$ there cannot be a smaller DFA for L.

Asymmetry

Left quotients are a powerful tool in the analysis of regular languages. Unfortunately, they are subject to an annoying asymmetry: write

$$\delta(L) = \text{ number of left quotients of } L$$

Claim: There can be an exponential gap between $\delta(L)$ and $\delta(L^{\mathsf{op}})$.

Direction should not matter at all in a way, yet the differences occur even for very simple languages: the go-to example are the kth position languages:

- $\delta(L_{a,k}) = k + 2$
- $\delta(L_{a,-k}) = 2^k$

Why?

For $L_{a,k} = \Sigma^{k-1} a \Sigma^\star$ the quotients are

$$\Sigma^{k-1} a \Sigma^\star, \Sigma^{k-2} a \Sigma^\star, \ldots, a \Sigma^\star, \Sigma^\star, \emptyset$$

But for $L_{a,-k} = \Sigma^\star a \Sigma^{k-1}$ the quotients are

$$\Sigma^\star + c_1 \Sigma^{k-1} + c_2 \Sigma^{k-2} + \ldots + c_k \qquad c_i = 0, 1$$

This follows easily from $a^{-1}L = L + \Sigma^{k-1}$ and $b^{-1}L = L$ for all $b \neq a$.

Generalizing Quotients

Given two languages K and L, we can naturally make sense out of

$$K^{-1}L = \{\, y \mid \exists\, x \in K\, (xy \in L) \,\}$$

In other words,

$$K^{-1}L = \bigcup_{x \in K} x^{-1}L$$

It follows that a language L is regular iff the number of language quotients is finite. We write

$$\partial(L) = \text{ number of left language quotients of } L$$

Clearly, $\delta(L) \leq \partial(L)$.

Example: Even/Even

Consider the language of all even/even strings over $\{a, b\}$.

Then the four word quotients are the languages

even/even   even/odd   odd/even   odd/odd

These are all pairwise disjoint, so the number of language quotients is 16.

Exercise

Figure out the number of language quotients for the complement of even/even.

Properties

- $K^{-1}0 = 0$
- $K^{-1}1 = \chi_K$.
- $K^{-1}a = \begin{cases} 0 & \text{if } a \notin K, \varepsilon \notin K \\ 1 & \text{if } a \in K, \varepsilon \notin K \\ a & \text{if } a \notin K, \varepsilon \in K \\ a+1 & \text{if } a \in K, \varepsilon \in K \end{cases}$
- $K^{-1}(L_1 + L_2) = K^{-1}L_1 + K^{-1}L_2$
- $K^{-1}(L_1L_2) = (K^{-1}L_1)L_2 + (L_1^{-1}K)^{-1}L_2$
- $K^{-1}L^\star = ((L^\star)^{-1}K)^{-1}L\, L^\star + \chi_K$

More Properties

- $0^{-1}L = 0$
- $w^{-1}L = \dots$
- $(K_1 + K_2)^{-1}L = K_1^{-1}L + K_2^{-1}L$
- $(K_1 K_2)^{-1}L = K_2^{-1}K_1^{-1}L$

Note that language quotients are not well-behaved with respect to other Boolean operations, though. For example, in general $\overline{K^{-1}L} \neq K^{-1}\overline{L}$.

Exercise

Prove all these properties. Establish counterexamples to all the missing claims regarding Boolean operations.

Conway Factorizations

Fix some language L once and for all.

Definition

A k-subfactorization of L (or subfactorization of order k) is a k-tuple of languages X_i, $1 \leq i \leq k$, such that

$$X_1 \cdot X_2 \cdot \ldots X_{k-1} \cdot X_k \subseteq L$$

For emphasis, we write $X_1{:}X_2{:}\ldots{:}X_k$ for a subfactorization.
A k-factorization is a k-subfactorization where every term is maximal.

Note that $\ldots{:}X{:}Y{:}\ldots$ is a subfactorization iff $\ldots{:}XY{:}\ldots$ is a subfactorization, albeit of order $k-1$. Alas, the corresponding claim for factorizations is wrong, in either direction.

We are mostly interested in the case $k = 2, 3$.

Factors

There is a natural partial order on subfactorizations by pointwise set inclusion $Y_1{:}Y_2{:}\ldots{:}Y_k \subseteq X_1{:}X_2{:}\ldots{:}X_k$ if $Y_i \subseteq X_i$ for all i. So a factorization is a maximal element in this order.

A factor is a term that appears in some place in some factorization. A left/right factor is one that appears at the left/right end of a factorization.

Question: For a regular language L, what can we say about its factors?

There is a surprisingly detailed answer, but we need to build up a few tools first.

Basics

Claim 1: Every subfactorization can be extended to a factorization. Maximal terms are preserved in the process.

Proof. For simplicity consider a 2-factorization $X{:}Y$. We can saturate, say, the left term via $X' = \bigcup\{\, Z \mid ZY \subseteq L \,\}$. Then $X'{:}Y$ is still a subfactorization, dominates $X{:}Y \subseteq X'{:}Y$, and X' is maximal. Repeat for Y. □

But note that the process does not commute: the final result depends on the saturation order. For example, for most L, we could extend $0{:}0$ to either $\Sigma^\star{:}0$ or $0{:}\Sigma^\star$. In fact, there are many other ways we can saturate the components by adding words in some fairly arbitrary manner.

Exercise

Figure out a general algorithm to extend $0{:}0$ to all possible factorizations.

More Basics

Claim 2: There is a one-one correspondence between all left factors and all right factors.

Proof. Suppose X is a left factor and let $X{:}Y$ be a corresponding factorization. Since Y is maximal there can be no other right factor matching X. The same argument works in the opposite direction, done. $\square$

We write $\rho(X)$ for the right factor corresponding to left factor X, and $\lambda(Y)$ for the inverse function. For example, for $L = a^\star \subseteq (a+b)^\star$ we have $\rho(0) = \Sigma^\star$, $\rho(a^\star) = a^\star$ and $\rho(\Sigma^\star) = 0$.

All Factors

Now saturate the middle term in $X{:}0{:}Y$, X and Y left/right factors, and write $Z = Z(X, Y)$ for the result. From Claim 1 we have that all factors occur as one of these $Z(X, Y)$.

Claim 3: The language L itself is a right factor $\rho(X')$ as well as a left factor $\lambda(Y')$. Moreover, all left factors are of the form $Z(X', Y)$, and all right factors are of the form $Z(X, Y')$. Lastly, $Z(X', Y') = L$.

Proof.

$1{:}L$ is a subfactorization and uniquely saturates to $X'{:}L$, so that $X' = \lambda(L)$. By symmetry, $Y' = \rho(L)$.

By our choice of X', $X'{:}\lambda(Y){:}Y$ is a subfactorization and even a factorization (check). The claim about enumerating left factors follows; right factors are analogous. Lastly, $X'{:}L{:}Y'$ is a factorization, and we are done. □

Regular Case

Theorem

The number of factors of L is finite iff L is regular. Moreover, the number of left/right factors is $\partial(\overline{L})$ in this case.

Proof. To see why, let $X{:}Y$ be a factorization of L. We have $Y = \bigcap_{w \in X} w^{-1}L$. To saturate the left term we choose X maximal so as to maintain the intersection. More precisely, by complementing we get

$$\overline{Y} = \bigcup_{w \in X} w^{-1}\overline{L} = X^{-1}\overline{L}$$

But L is regular iff the number of quotients (word or language) is finite.

$\square$

Careful, though: in general $\partial(L) \neq \partial(\overline{L})$, unlike with δ. It follows that there are at most $2^{\delta(L)}$ many left/right pairs.

Computation

Suppose $\mathcal{A}$ is the minimal DFA for L. As just mentioned, we need to determine all intersections

$$Y = \bigcap_{p \in P} [\![p]\!]$$

where $P \subseteq Q$. Let's call P critical if P produces Y in this manner, and P is maximal such. Note that P must actually be maximum (just take unions).

Given P critical for right factor Y we obtain the corresponding left factor $\lambda(Y)$ as

$$X = \mathcal{L}(\mathcal{A}(q_0, P))$$

Hence we can construct a list

$$X_1{:}Y_1, X_2{:}Y_2, \ldots, X_m{:}Y_m$$

of all left/right pairs where $Y_i = \rho(X_i)$.

More Computation

Suppose $U, V \subseteq Q$ are critical, and let X be the left factor for U, and Y the right factor for V. To determine $Z = Z(X, Y)$ note that

$$\begin{aligned} w \notin Z &\Leftrightarrow \exists\, x \in X, y \in Y\, (xwy \notin L) \\ &\Leftrightarrow \exists\, q \in U, y \in Y\, (q \cdot wy \notin F) \\ &\Leftrightarrow \exists\, q \in U\, (Y \not\subseteq [\![q \cdot w]\!]) \\ &\Leftrightarrow \exists\, q \in U\, (q \cdot w \notin V) \end{aligned}$$

But then $\overline{Z}$ is the language of $\mathcal{A}(U, \overline{V})$.

Together with the list of left/right pairs we now have a coordinate system and can organize the collection of all factors into a $m \times m$ matrix $\mathfrak{F}$ with entries $Z_{ij} = Z(X_i, Y_j)$.

Example: $a^\star b^\star c^\star$

The star-free language $L = a^\star b^\star c^\star$ has 5 left/right factors:

left	$\Sigma^\star$	L	$a^\star b^\star$	$a^\star$	0
right	0	$c^\star$	$b^\star c^\star$	L	$\Sigma^\star$

Factor matrix $\mathfrak{F} = (Z_{ij})$:

	0	$c^\star$	$b^\star c^\star$	L	$\Sigma^\star$
$\Sigma^\star$	$\Sigma^\star$	0	0	0	0
L	$\Sigma^\star$	$c^\star$	0	0	0
$a^\star b^\star$	$\Sigma^\star$	$b^\star c^\star$	$b^\star$	0	0
$a^\star$	$\Sigma^\star$	L	$a^\star b^\star$	$a^\star$	0
0	$\Sigma^\star$	$\Sigma^\star$	$\Sigma^\star$	$\Sigma^\star$	$\Sigma^\star$

The Factor Matrix

Theorem

Consider the $m \times m$ *factor matrix* $\mathfrak{F} = (Z_{ij})$. *Then*

- $Z_{ij} Z_{jk} \leq Z_{ik}$
- $X_1 X_2 \ldots X_s \leq L$ *iff* $X_j \leq Z_{i_{j\text{-}1}\, i_j}$ *for some* $i_j \in [m]$, $j \in [s]$, *where* $i_0 = \lambda(L)$ *and* $i_s = \rho(L)$.

Proof. By definition, $X_i Z_{ij} Y_j \leq L$, so that $X_i Z_{ij} \leq X_j$. Hence $X_i Z_{ij} Z_{jk} Y_k \leq X_j Z_{jk} Y_k \leq L$, and our claim follows.

It suffices to prove the binary case: $XY \leq Z_{ik}$ iff there is some j such that $X \leq Z_{ij}$ and $Y \leq Z_{jk}$.

To see this, note that $(X_i X)(Y Y_k) \leq L$, so that $X_i X \leq X_j$ and $Y Y_j \leq Y_j$ for some j. But then $X_i X Y_j \leq L$ and $X_J Y Y_k \leq L$, and the claim follows.

More

This may seem obvious, but we now have a proof that the factors of a factor are again factors of the original language. OK, a bit anticlimactic . . .

Recall that $\mathfrak{F}$ itself lives in another Kleene algebra and thus has a star. We have $\mathfrak{F}^\star = \mathfrak{F}$.

Exercise

Extract this information from the last theorem.

Reversal Symmetry

Back to our original complaint: the lack of invariance under string reversal.

Theorem (Conway)

Let L be a regular language. Then $\partial(L) = \partial(L^{\mathsf{op}})$.

Proof.

Consider all 2-factorizations $X{:}Y$ of $\overline{L}$.

As we have just seen, there are $\partial(L)$ choices for X.

By symmetry, there are $\partial(L^{\mathsf{op}})$ choices for Y.

But we already know that these two numbers agree.

$\square$

Left-To-Right

From the definition of a Turing machine, the read-only input tape can be scanned repeatedly and the tape head may move back and forth over it.

As it turns out, one can assume without loss of generality that the read head only moves from left to right only: at each step one symbol is scanned and then the head moves right and never returns.

Theorem (Rabin/Scott, Shepherdson)

Every decision problem solved by a constant space two-way machine can already be solved by a constant space one-way machine.

The original proof of this result is quite messy, see Rabin/Scott 59. Here is a sketch.

2-DFA

Right moves of the 2-DFA are easily simulated by the new DFA, so consider a left move. Say, the current configuration is xpa and $\delta(p, a) = (p, \mathsf{L})$. Then the machine enters the block x and we need to keep track of the state q it is in when it leaves the block to the right. Of course, this may never happen: the machine may have fallen off the left end of x, or may be stuck in an infinite loop. To deal with this issue, we augment Q by an additional state $\perp$. We also abuse $\perp$ to keep track of the state of the actual computation of the 2-DFA.

More formally, we define state vectors $V_x : Q_\perp \to Q_\perp$ for all non-empty words x as follows:

$$V_x(p) = \begin{cases} q & \text{if } w = ua,\ upa \vdash\ uaq,\ p \in Q \\ q & \text{if } w = ua,\ q_0 w \vdash\ wq,\ p = \perp \\ \perp & \text{otherwise.} \end{cases}$$

Simulation by DFA

It is not hard to check that these vectors have the property that $V_x = V_y$ implies $V_{xa} = V_{ya}$. Hence, they can be used to define a right semigroup action, giving rise to a one-way DFA. Let $\mathcal{V}$ be the set of all state vectors and add an extra initial state $\top$. Define a transition function γ on $\mathcal{V}_\top$ by

$$\begin{aligned}\gamma(\top, a) &= V_a \\ \gamma(V_x, a) &= V_{xa}\end{aligned}$$

and final states by

$$F' = \{\, V_x \mid V_x(\perp) \in F \,\}$$

If necessary, we can adjust to deal with ε.

Small 2-DFA

Consider the finite languages

$$L_n = \{\, \#ab^{e_1}ab^{e_2}a\dots ab^{e_n}c^k b^{e_k}\# \mid e_i, k \in [n] \,\}$$

There is a linear size 2-dfa for L_n, but every 1-dfa has exponentially many states.

Application: Roots

Consider a regular language $L \subseteq \Sigma^+$ and define

$$\text{root}(L) = \{\, x \in \Sigma^+ \mid x^+ \cap L \neq \emptyset \,\}$$

We want to show that root(L) is again regular. This can be done using a monoid automaton, but a very simple argument can be based on 2-DFAs.

Using endmarkers, we can build a 2-DFA that scans x and checks if it lies in L; if so, it accepts. Otherwise, it stores the current state q of the DFA for L, and rescans x, this time starting in state q. Rinse and repeat. If x is in root(L), this will lead to acceptance; otherwise, the 2-DFA is stuck in an infinite loop.

If you find this offensive, keep track of all states q seen so far at the end of a scan, and reject whenever a duplicate appears.

Monoid Automaton

Let $\mathcal{A}$ be a DFA. We can define an equivalent DFA $\mathcal{A}^{\text{sgr}}$ whose state set is the monoid of maps $Q \to Q$. The right action, initial and final states are given by

$$\begin{aligned} f \cdot a &= f \circ \delta_a \\ q_0 &= I \\ F' &= \{\, f \mid f(q_0) \in F \,\} \end{aligned}$$

This machine is useful for conceptual purposes such as establishing a link between finite state machines and monoids, but is obviously problematic from an algorithmic perspective (even if we restrict the state set to the monoid generated by the δ_a.

Roots

As and example, consider $L \subseteq \Sigma^+$ and define

$$\text{root}(L) = \{ x \in \Sigma^+ \mid x^+ \cap L \neq \emptyset \}$$

Proposition

root(L) *is regular whenever* L *is so regular.*

Proof.

Change the final states in the monoid automaton for L to be

$$F = \{ f : Q \to Q \mid \text{orb}^+(q_0; f) \cap F_{\mathcal{A}} \neq \emptyset \}$$

The new automaton accepts x iff there is some $k \geq 1$ such that $x^k \in L$, as required. $\square$

First Principle of FSM

Figure out what the darn states are.

Matrices

There are many ways one can associate a matrix with a finite state machine. Say, we are dealing with a DFA $\mathcal{A}$.

Probably the most obvious approach is to consider the transition function of $\mathcal{A}$ as a matrix $\Sigma^{Q\times\Sigma}$. This is useful from a data structure perspective, but not particularly interesting.

Much more useful is to consider square matrices $M^{Q\times Q}$ that live in some monoid, with matrix multiplication as the operation. In particular we can associate every input symbol to a Boolean matrix

$$\varphi : \Sigma \to \mathbf{2}^{Q\times Q}$$

giving rise to a monoid homomorphism. We have

$$x \in L \iff I \cdot \varphi(x) \cdot F = 1$$

where I and F are Boolean vectors indicating the initial and final states.