The Language, the Expression, and the (small) Automaton

Jacques Sakarovitch

LTCI, CNRS / ENST
46 rue Barrault, F-75634 Paris Cedex 13, France
sakarovitch@enst.fr

Abstract. This survey paper reviews the means that allow to go from one representation of the languages to the other and how, and to what extend, one can keep them small. Some emphasis is put on the comparison between the expressions that can be computed from a given automaton and on the construction of the derived term automaton of an expression.

1 A Plato's caverna

Formal language theory, especially that part which consists in the study of the so-called *regular* or *recognisable* languages, is a model instance of Plato's myth of the cavern. The real objects are the languages – or the power series – potentially *infinite* and what we, poor computer scientists bound to manipulate *finite* objects, can only see are the expressions that denote, or the automata that recognize them. Hopefully, these expressions and automata are fairly faithful descriptions of the languages (or of the series) they stand for and all the more effective that one can take advantage of this double light.

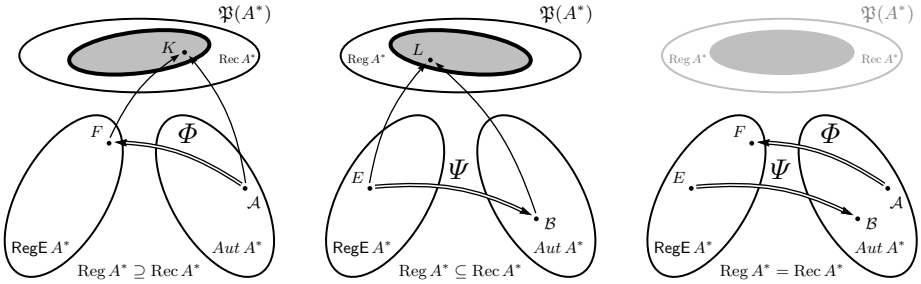


Fig. 1. The Φ and Ψ algorithms

It is the idea I have tried to illustrate with Figure 1 in the case of Kleene's Theorem. Kleene's Theorem states the equality of two classes of languages: the class of recognisable languages, that is those languages recognised by a finite automaton, and the class of regular languages, that is those languages denoted

by a regular expression. A closer look *at the proof* allows to argue that Kleene's Theorem is indeed the combination of two classes of algorithms: one that transform an automaton into an expression and one that build an automaton from an expression. In this setting, the real languages — or series — almost disappear: only exist their symbolic (and finitary) representations.

In this talk, mostly a survey, I review the means that allow to go from one representation of the languages to the other and how, and to what extend, one can keep them small.

The first section presents the classical methods of computing an expression from an automaton and of computing an automaton from an expression. We discuss the relationships between the different expressions obtained from a given automaton and the ways of reaching a compact one. In the second section, I classify the methods that build an automaton from an expression and describe with more details the one which is probably the lesser known: Antimirov's construction of *derived term automaton*.

As a conclusion, I mention the problem of finding an algorithm that is inverse to those which compute an expression from an automaton, hence taming the combinatorial explosion induced by the latter ones, and sketch a first attempt to solve it.

2 The Φ algorithms

We use mostly classical notation ([1, 2]). In particular we denote an automaton as $\mathcal{A} = \langle Q, A, E, I, T \rangle$ where I and T are subsets of the set Q of states, and E is the set of transitions labeled by letters of the alphabet A, or equivalently as $\mathcal{A} = \langle I, E, T \rangle$ where E is the square matrix of dimension Q whose entry (p, q) is the set of letters that label the transitions from p to q, and where I and T are Boolean vectors of dimension Q. The language accepted by $\mathcal{A}$ is denoted by $L(\mathcal{A})$ and with the latter notation, $L(\mathcal{A}) = I \cdot E^* \cdot T$.

A "Φ algorithms", computes an expression for $L(\mathcal{A})$ and thus amounts to compute expressions for the entries of the star of the matrix E. We shall consider this problem both from a theoretical and from an experimental point of view.

2.1 A theoretical point of view

There are (at least) four methods or algorithms for computing a regular expression that denotes $L(\mathcal{A})$:

1. Iterative computation of E^*: known as *McNaughton–Yamada algorithm* after their seminal paper ([3]) and probably the most popular among textbooks on automata theory. Called algorithm MNY here.
2. Direct computation of the entries of E^*: the so-called *state elimination method* ([4, 5]) looks more elementary and is indeed the easiest for hand computation as well as for computer implementation (*cf.* Figure 2).
3. Computation of $E^* \cdot T$ as a solution of a system of linear equations. Based on Arden's Lemma, it also allows to consider $E^* \cdot T$ as a fixed point.

4. Recursive computation of E^*: based on Arden's Lemma as well, this algorithm appeared first in Conway's book ([6]) conjugates mathematical elegance and computational inefficiency.

Fig. 2. One step in the state elimination method

The first three algorithms rely on a total order ω on Q, the fourth on a recursive division τ of the same set Q. All these algorithms, and for each algorithm all orders on Q will give by definition equivalent, but likely distinct, expressions. It is thus a natural problem that to compare these expressions; but this raise the question: 'what does it mean *to compare* expressions'? A possible answer — the one we choose here — consists in the characterisation of which of the *basic identities* are necessary to transform one into another. We thus first begin with a presentation of those identities which roughly follows that that Krob ([7]) gave of Conway's system ([6]).

Trivial and natural identities

$$\mathsf{E}+\mathsf{0}\equiv\mathsf{0}+\mathsf{E}\equiv\mathsf{E}\ ,\quad \mathsf{E}\cdot\mathsf{0}\equiv\mathsf{0}\cdot\mathsf{E}\equiv\mathsf{0}\ ,\quad \mathsf{E}\cdot\mathsf{1}\equiv\mathsf{1}\cdot\mathsf{E}\equiv\mathsf{E} \qquad (\mathbf{T})$$

$$(\mathsf{E}+\mathsf{F})+\mathsf{G}\equiv\mathsf{E}+(\mathsf{F}+\mathsf{G})\ ,\quad (\mathsf{E}\cdot\mathsf{F})\cdot\mathsf{G}\equiv\mathsf{E}\cdot(\mathsf{F}\cdot\mathsf{G}) \qquad (\mathbf{A})$$

$$\mathsf{E}\cdot(\mathsf{F}+\mathsf{G})\equiv\mathsf{E}\cdot\mathsf{F}+\mathsf{E}\cdot\mathsf{G}\ ,\quad (\mathsf{E}+\mathsf{F})\cdot\mathsf{G}\equiv\mathsf{E}\cdot\mathsf{G}+\mathsf{F}\cdot\mathsf{G} \qquad (\mathbf{D})$$

$$\mathsf{E}+\mathsf{F}\ \equiv\ \mathsf{F}+\mathsf{E} \qquad (\mathbf{C})$$

Aperiodic identities.

$$\mathsf{E}^*\ \equiv\ \mathsf{1}+\mathsf{E}\cdot\mathsf{E}^*\ ,\quad \mathsf{E}^*\ \equiv\ \mathsf{1}+\mathsf{E}^*\cdot\mathsf{E} \qquad (\mathbf{U})$$

$$(\mathsf{E}+\mathsf{F})^*\ \equiv\ \mathsf{E}^*\cdot(\mathsf{F}\cdot\mathsf{E}^*)^*\ ,\quad (\mathsf{E}+\mathsf{F})^*\ \equiv\ (\mathsf{E}^*\cdot\mathsf{F})^*\cdot\mathsf{E}^* \qquad (\mathbf{S})$$

$$(\mathsf{E}\cdot\mathsf{F})^*\ \equiv\ \mathsf{1}+\mathsf{E}\cdot(\mathsf{F}\cdot\mathsf{E})^*\cdot\mathsf{F} \qquad (\mathbf{P})$$

Cyclic identities.

$$\mathsf{E}^*\ \equiv\ \mathsf{E}^{<\mathsf{n}}\cdot(\mathsf{E}^{\mathsf{n}})^* \qquad (\mathbf{Z})_n$$

Idempotency identities.

$$\mathsf{E}+\mathsf{E}\ \equiv\ \mathsf{E} \qquad (\mathbf{I}) \qquad\qquad (\mathsf{E}^*)^*\ \equiv\ \mathsf{E}^* \qquad (\mathbf{J})$$

The state elimination and equation solution methods

Proposition 1 ([8]). *The state elimination method and the solution (by Gaussian elimination) of a system of linear equations taken from an automaton give the same regular expression (assuming that the same order in elimination is used in both cases).*

Proof. For p and q in Q, the *set of words* which are the label of a computation which goes from p to a final state of $\mathcal{A}$ is written: $L_p = \{f \mid \exists t \in T \quad p \xrightarrow[\mathcal{A}]{f} t\}$ and we write $E_{p,q}$ for the *set of labels of transitions* which go from p to q and the symbol $\delta_{p,R}$ for a subset R of Q, which is 1_{A^*} if p is in R and $\emptyset$ if not. The system of equations associated with $\mathcal{A}$ is written:

$$L(\mathcal{A}) = \sum_{p \in I} L_p \qquad = \sum_{p \in Q} \delta_{p,I}\, L_p \tag{1}$$

$$\forall p \in Q \qquad L_p = \sum_{q \in Q} E_{p,q}\, L_q + \delta_{p,T} \tag{2}$$

After the elimination of a certain number of unknowns L_p – we write Q' for the set of indices of those which have not been eliminated – we obtain a system of the form:

$$L(\mathcal{A}) = \sum_{p \in Q'} G_p\, L_p + H \tag{3}$$

$$\forall p \in Q' \qquad L_p = \sum_{q \in Q'} F_{p,q}\, L_q + K_p \tag{4}$$

We can make a generalised automaton $\mathcal{B}'$ corresponding to such a system, whose set of states is $Q' \cup \{i, t\}$, where i and t do not belong to Q', and such that, for all p and q in Q': (i) the transition from p to q is labelled $F_{p,q}$; (ii) the transition from p to t is labelled K_p; (iii) the transition from i to p is labelled G_p; and (iv) the transition from i to t is labelled H.
Note that this definition applied to the system (1)–(2) characterises the automaton constructed in the first phase of the state elimination method applied to $\mathcal{A}$.

The elimination in the system (3)–(4) of the unknown L_p by substitutions and the application of Arden's Lemma give the system:

$$L(\mathcal{A}) = \sum_{r \in Q' \setminus p} \left[G_r + G_p\, F_{p,p}^* F_{p,r}\right]\, L_r + \left[H + G_p\, F_{p,p}^* K_p\right] \tag{5}$$

$$\forall r \in Q' \setminus p \quad L_r = \sum_{q \in Q' \setminus p} \left[F_{r,q} + F_{r,p}\, F_{p,p}^* F_{r,q}\right]\, L_q + \left[K_r + F_{r,p}\, F_{p,p}^* K_p\right] \tag{6}$$

whose coefficients are exactly the transition labels of the generalised automaton obtained by removing the state p from $\mathcal{B}'$.

Thus, since the starting points correspond and since each step maintains the correspondence, the expression obtained for $L(\mathcal{A})$ by the state elimination

method is the same as that obtained by the solution of the system (1)–(2). More precisely, we can say that the state elimination method reproduces in the automaton $\mathcal{A}$ the computations corresponding to the solution of the system.

The state elimination and MNY algorithms, identical orders

The order ω fixes the operation of the state elimination method whose result is a rational expression over A^*, written[1] $\mathsf{E}_{BMC}(\mathcal{A},\omega)$. For greater precision, we write the result of this algorithm $\mathsf{E}_{BMC}(\mathcal{A},\omega,(p,q))$ when we take p as the initial state and q as the final state.

On the other hand, we will write $\mathsf{M}_{MNY}(\mathcal{A},\omega)$ for the matrix of rational expressions obtained when we apply the McNaughton–Yamada algorithm to the automaton $\mathcal{A}$ whose states are ordered by ω. It then follows that:

Proposition 2 ([8]). *Let $\mathcal{A} = \langle\, Q, A, E, I, T \,\rangle$ an automaton over A^*. For every (total) order ω on Q and all p and q in Q, we have:*

$$(\mathbf{U}) \vdash \qquad [\mathsf{M}_{MNY}(\mathcal{A},\omega)]_{p,q} \quad \equiv \quad \mathsf{E}_{BMC}(\mathcal{A},\omega,(p,q)) \ .$$

Proof. To prove this result we will show a correspondence between the operations performed by the two algorithms. The difficulty, if it can be called that, is that we have to compare two objects whose form and mode of construction are rather different: on one hand a $Q\times Q$ matrix obtained by successive transformations, from which we choose one entry, and on the other an expression obtained by repeated modification of an automaton, hence of a matrix, but one whose size decreases at each step.

In the following, $\mathcal{A}$ and ω are fixed and remain implicit. The automaton $\mathcal{A}$ has n states, identified with the integers from 1 to n; the two algorithms perform n steps starting in a situation called 'step 0', the kth step of the state elimination method consisting of the removal of state k, and that of algorithm MNY consisting of calculating the labels of paths that do not include nodes (strictly) greater than k. We write:

$$\mathsf{E}^{(k)}(r,s)$$

for the label of the transition from r to s in the automaton obtained from $\mathcal{A}$ (and ω) at the kth step of the state elimination method; necessarily, in this notation, $k+1 \leqslant r$ and $k+1 \leqslant s$ (abbreviated to $k+1 \leqslant r,s$). We write:

$$\mathsf{M}^{(k)}_{r,s}$$

for the entry r,s of the $n\times n$ matrix computed by the kth step of algorithm MNY. At step 0, the automaton $\mathcal{A}$ has not been modified and we have:

$$\forall r,s\,,\ 1 \leqslant r,s \leqslant n \qquad \mathsf{M}^{(0)}_{r,s} = \mathsf{E}^{(0)}(r,s)\,, \tag{7}$$

[1] A reminder that this algorithm is due to J. Brzozowski and E. McCluskey ([9]).

which will be the base case of the inductions to come. Algorithm MNY is written:

$$\forall k\,,\ 0 < k \leqslant n\,,\ \forall r, s\,,\ 1 \leqslant r, s \leqslant n$$
$$\mathsf{M}^{(k)}_{r,s} = \mathsf{M}^{(k-1)}_{r,s} + \mathsf{M}^{(k-1)}_{r,k} \cdot [\mathsf{M}^{(k-1)}_{k,k}]^* \cdot \mathsf{M}^{(k-1)}_{k,s} \quad . \tag{8}$$

The state elimination algorithm is written:

$$\forall k\,,\ 0 < k \leqslant n\,,\ \forall r, s\,,\ k < r, s \leqslant n$$
$$\mathsf{E}^{(k)}(r,s) = \mathsf{E}^{(k-1)}(r,s) + \mathsf{E}^{(k-1)}(r,k) \cdot [\mathsf{E}^{(k-1)}(k,k)]^* \cdot \mathsf{E}^{(k-1)}(k,s) \tag{9}$$

Hence we conclude, for given r and s and by induction on k:

$$\forall r, s\,,\ 1 \leqslant r, s \leqslant n\,,\ \forall k\,,\ 0 \leqslant k < \min(r,s) \qquad \mathsf{M}^{(k)}_{r,s} = \mathsf{E}^{(k)}(r,s) \tag{10}$$

We see in fact (as there is even so something to see) that if $k < \min(r,s)$ then all integer triples (l,u,v) such that $\mathsf{M}^{(l)}_{u,v}$ occurs in the computation of $\mathsf{M}^{(k)}_{r,s}$ by the (recursive) use of (8), are such that $l < \min(u,v)$.

Suppose now that we have p and q, also fixed, such that $1 \leqslant p < q \leqslant n$ (the other cases are dealt with similarly). We call the initial and final states added to $\mathcal{A}$ in the first phase of the state elimination method i and t respectively; i and t are not integers between 1 and n. The transition from i to p and that from q to t are labelled 1_{A^*}. Now let us consider step p of each algorithm. For every state s, $p < s$, $\mathsf{M}^{(p)}_{p,s}$ is given by (8):

$$\mathsf{M}^{(p)}_{p,s} = \mathsf{M}^{(p-1)}_{p,s} + \mathsf{M}^{(p-1)}_{p,p} \cdot [\mathsf{M}^{(p-1)}_{p,p}]^* \cdot \mathsf{M}^{(p-1)}_{p,s}$$

and $\mathsf{E}^{(p)}(i,s)$ by:

$$\mathsf{E}^{(p)}(i,s) = [\mathsf{E}^{(p-1)}(p,p)]^* \cdot \mathsf{E}^{(p-1)}(p,s)$$

and hence, by (10):

$$\forall s\,,\ p < s \leqslant n \qquad (\mathbf{U}) \vdash \qquad \mathsf{M}^{(p)}_{p,s} \equiv \mathsf{E}^{(p)}(i,s) \quad . \tag{11}$$

Next we consider the steps following p (and row p of the matrices $\mathsf{M}^{(k)}$). For all k, $p < k$, and all s, $k < s \leqslant n$, $\mathsf{M}^{(k)}_{p,s}$ is always computed by (8) and $\mathsf{E}^{(k)}(i,s)$ by:

$$\mathsf{E}^{(k)}(i,s) = \mathsf{E}^{(k-1)}(i,s) + \mathsf{E}^{(k-1)}(i,k) \cdot [\mathsf{E}^{(k-1)}(k,k)]^* \cdot \mathsf{E}^{(k-1)}(k,s) \quad . \tag{12}$$

From (11), and based on an observation analogous to the previous one, we conclude from the term-by-term correspondence of (8) and (12) that:

$$\forall k\,,\ p < k\,,\ \forall s\,,\ p < s \leqslant n \qquad (\mathbf{U}) \vdash \qquad \mathsf{M}^{(k)}_{p,s} \equiv \mathsf{E}^{(k)}(i,s) \quad . \tag{13}$$

The analysis of step q gives a similar, and symmetric, result to that which we have just obtained from the analysis of step p: for all r, $q < r$, we have:

$$\mathsf{M}^{(q)}_{r,q} = \mathsf{M}^{(q-1)}_{r,q} + \mathsf{M}^{(q-1)}_{r,q} \cdot [\mathsf{M}^{(q-1)}_{q,q}]^* \cdot \mathsf{M}^{(q-1)}_{q,q}$$

and

$$\mathsf{E}^{(q)}(r,t) = \mathsf{E}^{(q-1)}(r,q) \cdot [\mathsf{E}^{(q-1)}(q,q)]^*$$

and hence

$$\forall r\,,\ q < r \leqslant n \qquad (\mathbf{U}) \vdash \qquad \mathsf{M}^{(q)}_{r,q} \equiv \mathsf{E}^{(q)}(r,t)\ . \tag{14}$$

The steps following q give rise to an equation symmetric to (13) (for column q of the matrices $\mathsf{M}^{(k)}$):

$$\forall k\,,\ q < k\,,\ \forall r\,,\ q < r \leqslant n \qquad (\mathbf{U}) \vdash \qquad \mathsf{M}^{(k)}_{r,q} \equiv \mathsf{E}^{(k)}(r,t)\ . \tag{15}$$

Finally, from:

$$\mathsf{M}^{(k)}_{p,q} = \mathsf{M}^{(k-1)}_{p,q} + \mathsf{M}^{(k-1)}_{p,k} \cdot [\mathsf{M}^{(k-1)}_{k,k}]^* \cdot \mathsf{M}^{(k-1)}_{k,q}$$

$$\text{and} \qquad \mathsf{E}^{(k)}(i,t) = \mathsf{E}^{(k-1)}(i,t) + \mathsf{E}^{(k-1)}(i,k) \cdot [\mathsf{E}^{(k-1)}(k,k)]^* \cdot \mathsf{E}^{(k-1)}(k,t)$$

Equations (10), (13) and (15) together allow us to conclude, by induction on k, that:

$$\forall k\,,\ q \leqslant k \leqslant n \qquad (\mathbf{U}) \vdash \qquad \mathsf{M}^{(k)}_{p,q} \equiv \mathsf{E}^{(k)}(i,t)\ . \tag{16}$$

When we reach $k = n$ in this equation we obtain the identity we want.

The state elimination and MNY algorithms, distinct orders

Having compared the state elimination and MNYalgorithms under the same order, that is the same execution conditions, we can compare the results of these algorithms for different execution conditions.

Theorem 1 (Conway [6], Krob [7]). *Let $\mathcal{A} = \langle\, Q, A, E, I, T \,\rangle$ be an automaton over A^*. The expressions denoting $L(\mathcal{A})$ computed by the McNaughton–Yamada algorithm, like those computed by the state elimination method or the solution of a system of equations, are all equivalent modulo* (**S**) *and* (**P**)*, i.e., for all orders ω and ω' on Q and all p and q in Q:*

$$(\mathbf{S}) \wedge (\mathbf{P}) \quad \vdash \quad [\mathsf{M}_{MNY}(\mathcal{A},\omega)]_{p,q} \quad \equiv \quad [\mathsf{M}_{MNY}(\mathcal{A},\omega')]_{p,q}\ ,$$

$$(\mathbf{S}) \wedge (\mathbf{P}) \quad \vdash \quad \mathsf{E}_{BMC}(\mathcal{A},\omega,(p,q)) \quad \equiv \quad \mathsf{E}_{BMC}(\mathcal{A},\omega',(p,q))\ .$$

Proof. The previous proposition allows us to show the property for expressions computed by the state elimination method, which is easier to deal with (remembering that (**P**) 'contains' (**U**)). Furthermore, we can go from an order ω to any other order ω', a permutation of Q, by a series of transpositions.

We therefore arrive at the situation illustrated in Figure 3 (left) and need to show that the expressions obtained by the state elimination method when we first remove the state r and then r' are equivalent to those obtained from removing first r' and then r, modulo (**S**) $\wedge$ (**P**).

The removal of state r gives the expressions in Figure 3 (right). The removal of state r' gives the expression:

$$\mathsf{E} = K\,L^*H + (K\,L^*G + K')\,[G'\,L^*G + L']^*\,(G'\,L^*H + H')\ ,$$

which using (**S**) (and the natural identities) becomes:

$$\begin{aligned}\mathsf{E} \equiv\ & K\,L^*H + K\,L^*G\,\left[L'^*G'\,L^*G\right]^*\,L'^*G'\,L^*H \\ & + K'\,\left[L'^*G'\,L^*G\right]^*\,L'^*G'\,L^*H + K\,L^*G\,\left[L'^*G'\,L^*G\right]^*\,L'^*H' \\ & + K'\,\left[L'^*G'\,L^*G\right]^*\,L'^*H'\,.\end{aligned}$$

We write:

$$K'\,\left[L'^*G'\,L^*G\right]^*\,L'^*H' \equiv K'\,L'^*H' + K'\,L'^*G'\,L^*\,\left[G\,L'^*G'\,L^*\right]^*\,G\,L'^*H'$$

by using (**P**) then, by 'switching the brackets' (using the identity $(\mathsf{XY})^*\mathsf{X} \equiv \mathsf{X}(\mathsf{YX})^*$ which is also a consequence of (**P**)), we obtain:

$$\begin{aligned}\mathsf{E} \equiv\ & K\,L^*H \\ & + K\,L^*G\,\left[L'^*G'\,L^*G\right]^*\,L'^*G'\,L^*H + K'\,L'^*G'\,\left[L^*G\,L'^*G'\right]^*\,L^*H \\ & + K\,L^*G\,\left[L'^*G'\,L^*G\right]^*\,L'^*H' + K'\,L'^*G'\,\left[L^*G\,L'^*G'\right]^*\,L^*G\,L'^*H' \\ & + K'\,L'^*H'\end{aligned}$$

an expression that is perfectly symmetric in the letters with and without ticks, which shows that we would have obtained the same result if we had started by removing r' then r.

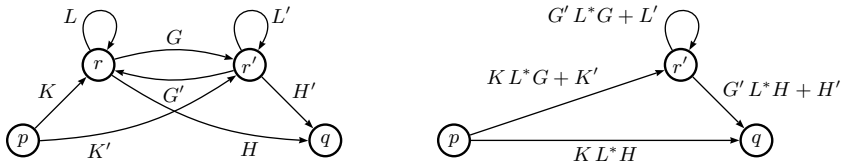


Fig. 3. First step of two in the state elimination method

Remark 1. It is known that the Φ-algorithms described above are valid for automata with multiplicity. It is thus not surprising that the idempotency identities are not used to pass from an expression to another one. On the other hand, it is also known ([6]) that an infinite number of identities (among which the cyclic identities $(\mathbf{Z})_n$ for all *prime numbers* n) are necessary to derive all possible equivalence among epressions. Taking this into account, the above results show that all expressions computed from a given automaton can be considered as 'close' since only the two identities (**S**) and (**P**) are necessary to derive one from another.

The state elimination and the recursive methods Finally, it remains to compare the matrices obtained by the algorithm MNY and the recursive algorithm. A simple two state automaton is sufficient for observing that there is no hope for a global comparison of the entries of the two matrices. We can however state the following conjecture.

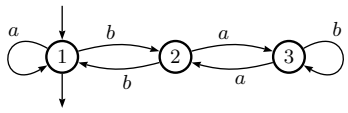
Conjecture 1. For every recursive division τ of Q and for every pair (p, q) of states, there exists an ordering ω' of Q such that

$$(\mathbf{U}) \quad \vdash \quad [\mathsf{C}(\tau)]_{p,q} \quad \equiv \quad \mathsf{E}(\omega', p, q) \ .$$

2.2 An experimental point of view

It is easily seen that the size of a regular expression E computed from an automaton $\mathcal{A}$ may be exponential in the number of states of $\mathcal{A}$. A complete graph shows that this combinatorial explosion is unavoidable.

But most of the interesting automata are not complete graph. Basic examples show how different the size of expressions computed from a same automaton can be: in Figure 4, E_1 is obtained by eliminating the states in the order 1–2–3 whereas E_2 is obtained with the reverse order 3–2–1.



$$\mathsf{E}_1 = a^* + a^*b(ba^*b)^*ba^* + a^*b(ba^*b)^*a(b + a(ba^*b)^*a)^*a(ba^*b)^*ba^*$$
$$\mathsf{E}_2 = (a + b(ab^*a)^*b)^*$$

Fig. 4. Two results of the state elimination method

Finding the ordering of states that yields the shortest expression for a given automaton is probably a hard combinatorial problem. On the other hand, it is not too difficult to design heuristics which do not imply heavy computations and prove to be pretty efficient.

In order to create as few transitions as possible at a given step (*cf.* Figure 2), one associates to every state q an index which is the product of the in-degree of q by its out-degree (once the possible loop on q is discarded); one then choose to eliminate among those states with smallest index a state without loop, if any; the index is then recomputed at each step.

This rather naive heuristic had been implemented in VAUCANSON ([10]). Delgado and Morais ([11]) have proposed a heuristic which is based on the same principle, but in which the length of the expressions that label the transitions is also taken into account in the computation of the index. This other heuristic has also been implemented in the newer version of VAUCANSON ([12]). First experiments show that it might be better (on a first set of "random" automata, it outperforms the naive one in 55% of the cases). More experiments on much larger sets of automata need certainly to be done: *the proof of a heuristic is in the computing.*

3 The Ψ algorithms

We call "Ψ algorithm" an algorithm that is given a regular expression E and computes an automaton which accepts the language denoted by E. As for the Φ algorithms, there is no much mystery left in this question. But not all aspects are equally well-known.

3.1 A theoretical point of view

Although there are numerous ways to present them, there are *two* main distinct constructions of an automaton from a regular expression: the *standard automaton* and the *derived term automaton*. Automata are compared via *morphisms*.

The standard automaton We say that an automaton is *standard* if it has only one initial state and if this initial state is not the end of any transition (and if the automaton is accessible). We call *standard automaton* of an expression E *the* automaton $\mathcal{S}_{\mathsf{E}}$ build by induction on the depth of E, starting from the (unique possible) standard automata for $\mathsf{0}$, $\mathsf{1}$, and every letter a in A, and with the "natural" constructions for the union, product and star: *cf.* Figures 5 and 6. Of course, any standard automaton is not, in general, the standard automaton of an expression.

Let us denote by $\ell(\mathsf{E})$ the *literal length* of the expression E — that is the number of occurrences of letters in E.

Proposition 3 (Glushkov [13]). *The standard automaton $\mathcal{S}_{\mathsf{E}}$ of the expression E has $\ell(\mathsf{E}) + 1$ states.*

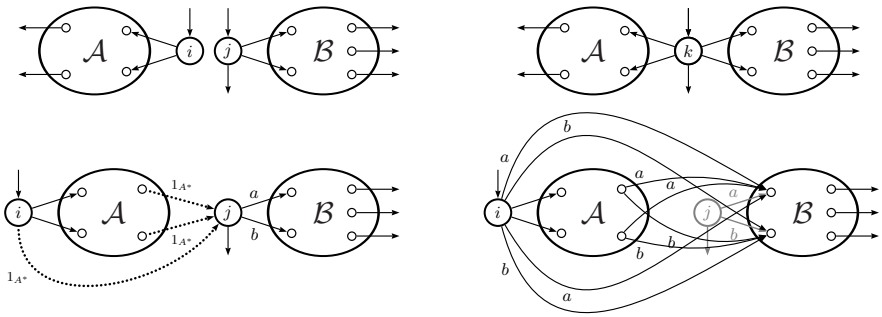


Fig. 5. Construction of the standard automaton for the union and the product

The 'standard automaton of an expression' is usually attributed to Glushkov [13] and hence often called *Glushkov's automaton*. It is also called *position* automaton of E as the original method of construction somehow starts from the occurrences of letters in E, taken as states, and then computes the transitions

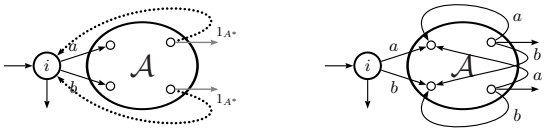


Fig. 6. Construction of the standard automaton for the star

— also by induction on the depth of E. A characteristic feature of $\mathcal{S}_{\mathsf{E}}$ is that it is *small* in terms of the 'input' E: linear for the states, quadratic for the transitions and it is so because it is *non deterministic*. In [3], McNaughton and Yamada already had the idea of using the positions of letters in the expression in order to define an automaton but they computed directly[2] its determinised version and thus lost any property on the size of the result. The mode of construction given here is adapted from [14]; it is well suited to the generalisation to automata with multiplicity ([14, 8, 15]).

Another method for building an automaton from an expression was given by Thompson ([16]). It amounts to recursive connection via spontaneous transitions (*i.e.* ε-moves) of 'atomic' automata that recognise letters and it was designed for a direct array implementation. It is folklore that the *backward closure* (*i.e.* suppression of spontaneous transitions by following first the spontaneous transitions and then a transition labeled with a letter) in the Thompson's automaton of E yields the standard automaton of E. Hence the former can be seen as an 'extended version' of the latter and falls in the same category.

The derived term automaton A second class of algorithms is based on the definition of the *derivation of an expression*. First introduced by Brzozowski [17], the definition of derivation has been slightly, but smartly, modified by Antimirov [18] and yields a non deterministic automaton $\mathcal{A}_{\mathsf{E}}$ which we propose to call the *derived term automaton* of the expression E. This automaton $\mathcal{A}_{\mathsf{E}}$ is smaller than or equal to the standard automaton $\mathcal{S}_{\mathsf{E}}$. The automaton of derived expressions computed in [17] is the determinised automaton of $\mathcal{A}_{\mathsf{E}}$.

An algebraic characterization of regular languages is that every regular language has a finite number of left quotients. The purpose of "Brozozowski" derivatives was to lift that characterization at the level of expressions [17]. Antimirov "partial derivatives" achieve the same lifting in an indirect but more efficient way. To an expression E that denotes a language L is associated a finite set $\mathcal{T}$ of expressions — which we call *derived terms* of E — such that any left quotient of L is a *union* of some of the languages denoted by the expressions in $\mathcal{T}$ [18].

The notion of derived terms is indeed better understood when expressed in the larger framework of power series — languages being series with coefficients in the Boolean semiring — and of expressions *with multiplicity* (*cf.* [15]). A series s is rational — *i.e.* denoted by a regular expression E — iff it is contained in a *finitely generated* module (of series) U which is *closed under left quotient*. The

[2] Probably because in those early times, an automaton *had to be* deterministic.

derived terms of E are then expressions that denote a *set of generators* of U. The following definitions give a procedure for computing the derived terms of an expression.

Definition 1 (Brozozowski–Antimirov [18]). *Let* E *be a regular expression on* A *and let* a *be a letter in* A. *The* $\mathbb{B}$-derivative[3] *of* E *with respect to* a, *denoted* $\frac{\partial}{\partial a}\mathsf{E}$, *is* a set of regular expressions on A, *recursively defined by:*

$$\frac{\partial}{\partial a}\mathsf{0} = \frac{\partial}{\partial a}\mathsf{1} = \emptyset\,,$$

$$\forall a, b \in A \quad \frac{\partial}{\partial a} b = \begin{cases} \{\mathsf{1}\} & \text{if} \quad b = a \\ \emptyset & \text{otherwise} \end{cases}$$

$$\frac{\partial}{\partial a}(\mathsf{E}{+}\mathsf{F}) = \frac{\partial}{\partial a}\mathsf{E} \cup \frac{\partial}{\partial a}\mathsf{F} \tag{17}$$

$$\frac{\partial}{\partial a}(\mathsf{E}\cdot\mathsf{F}) = \left[\frac{\partial}{\partial a}\mathsf{E}\right]\cdot\mathsf{F} \cup \mathsf{c}(\mathsf{E})\,\frac{\partial}{\partial a}\mathsf{F} \tag{18}$$

$$\frac{\partial}{\partial a}(\mathsf{E}^*) = \left[\frac{\partial}{\partial a}\mathsf{E}\right]\cdot\mathsf{E}^* \tag{19}$$

The induction implied by (17 – 19) should be interpreted by distributing derivation and product over union:

$$\frac{\partial}{\partial a}\left[\bigcup_{i\in I}\mathsf{E}_i\right] = \bigcup_{i\in I}\frac{\partial}{\partial a}\mathsf{E}_i\,, \qquad \left[\bigcup_{i\in I}\mathsf{E}_i\right]\cdot\mathsf{F} = \bigcup_{i\in I}(\mathsf{E}_i\cdot\mathsf{F})\,.$$

Definition 2. *Let* E *be a regular expression on* A *and* g *a non empty word of* A^*, *i.e.* $g = f\,a$ *with* a *in* A. *The* $\mathbb{B}$-derivative *of* E *with respect to* g, *denoted* $\frac{\partial}{\partial g}\mathsf{E}$, *is* the set of regular expressions on A, *recursively defined by formulae (17) – (19) and by:*

$$\forall f \in A^+\,,\ \forall a \in A \quad \frac{\partial}{\partial f a}\mathsf{E} = \frac{\partial}{\partial a}\left(\frac{\partial}{\partial f}\mathsf{E}\right). \tag{20}$$

We shall call derived term *of* E *the expression* E *itself or any of the expressions which belongs to a set* $\frac{\partial}{\partial g}\mathsf{E}$ *for some* g *in* A^+.

Theorem 2 (Antimirov [18]). *The number of derived terms of an expression* E *is finite and smaller than or equal to* $\ell(\mathsf{E}) + 1$.

Remark 2. Contrary to the derivation defined by Brzozowski [17], the result of the $\mathbb{B}$-derivation of an expression is *not one* expression but a *set of* expressions. As a result, it overcomes another drawback of its predecessor. The number of

[3] We call it "$\mathbb{B}$-derivative" and not simply "derivative" for two reasons. First in order to avoid confusion with the derivation defined by Brzozowski, and second because the formulae depend on the semiring of multiplicities and can be defined for other semirings (*cf.* [15]).

Brzozowski derivatives of an expression is not finite directly but only modulo the identities (**A**), (**C**) and (**I**) described above. The computation of the derived terms does not involve any identity.

Definition 3. *The* derived term automaton *of an expression* E *is the finite automaton* $\mathcal{A}_{\mathsf{E}}$ *whose states are the derived terms of* E *and whose transitions are defined by:*
(i) *if* K *and* K' *are derived terms of* E *and if* a *is a letter of* A, $(\mathsf{K}, a, \mathsf{K}')$ *is a transition of* $\mathcal{A}_{\mathsf{E}}$ *if and only if* K' *belongs to* $\frac{\partial}{\partial a}\mathsf{K}$*;*
(ii) *the initial state of* $\mathcal{A}_{\mathsf{E}}$ *is* E*;*
(iii) *a derived term* K *is a final state of* $\mathcal{A}_{\mathsf{E}}$ *if and only if* $\mathsf{c}(\mathsf{K}) = 1$*;*

Figure 7 shows the derived terms of the expression E_1 computed at Figure 4 and the corresponding derived term automaton.

We write $\mathsf{E}_1 = a^* + a^*b\,\mathsf{H}_1 + a^*b\,\mathsf{F}_1\,\mathsf{G}_1\,\mathsf{H}_1$ with $\mathsf{H}_1 = (b\,a^*b)^*b\,a^*$, $\mathsf{F}_1 = (b\,a^*b)^*a$, and $\mathsf{G}_1 = (b + a\,(b\,a^*b)^*a)^*a$.

The successive derivations of E_1 with respect to a and b give 7 derived terms: E_1 itself, a^* , H_1 , $\mathsf{X}_1 = a^*b\,\mathsf{H}_1$, $\mathsf{Y}_1 = a^*b\,\mathsf{F}_1\,\mathsf{G}_1\,\mathsf{H}_1$, $\mathsf{Z}_1 = \mathsf{F}_1\,\mathsf{G}_1\,\mathsf{H}_1$, and $\mathsf{T}_1 = \mathsf{G}_1\,\mathsf{H}_1$.

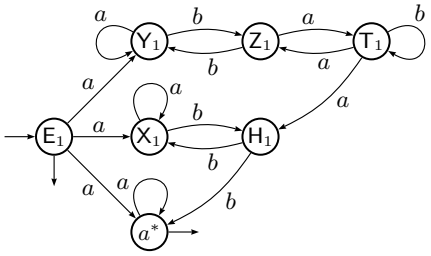


Fig. 7. The derived terms of E_1 and its derived term automaton

The two classes of algorithms are not without relationships between them. A first one was given by Berry–Sethi who showed that the Brzozowski derivation applied on a " linearized" version of an expression gives the standard automaton of that expression [19, 20]. A more interesting one is established by means of morphisms of automata that we should define first.

Morphisms of automata Let $\mathcal{A} = \langle\, Q, A, E, I, T \,\rangle$ and $\mathcal{B} = \langle\, R, A, F, J, U \,\rangle$ be two $\mathbb{B}$-automata. A (surjective) map $\varphi\colon Q \longrightarrow R$ induces (or *is*) a morphism from $\mathcal{A}$ onto $\mathcal{B}$ if $(p, a, q) \in E$ implies $(\varphi(p), a, \varphi(q)) \in F$ and this morphism is a *($\mathbb{B}$)-quotient* if moreover $(r, a, s) \in F$ and $p \in \varphi^{-1}(r)$ implies that there exists q in $\varphi^{-1}(s)$ such that $(p, a, q) \in E$. *Every automaton has a unique minimal quotient.*

Theorem 3 (Champarnaud–Ziadi [21]). *For any expression* E*, the derived term automaton* $\mathcal{A}_{\mathsf{E}}$ *is a quotient of the standard automaton* $\mathcal{S}_{\mathsf{E}}$*.*

This result implies in particular the bound of Theorem 2 on the number of derived terms. It is to be noted also that if the derived term automaton is a quotient of the standard automaton, *it is not* its minimal quotient. Theorem 3

has been generalised to expressions with multiplicity but this generalisation requires special care in the definition of the derived terms in the case where the multiplicity semiring is not a positive semiring ([15]).

3.2 An experimental point of view

The effective computation of the standard automaton of an expression has been the subject of many works. If the actual efficiency of the computation depends unpon the implementation, it is known that the construction of $\mathcal{S}_{\mathsf{E}}$ is of quadratic complexity (with respect to $\ell(\mathsf{E})$) ([22]).

The determination of the complexity of the computation of the derived term automaton if an expression E is more difficult. The key property, proven in [21], is that every derived term of E is a product of subexpressions of E.

Proposition 4 (Champarnaud–Ziadi [21]). *For every expression* E*, the derived term automaton* $\mathcal{A}_{\mathsf{E}}$ *can be computed with a quadratic complexity (with respect to* $\ell(\mathsf{E})$*).*

The Champarnaud–Ziadi algorithm has been transformed in order to be valid for automata with multiplicity and it has been implemented in VAUCANSON (*cf.* [12] in this volume).

It appears that $\mathcal{A}_{\mathsf{E}}$ is particularly 'economical' — sizewize and by comparison with $\mathcal{S}_{\mathsf{E}}$ — when E is obtained by a Φ algorithm from a finite automaton. We come back to his fact in the conclusion. However, it seems that, e*ven in this case*, the computation of $\mathcal{S}_{\mathsf{E}}$ followed by a quotient is far more efficient than the direct computation of $\mathcal{A}_{\mathsf{E}}$. Other constructions have been proposed recently that yield automata which are smaller than the standard automaton ([23–25]). Their proper relationships with the derived term automaton, and the efficiency of their computation are still to be worked out by extensive experimentations (*cf.* [26]).

4 Can expressions and automata code for each other?

We have seen that a Φ-algorithm is likely to generate, from an automaton $\mathcal{A}$, an expression with a literal length which is exponential in the number of states of $\mathcal{A}$ and that a Ψ-algorithm is likely to build, from an expression E, an automaton whose number of states is (rougly) equal to the literal length of E. These two facts together imply that there is little hope to find algorithms which are inverse of each other in these general families. However, the standard automaton of an expression on one hand, and an expression computed, for instance, by the state elimination method on the other hand, are of such particular form that the problem is certainly to be tackled.

In [27], Caron and Ziadi have described an algorithm, say Θ, which decides whether or not an automaton $\mathcal{A}$ is *the* standard automaton of an expression E; and if the answer is positive, Θ moreover computes an expression which is *almost* E, namely the *star normal form* of E as defined by Brüggemann-Klein

[22]. Even if Θ is not properly a Φ-type algorithm since it does not compute an expression for every automaton, it holds:

$$\text{For any star normal form regular expression } \mathsf{E}, \quad \Theta(\Psi_{\mathsf{s}}(\mathsf{E})) = \mathsf{E}\,.$$

The problem of finding an algorithm that is inverse of a Φ-algorithm has been addressed in a recent joint paper of mine and Sylvain Lombardy ([28]). We give there a partial solution to that problem in the following way.

There are two main ingredients in the construction of an algorithm Ω that gives back an automaton $\mathcal{A}$ from an expression that has been computed from $\mathcal{A}$. The first one is a *sligthly modified* derivation which, roughly speaking, 'breaks' the sums at the upper level. As a result, in particular, the corresponding derived term automaton may have *more than one* initial state. The second step is to take the *minimal co-quotient* of this new derived term automaton. [The minimal co-quotient is the transposed of the minimal quotient of the transposed automaton.] This Ω is not an inverse of a Φ-algorithm as described above but of a Φ'-algorithm which consists in performing first a *partial linearisation* Λ of the automaton $\mathcal{A}$ and then a normal Φ-algorithm. We then have (*cf.* [28] for more details):

$$\text{For any automaton } \mathcal{A}, \quad \Omega(\Phi'(\mathcal{A})) = \mathcal{A}\,.$$

Reducing the amount of information that one has to bring in with the linearisation Λ is the subject of ongoing research work.

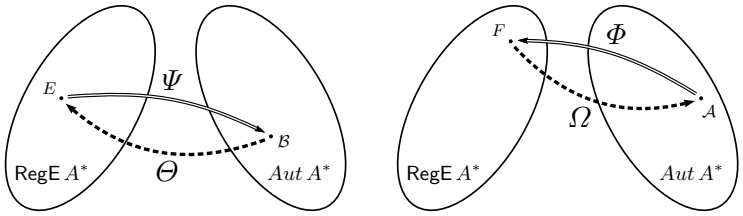


Fig. 8. The Θ and Ω algorithms

References

1. Eilenberg, S.: Automata, Languages, and Machines. Vol. A. Academic Press (1974)
2. Berstel, J.: Transductions and Context-free Languages. B. G. Teubner (1979)
3. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IRE Trans. Electronic Computers **9** (1960) 39–47
4. Wood, D.: Theory of Computation. John Wiley (1987)
5. Yu, S.: Regular languages. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Volume 1., Elsevier (1997) 41–111
6. Conway, J.H.: Regular Algebra and Finite Machines. Chapman and Hall (1971)

7. Krob, D.: Complete systems of B-rational identities. Theoret. Computer Sci. **89** (1991) 207–343
8. Sakarovitch, J.: Eléments de théorie des automates. Vuibert (2003) English translation: Elements of Automata Theory, Cambridge University Press, to appear.
9. Brzozowski, J.A., McCluskey, E.J.: Signal flow graph techniques for sequential circuit state diagrams. IEEE Trans. Electronic Computers **12** (1963) 67–76
10. Lombardy, S., Régis-Gianas, Y., Sakarovitch, J.: Introducing Vaucanson. Theoret. Computer Sci. **328** (2004) 67–76 Journal version of Proc. of CIAA 2003, Lect. Notes in Comp. Sc. 2759, (2003), 96–107 (with R. Poss).
11. Delgado, Morais: Approximation to the smallest regular expression for a given regular language. In Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S., eds.: Proc. of CIAA 04, Lecture Notes in Computer Science 3317, Springer (2004) 312–314
12. Claveirole, T., Lombardy, S., 0'Connor, S., Pouchet, L.N., Sakarovitch, J.: Inside Vaucanson. In Farré, J., Litovsky, I., eds.: Proc. of CIAA 05, Springer (2005) this volume.
13. Glushkov, V.M.: The abstract theory of automata. Russian Math. Surveys **16** (1961) 1–53
14. Caron, P., Flouret, M.: Glushkov construction for multiplicities. In Daley, M., Eramian, M., Yu, S., eds.: Pre-Proceedings of CIAA'00. (2000) 52–61
15. Lombardy, S., Sakarovitch, J.: Derivatives of rational expressions with multiplicity. Theor. Comput. Sci. **332** (2005) 141–177
16. Thompson, K.: Regular expression search algorithm. Comm. Assoc. Comput. Mach. **11** (1968) 419–422
17. Brzozowski, J.A.: Derivatives of regular expressions. J. Assoc. Comput. Mach. **11** (1964) 481–494
18. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theor. Comput. Sci. **155** (1996) 291–319
19. Berry, G., Sethi, R.: From regular expressions to deterministic automata. Theor. Comput. Sci. **48** (1986) 117–126
20. Berstel, J., Pin, J.E.: Local languages and the Berry-Sethi algorithm. Theor. Comput. Sci. **155** (1996) 439–446
21. Champarnaud, J.M., Ziadi, D.: Canonical derivatives, partial derivatives and finite automaton constructions. Theor. Comput. Sci. **289** (2002) 137–163
22. Brügemann-Klein, A.: Regular expressions into finite automata. Theor. Comput. Sci. **120** (1993) 197–213
23. Hagenah, C., Musholl, A.: Computing ε-free NFAs from regular expressions in $O(n \log^2(n))$ time. Theoret. Inform. Appl. **34** (2000) 257–277
24. Hromkovic, J., Seibert, S., Wilke, T.: Translating regular expressions into small ε-free nondeterministic finite automata. J. Comput. System Sci. **62** (2001) 565–588
25. Ilie, L., Yu, S.: Constructing NFAs by optimal use of positions in regular expressions. In Apolostolico, A., Takeda, M., eds.: Proc. of CPM'02, Lecture Notes in Computer Science 2373, Springer (2002) 279–288
26. Champarnaud, J.M., Nicart, F., Ziadi, D.: Computing the follow automaton of an expression. In Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S., eds.: Proc. of CIAA 04, Lecture Notes in Computer Science 3317, Springer (2004) 90–101
27. Caron, P., Ziadi, D.: Characterization of Glushkov automata. Theor. Comput. Sci. **233** (2000) 75–90
28. Lombardy, S., Sakarovitch, J.: How expressions can code for automata. Theoret. Inform. App. **39** (2005) 217–237