# A Unified Construction of the Glushkov, Follow, and Antimirov Automata[⋆,⋆⋆] (TR2006-880)

Cyril Allauzen and Mehryar Mohri

Courant Institute of Mathematical Sciences
251 Mercer Street, New York, NY 10012, USA
{allauzen, mohri}@cs.nyu.edu
http://www.cs.nyu.edu/~{allauzen,mohri}

**Abstract.** Many techniques have been introduced in the last few decades to create $\epsilon$-free automata representing regular expressions: Glushkov automata, the so-called follow automata, and Antimirov automata. This paper presents a simple and unified view of all these $\epsilon$-free automata both in the case of unweighted and weighted regular expressions. It describes simple and general algorithms with running time complexities at least as good as that of the best previously known techniques, and provides concise proofs. The construction methods are all based on two standard automata algorithms: epsilon-removal and minimization. This contrasts with the multitude of complicated and special-purpose techniques and proofs put forward by others to construct these automata. Our analysis provides a better understanding of $\epsilon$-free automata representing regular expressions: they are all the results of the application of some combinations of epsilon-removal and minimization to the classical Thompson automata. This makes it straightforward to generalize these algorithms to the weighted case, which also results in much simpler algorithms than existing ones. For weighted regular expressions over a closed semiring, we extend the notion of follow automata to the weighted case. We also present the first algorithm to compute the Antimirov automata in the weighted case.

## 1 Introduction

The construction of finite automata representing regular expressions has been widely studied due to its multiple applications to pattern-matching and many

⋆ This work was partially funded by the New York State Office of Science Technology and Academic Research (NYSTAR).

⋆⋆ This project was sponsored in part by the Department of the Army Award Number W23RYX-3275-N605. The U.S. Army Medical Research Acquisition Activity, 820 Chandler Street, Fort Detrick MD 21702-5014 is the awarding and administering acquisition office. The content of this material does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

other areas of text processing [1, 22]. The most classical construction, Thompson's construction [14, 25], creates a finite automaton with a number of states and transitions linear in the length $m$ of the regular expression. Figure 1(a) shows an example. The time complexity of the algorithm is also linear, $O(m)$. But Thompson's automaton contains transitions labeled with the empty string $\epsilon$ which create a delay in pattern matching.

Many alternative techniques have been introduced in the last few decades to create $\epsilon$-free automata representing regular expressions, in particular, Glushkov automata [11], follow automata [13], and Antimirov automata [2].

The Glushkov automaton, or position automaton, was independently introduced by [11] and [17]. Figure 1(b) shows an example for a particular regular expression. The automaton has exactly $n + 1$ states but up to $n^2$ transitions, where $n$ is the number of occurrences of alphabet symbols appearing in the expression. For a reasonable expression, $m = O(n)$, making it quadratically larger than the Thompson automaton. When using bit-parallelism for regular expression search, due to its smaller number of states, the Glushkov automaton can be represented with half the number of machine words required by the Thompson automaton [21, 22].

Several techniques have been suggested for constructing the Glushkov automaton. In [3], the construction is based on the recursive definition of the follow function and has a complexity of $O(n^3)$. The algorithm described by [4] has complexity of $O(m + n^2)$ and is based on an optimization of the recursive definition of the follow function. It requires the expression to be first rewritten in star-normal form, which can be done non-trivially in $O(m)$. Several other quadratic algorithms have been given: that of [9] which is based on an optimization of the follow recursion, and that of [23], based on the ZPC structure, which consists of two mutually linked copies of the syntactic tree of the expression.

The Antimirov or partial derivatives automaton was introduced by [2]. Figure 1(d) shows an example. It is in general smaller than the Glushkov automaton with up to $n+1$ states and up to $n^2$ transitions. It was in fact proven by [8] (see [13] for a simpler proof) to be the quotient of the Glushkov automaton for some equivalence relation. The complexity of the original construction algorithm by [2] is $O(m^5)$. [8] presented an algorithm whose complexity is $O(m^2)$.

Finally, the follow automaton was introduced by [13], it is the quotient of the Glushkov automaton by the *follow equivalence*: two states are equivalent if they have the same follow and the same finality. Figure 1(c) presents an example. The author gave an $O(m+n^2)$ algorithm where some $\epsilon$-transitions are removed from the automaton at each step of the construction of the Thompson construction as well as at the end. An $O(m + n^2)$ algorithm using the ZPC structure was given in [7], which requires the regular expression to be rewritten in star-normal form.

Some of these results have been extended to weighted regular expressions over arbitrary semirings. The generalization of the Thompson construction trivially follows from [24]. The Glushkov automaton can be naturally extended to the weighted case [5], and an $O(m^2)$ construction algorithm based on the generalization of the ZPC construct was given by [6]. The Antimirov automaton was

generalized to the weighted case by [16], but no explicit construction algorithm or complexity analysis was given by the authors.

This paper presents a simple and unified view of all these $\epsilon$-free automata (Glushkov, follow, and Antimirov) both in the case of unweighted and weighted regular expressions. It describes simple and general algorithms with running time complexities at least as good as that of the best previously known techniques, and provides concise proofs. The construction methods are all based on two standard automata algorithms: epsilon-removal[1] and minimization as summarized by the following table:

| Automaton | Algorithm | Complexity |
|---|---|---|
| Glushkov | $\mathrm{rmeps}(T)$ | $O(mn)$ |
| Follow | $\overline{\min(\mathrm{rmeps}(\overline{T}))}$ | $O(mn)$ |
| Antimirov | $\widehat{\mathrm{rmeps}}(\min(\mathrm{rmeps}(\widehat{T})))$ | $O(m \log m + mn)$ |

Where $T$ is the Thompson automaton, $\overline{T}$ is the automaton derived from $T$ by marking alphabet symbols with their position in the expression. When the symbols are marked, the same notation denotes the operation that removes the marking. $\widehat{T}$ is obtained by marking some $\epsilon$-transitions in $T$, making it deterministic (the $\epsilon$-transitions marked are removed by the $\widehat{\mathrm{rmeps}}$ operation).

This contrasts with the multitude of complicated and special-purpose techniques and proofs put forward by others to construct these automata. No need for fine-tuning some recursions, no requirement that the regular expression be in star-normal form, and no need to maintain multiple copies of the syntactic tree.

Our analysis provides a better understanding of $\epsilon$-free automata representing regular expressions: they are all the results of the application of some combinations of epsilon-removal and minimization to the classical Thompson automata. This makes it straightforward to generalize these algorithms to the weighted case by using the generalization of $\epsilon$-removal and minimization [18, 19]. This also results in much simpler algorithms than existing ones.

In particular, this leads to a straightforward algorithm for the construction of the Glushkov automaton of a weighted regular expression, and, in the case of closed semirings, allows us to generalize the notion of follow automaton to the weighted case. We also give the first explicit construction algorithm of the Antimirov automaton of a weighted expression. When the semiring is $k$-closed (or only $\epsilon$-$k$-closed for the regular expression in the Glushkov case), the complexities of the construction algorithms are the same as in the unweighted case.

## 2 Preliminaries

*Semirings* A *semiring* $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a ring that may lack negation. $\mathbb{K}$ is *closed* if $a^* = \bigoplus_{n \geq 0} a^n$ is defined for all $a \in \mathbb{K}$, and *$k$-closed* if there exists

[1] $\epsilon$-removal is less well known as an algorithm because it has often been and continue to be only presented as part of determinization in many textbooks.

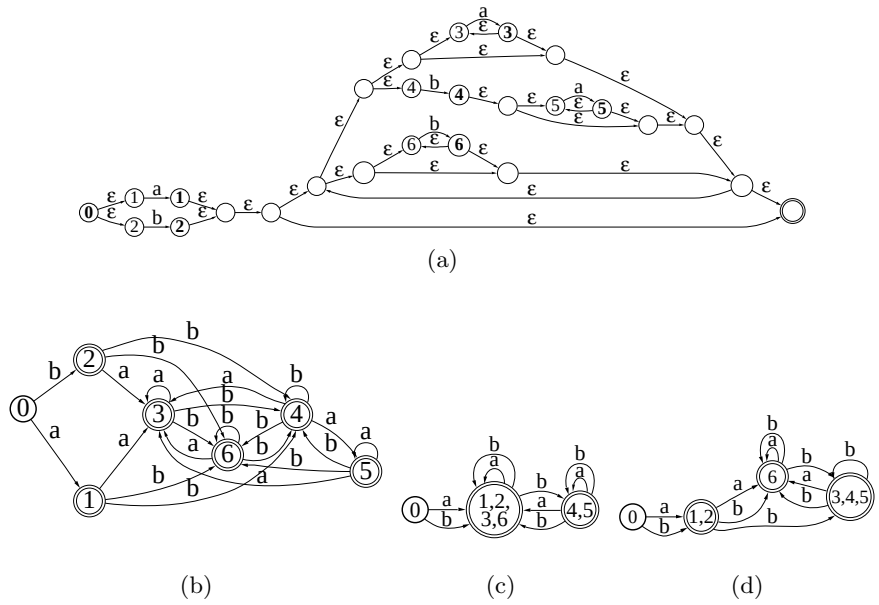



**Fig. 1.** (a) The Thompson automaton, (b) Glushkov automaton, (c) Follow automaton, and (d) Antimirov automaton representing the regular expression $\alpha = (a+b)(a^*+ba^*+b^*)^*$. This regular expression is the running example from [13].

$k \geq 0$ such that $a^* = a^k$ for all $a \in \mathbb{K}$. Examples of semirings are the boolean semiring $(\mathbb{B}, \vee, \wedge, 0, 1)$, the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, and the real semiring $(\mathbb{R}_+, +, \times, 0, 1)$.

*Weighted automata* A *weighted automaton* $A$ over a semiring $\mathbb{K}$ is a 7-uple $(\Sigma, Q, E, I, F, \lambda, \rho)$ where: $\Sigma$ is a finite alphabet; $Q$ is a finite set of states; $I \subseteq Q$ the set of initial states; $F \subseteq Q$ the set of final states; $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \mathbb{K} \times Q$ a finite set of transitions; $\lambda : I \to \mathbb{K}$ the initial weight function; and $\rho : F \to \mathbb{K}$ the final weight function mapping $F$ to $\mathbb{K}$.

Given a transition $e \in E$, we denote by $i[e]$ its input label, $p[e]$ its origin or previous state and $n[e]$ its destination state or next state, $w[e]$ its weight. Given a state $q \in Q$, we denote by $E[q]$ the set of transitions leaving $q$.

A *path* $\pi = e_1 \cdots e_k$ is an element of $E^*$ with consecutive transitions: $n[e_{i-1}] = p[e_i]$, $i = 2, \ldots, k$. We extend $n$ and $p$ to paths by setting: $n[\pi] = n[e_k]$ and $p[\pi] = p[e_1]$. A cycle $\pi$ is a path whose origin and destination states coincide: $n[\pi] = p[\pi]$. We denote by $P(q, q')$ the set of paths from $q$ to $q'$ and by $P(q, x, q')$ the set of paths from $q$ to $q'$ with input label $x \in \Sigma^*$ . These definitions can be extended to subsets $R, R' \subseteq Q$, by: $P(R, x, R') = \cup_{q \in R,\, q' \in R'} P(q, x, q')$. The labeling function $i$ and the weight function $w$ can also be extended to paths: $i[\pi] = i[e_1] \cdots i[e_k]$, $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$. The weight associated by $A$ to

each input string $x \in \Sigma^*$ is $[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$, $[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I, x, F) = \emptyset$.

*General algorithms* Let $A$ be a weighted automaton over $\mathbb{K}$. The shortest distance from $p$ to $q$ is defined as $d[p.q] = \bigoplus_{\pi \in P(p,q)} w[\pi]$. It can be computed using the generic single-source shortest-distance algorithm of [20] if $\mathbb{K}$ is $k$-closed for $A$, or using a generalization of Floyd-Warshall [15, 20] if $\mathbb{K}$ is closed for $A$.

The general $\epsilon$-removal algorithm of [19] consists of first computing the $\epsilon$-closure of each state $p$ in $A$,

$$\text{closure}(p) = \{(q, w) | w = d_\epsilon[p, q] = \bigoplus_{\pi \in P(p,q), i[\pi] = \epsilon} w[\pi] \neq \overline{0}\}, \tag{1}$$

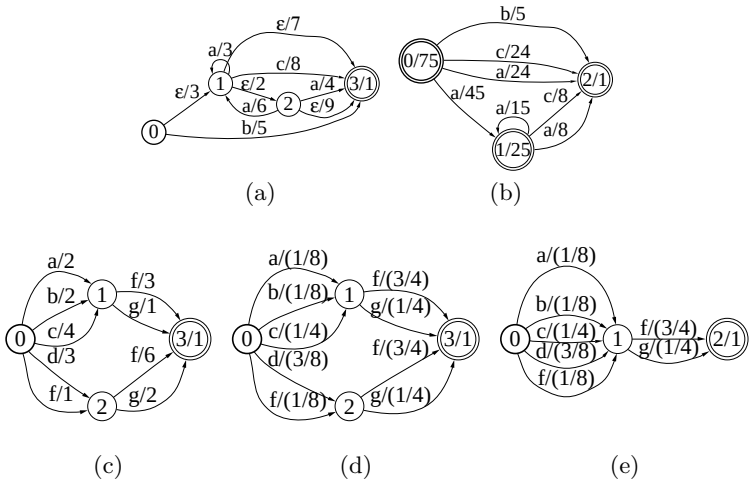
and then, for each state $p$, of deleting all the outgoing $\epsilon$-transitions of $p$, and adding out of $p$ all the non-$\epsilon$ transitions leaving each state $q \in \text{closure}(p)$ with their weight pre-$\otimes$-multiplied by $d_\epsilon[p, q]$. If $\mathbb{K}$ is $k$-closed for the $\epsilon$-cycles of $A$,[2] then the generic single-source shortest-distance algorithm [20] can be used to compute the $\epsilon$-closures.

Weight pushing [18] is a normalization algorithm that redistribute the weights along the paths of $A$ such that $\bigoplus_{e \in E[q]} w[e] + \rho(q) = \overline{1}$ for every state $q \in Q$, we will denote by $\text{push}(A)$ the resulting automaton. The algorithm requires that $\mathbb{K}$ is zero-sum free, weakly left divisible and closed or $k$-closed for $A$ since it depends on the computation of $d[q, F]$ for all $q \in Q$. It was proved in [18] that, if $A$ is deterministic (*i.e.* if no two transitions leaving any state share the same label and if it has a unique initial state), then the algorithm consisting of weight pushing followed by unweighted minimization (considering the pairs (label,weight) as a single symbol) leads to a minimal automaton equivalent to $A$, denoted by $\min(A)$.

See figure 2 for an illustration of these algorithms, more detailed descriptions are given in the appendix.

*Regular expressions* A *weighted regular expression* over the semiring $\mathbb{K}$ is recursively defined by: $\emptyset$, $\epsilon$ and $a \in \Sigma$ are regular expressions, and if $\alpha$ and $\beta$ are regular expressions then $k\alpha$, $\alpha k$ for $k \in \mathbb{K}$, $\alpha + \beta$, $\alpha \cdot \beta$ and $\alpha^*$ are also regular expressions. We denote by $|\alpha|$ the length of $\alpha$, and by $|\alpha|_\Sigma$ the width $\alpha$, *i.e.* the number of occurrences of alphabet symbols in $\alpha$. Let $\text{pos}(\alpha) = \{1, 2, \ldots, |\alpha|_\Sigma\}$ be the set of (alphabet symbol) positions in $\alpha$. An unweighted regular expression can be seen as a weighted expression over the boolean semiring $(\mathbb{B}, \vee, \wedge, 0, 1)$. We denote by $A_T(\alpha)$ the *Thompson automaton* of $\alpha$ and by $I_{A_T(\alpha)}$ and $F_{A_T(\alpha)}$ its unique initial and final states. For $i \in \text{pos}(\alpha)$, we defined $p_i$ and $q_i$ as the states such that the alphabet symbol at the $i$-th position in $\alpha$ corresponds to the transition from $p_i$ to $q_i$. These states are the only states having respectively a non-$\epsilon$ outgoing or incoming transition.

[2] For $A$ to be well defined, $\mathbb{K}$ needs to be closed for the $\epsilon$-cycles of $A$.




(a) (b) (c) (d) (e)

**Fig. 2.** (a) A weighted automaton $A_1$ over the real semiring $(\mathbb{R}_+, +, \times, 0, 1)$. (b) The result of the application of $\epsilon$-removal to $A$. (c) A weighted automaton $A_2$ over the real semiring $(\mathbb{R}_+, +, \times, 0, 1)$. (d) The result of weight pushing. (e) The result of minimization. The initial weight in the last two automata is 64.

## 3 Glushkov Automaton

Let $\alpha$ be a weighted regular expression over the alphabet $\Sigma$ and the semiring $\mathbb{K}$. We denote by $\overline{\alpha}$ the weighted regular expression obtained by marking each symbol of $\alpha$ with its position. The *Glushkov* or *position automaton* $A_G(\alpha)$ of $\alpha$ is is defined by the 7-uple $(\Sigma, \mathrm{pos}_0(\alpha), E, 0, \overline{1}, F, \rho)$ where $\mathrm{pos}_0(\alpha) = \mathrm{pos}(\alpha) \cup \{0\}$,

$$E = \{(i, a, w, j) : (j, w) \in \mathrm{follow}(\overline{\alpha}, i) \text{ and } \mathrm{pos}(\alpha, j) = a\}, \tag{2}$$

and for $i \in \mathrm{pos}_0(\alpha)$, $i \in F$ iff there exist $w \in \mathbb{K}$ such that $(i, w) \in \mathrm{last}_0(\overline{\alpha})$, and then $\rho(i) = w$.

The functions $\mathrm{null}(\overline{\alpha}) \in \mathbb{K}$, $\mathrm{first}(\overline{\alpha}) \subseteq \mathrm{pos}(\overline{\alpha}) \times \mathbb{K}$, $\mathrm{last}(\overline{\alpha}) \subseteq \mathrm{pos}(\overline{\alpha}) \times \mathbb{K}$ and $\mathrm{follow}(\overline{\alpha}, i) \subseteq \mathrm{pos}(\overline{\alpha}) \times \mathbb{K}$ are recursively defined over the subterms of $\overline{\alpha}$ as shown in the tables below. We also define $\mathrm{follow}(\overline{\alpha}, 0) = \mathrm{first}(\overline{\alpha})$ and $\mathrm{last}_0(\overline{\alpha})$ as $\mathrm{last}(\overline{\alpha}) \cup \{(0, \mathrm{null}(\overline{\alpha}))\}$ if $\mathrm{null}(\overline{\alpha}) \neq \overline{0}$, and $\mathrm{last}(\overline{\alpha})$ otherwise. For $X \subseteq \mathrm{pos}(\alpha) \times \mathbb{K}$, $k \in \mathbb{K}$ and $i \in \mathrm{pos}(\alpha)$, $k \cdot X = \{(i, k \times w) | (i, w) \in X\}$ if $k \neq \overline{0}$, $\overline{0} \cdot X = \emptyset$ ($X \cdot k$ is defined similarly), and $\langle X, i \rangle = w$ if there exists $w$ such that $(i, w) \in X$, and $\langle X, i \rangle = \overline{0}$ otherwise. The union of two weighted subsets $X$ and $Y$ is defined by $X \cup Y = \{(i, \langle X, i \rangle \oplus \langle Y, i \rangle) | \langle X, i \rangle \oplus \langle Y, i \rangle \neq \overline{0}\}$. For example, $\{(i, w)\} \cup \{(i, w')\} = \{(i, w \oplus w')\}$.

| | null | first | last |
|---|---|---|---|
| $\emptyset$ | $\overline{0}$ | $\emptyset$ | $\emptyset$ |
| $\epsilon$ | $\overline{1}$ | $\emptyset$ | $\emptyset$ |
| $a_i$ | $\overline{0}$ | $\{(i, \overline{1})\}$ | $\{(i, \overline{1})\}$ |
| $k\beta$ | $k \otimes \text{null}(\beta)$ | $k \cdot \text{first}(\beta)$ | $\text{last}(\beta)$ |
| $\beta k$ | $\text{null}(\beta) \otimes k$ | $\text{first}(\beta)$ | $\text{last}(\beta) \cdot k$ |
| $\beta + \gamma$ | $\text{null}(\beta) \oplus \text{null}(\gamma)$ | $\text{first}(\beta) \cup \text{first}(\gamma)$ | $\text{last}(\beta) \cup \text{last}(\gamma)$ |
| $\beta \cdot \gamma$ | $\text{null}(\beta) \otimes \text{null}(\gamma)$ | $\text{first}(\beta) \cup \text{null}(\beta) \cdot \text{first}(\gamma)$ | $\text{last}(\beta) \cdot \text{null}(\gamma) \cup \text{last}(\gamma)$ |
| $\beta^*$ | $\text{null}(\beta)^*$ | $\text{null}(\beta)^* \cdot \text{first}(\beta)$ | $\text{last}(\beta) \cdot \text{null}(\beta)^*$ |

| $\cdot$ | $\text{follow}(\cdot, i)$ |
|---|---|
| $\emptyset$ | $\emptyset$ |
| $\epsilon$ | $\emptyset$ |
| $a_i$ | $\emptyset$ |
| $k\beta$ | $\text{follow}(\beta, i)$ |
| $\beta k$ | $\text{follow}(\beta, i)$ |

| $\cdot$ | $\text{follow}(\cdot, i)$ |
|---|---|
| $\beta + \gamma$ | $\begin{cases} \text{follow}(\beta, i) \text{ if } i \in \text{pos}(\beta) \\ \text{follow}(\gamma, i) \text{ if } i \in \text{pos}(\gamma) \end{cases}$ |
| $\beta \cdot \gamma$ | $\begin{cases} \text{follow}(\beta, i) \cup \langle \text{last}(\beta), i \rangle \cdot \text{first}(\gamma) & \text{if } i \in \text{pos}(\beta) \\ \text{follow}(\gamma, i) & \text{if } i \in \text{pos}(\gamma) \end{cases}$ |
| $\beta^*$ | $\text{follow}(\beta, i) \cup \langle \text{last}(\beta^*), i \rangle \cdot \text{first}(\gamma)$ |

$\text{null}(\alpha) = \text{null}(\overline{\alpha})$ is the value associated by $\overline{\alpha}$ to $\epsilon$. For $\alpha$ to be well defined, $\text{null}(\beta)^*$ must be defined for every subterm $b^*$. There is in fact a very simple relationship between the first, last and follow functions and the $\epsilon$-closures of the states in the Thompson automaton that admit a non-$\epsilon$ incoming transition.

**Lemma 1.** *Let $\alpha$ be a weighted regular expression. Let $A = A_T(\alpha)$. Then*

*(i)* $(i, w) \in \text{first}(\overline{\alpha})$ *iff* $(p_i, w) \in \text{closure}(I_A)$*;*
*(ii)* $(i, w) \in \text{follow}(\overline{\alpha}, j)$ *iff* $(p_i, w) \in \text{closure}(q_j)$*; and*
*(iii)* $(i, w) \in \text{last}(\overline{\alpha})$ *iff* $(F_A, w) \in \text{closure}(q_i)$.

*Proof.* The proof is by induction on the length of the regular expression. If $\alpha = a$, $\alpha = \epsilon$ or $\alpha = \emptyset$, then the properties trivially hold. Due to lack of space, we will only treat the case $\alpha = \beta \cdot \gamma$, other cases can be treated similarly. Let $A = A_T(\alpha)$, $B = A_T(\beta)$ and $C = A_T(\gamma)$.

If $\alpha = \beta \cdot \gamma$, then $\text{closure}_A(I_A) = \text{closure}_B(I_B) \cup [\![B]\!][\epsilon] \cdot \text{closure}_C(I_A)$, thus (i) recursively holds since $[\![B]\!][\epsilon] = \text{null}(\beta)$. If $j \in \text{pos}(\gamma)$, then $\text{closure}_A(q_j) = \text{closure}_C(q_j)$. Otherwise $j \in \text{pos}(\beta)$ and

$$\text{closure}_A(q_j) = \text{closure}_B(q_j) \cup \langle \text{closure}_B(q_j), F_B \rangle \cdot \text{closure}_C(I_C). \tag{3}$$

Thus, (ii) and (iii) recursively hold. □

The following theorem follows directly from the lemma just presented.

**Theorem 1.** *Let $\alpha$ be a weighted regular expression. Then:*

$$A_G(\alpha) = \text{rmeps}(A_T(\alpha)). \tag{4}$$

Let $\alpha$ be a weighted regular expression $\alpha$ over $\mathbb{K}$. We will say that $\mathbb{K}$ is *$\epsilon$-$k$-closed for* $\alpha$ if there exist $k$ such that for every subterm $\beta^*$ of $\alpha$, $\text{null}(\beta)^* = \text{null}(\beta)^k$.

**Lemma 2.** *Let $A$ be the Thompson automaton of a weighted regular expression over a $k$-closed semiring. There is a queue discipline for which the complexity of the single-source shortest-distance algorithm from any state in $A$ is linear.*

*Proof.* We define the subterm depth of a state $q$ in $A$ as the number of subterms $\beta + \gamma$ and $\beta^*$ it belongs to. We then use a larger subterm-depth first queue discipline. The queue can be maintained in constant time since (1) there is at most two states having the same subterm depth in the queue at anytime and (2) if $d$ is the maximal subterm depth of an element in the queue at a given time, the subterm depth of the state inserted next will be $d-1$, $d$ or $d+1$. □

**Theorem 2.** *Let $\alpha$ be a weighted regular expression over a semiring $\mathbb{K}$ that is $\epsilon$-$k$-closed for $\alpha$. The Glushkov automaton of $\alpha$ can be constructed in time $O(mn)$ by applying $\epsilon$-removal to its Thompson automaton.*

*Proof.* If $\mathbb{K}$ is $\epsilon$-$k$-closed for $\alpha$, then $\mathbb{K}$ is $k$-closed for all the paths considered during the computation of the $\epsilon$-closures and, by Lemma 2, each $\epsilon$-closure can be computed in $O(m)$. Since $n+1$ closures need to be computed, the total complexity is in $O(mn + n^2) = O(mn)$. □

In the unweighted case, the unpublished manuscript [10] showed that the Glushkov automaton could be obtained by removing the $\epsilon$-transitions from the Thompson automaton. However, the authors used a special-purpose $\epsilon$-removal algorithm and not the classical $\epsilon$-removal algorithm, limiting the scope of their results.

## 4 Follow Automaton

The *follow automaton* of an unweighted regular expression $\alpha$, denoted by $A_F(\alpha)$ was introduced by [13]. It is the quotient of $A_G(\alpha)$ by the equivalence relation $\equiv_F$ defined over $\mathrm{pos}_0(\alpha)$ by:

$$i \equiv_F j \text{ iff } \begin{cases} \{i,j\} \subseteq \mathrm{last}_0(\overline{\alpha}) \text{ or } \{i,j\} \cap \mathrm{last}_0(\overline{\alpha}) = \emptyset, \text{ and} \\ \mathrm{follow}(\overline{\alpha}, i) = \mathrm{follow}(\overline{\alpha}, j). \end{cases} \tag{5}$$

**Theorem 3.** *For any regular expression $\alpha$, the following identities hold:*

$$A_F(\overline{\alpha}) = \min(A_G(\overline{\alpha}))$$
$$A_F(\alpha) = \overline{\min(A_G(\overline{\alpha}))}.$$

Note that it is mentioned in [13] that minimization could be used to construct the follow automata but the authors claim that the complexity of minimization would be in $O(n^2 \log n)$ making this approach less efficient. The following theorem shows that minimization has in fact a better complexity in this case. Observe that $A_G(\overline{\alpha})$ is deterministic.

**Theorem 4.** *The time complexity of the Hopcroft's minimization algorithm when applied to $A_G(\overline{\alpha})$ is linear, i.e., in $O(n^2)$ where $n = |\alpha|_\Sigma$.*

*Proof.* Due to space constraints, we will give only a sketch of the proof. The $\log |Q|$ factor in Hopcroft's algorithm corresponds to the number of times the incoming transitions at a given state $q$ are used to split a subset (tentative equivalence class). In $A_G(\overline{\alpha})$, transitions sharing the same label have all the same destination state (the automaton is *1-local*), thus each incoming transition of a state $q$ can only be used once to split a subset. □

This theorem actually holds for all 1-local automata.

This leads to a simple algorithm for constructing the follow automaton of a regular expression $\alpha$:

$$A_F(\alpha) = \overline{\min(\text{rmeps}(A_T(\overline{\alpha})))}. \tag{6}$$

whose complexity $O(mn)$ is identical to that of the more complicated and special-purpose algorithms of [13, 7]. When the semiring $\mathbb{K}$ is weakly divisible, zero-sum free, and closed, we can then define the *follow automaton* of a weighted regular expression $\alpha$ as: $A_F(\alpha) = \overline{\min(A_G(\overline{\alpha}))}$.

**Theorem 5.** *If $\mathbb{K}$ is k-closed, then $A_F(\alpha)$ can be computed in $O(mn)$.*

*Proof.* The shortest-distance computation required by weight pushing can be done in $O(m)$ in the case of $A_T(\overline{\alpha})$ and is preserved by $\epsilon$-removal. The weighted automaton $\text{push}(A_G(\overline{\alpha}))$ is 1-local when considered as a finite automaton over pairs (label, weight), thus theorem 4 can be applied. □

## 5 Antimirov Automaton

In the following we will consider pairs $(w, \alpha)$ with $w \in \mathbb{K}$, and we define $k \cdot (w, \alpha) = (k \otimes w, \alpha)$, $(w, \alpha) \cdot k = (w, \alpha k)$ and $(w, \alpha) \cdot \beta = (w, \alpha \cdot \beta)$. These operations can naturally be extended to multisets[3] of pairs (weight, expression).

The *partial derivative* of $\alpha$ with respect to $a \in \Sigma$ is the multiset of pairs (weight, expression) recursively defined by:

$$\begin{array}{ll} \partial_a(\epsilon) = \partial_a(1) = \emptyset & \partial_a(\beta + \gamma) = \partial_a(\beta) \cup \partial_a(\gamma) \\ \partial_a(b) = \epsilon \text{ if } a = b, \emptyset \text{ otherwise} & \partial_a(\beta \cdot \gamma) = \partial_a(\beta) \cdot \gamma \cup \text{null}(\beta) \cdot \partial_a(\gamma) \\ \partial_a(k\beta) = k \cdot \partial_a(\beta) & \partial_a(\beta^*) = \text{null}(\beta)^* \cdot \partial_a(\beta) \cdot \beta^* \\ \partial_a(\beta k) = \partial_a(\beta) \cdot k & \end{array}$$

The partial derivative of $\alpha$ with respect to the string $s \in \Sigma^*$, denoted $\partial_s(\alpha)$, is recursively defined by $\partial_{sa}(\alpha) = \partial_a(\partial_s(\alpha))$. Let $D(\alpha) = \{\beta : (w, \beta) \in \partial_s(\alpha) \text{ with } s \in \Sigma^* \text{ and } w \in \mathbb{K}\}$. Note that for $D(\alpha)$ to be well-defined, we need to define when two expressions are the same. Here we will only allow the following identities: $\emptyset \cdot \alpha = \alpha \cdot \emptyset = \emptyset$, $\emptyset + \alpha = \alpha + \emptyset = \emptyset$, $\overline{0}\alpha = \alpha\overline{0} = \emptyset$, $\epsilon \cdot \alpha = \alpha \cdot \epsilon = \alpha$, $\overline{1}\alpha = \alpha\overline{1} = \alpha$, $k(k'\alpha) = (k \otimes k')\alpha$, $(\alpha k)k' = \alpha(k \otimes k')$ and $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$. [4]

[3] By multisets, we mean that $\{(w, \alpha)\} \cup \{(w', \alpha)\} = \{(w, \alpha), (w', \alpha)\}$.

[4] These identities are the *trivial identities* considered in [16] except for the last two which were added to simplify our presentation. Any larger set of identities can be handled with our method by rewriting $\alpha$ in the corresponding normal form.

The *Antimirov* or *partial derivatives automaton* of $\alpha$ is defined by the 7-uple $(\Sigma, D(\alpha), E, \alpha, \overline{1}, F, \mathrm{null})$ where $E = \{(\beta, a, w, \gamma) | w = \bigoplus_{(w',\gamma)\in\partial_a(\beta)} w'\}$ and $F = \{\beta \in D(\alpha) | \mathrm{null}(\beta) \neq \overline{0}\}$.

Let $\widehat{\Sigma} = \Sigma \cup \{\epsilon^1_+, \epsilon^2_+, \epsilon^1_*, \epsilon^2_*\}$. We denote by $\widehat{A_T(\alpha)}$ the weighted automaton over $\widehat{\Sigma}$ obtained by recursively marking some of the $\epsilon$-transitions of $A_T(\alpha)$ as follows: if $\alpha = \beta + \gamma$, we label by $\epsilon^1_+$ (resp. $\epsilon^2_+$) the $\epsilon$-transition from $I_{A_T(\alpha)}$ to $I_{A_T(\beta)}$ (resp. $I_{A_T(\gamma)}$); if $\alpha = \beta^*$, we label by $\epsilon^1_*$ (resp. $\epsilon^2_*$) the two $\epsilon$-transitions to $I_{A_T(\beta)}$ (resp. $F_{A_T(\alpha)}$). Observe that $\widehat{A_T(\alpha)}$ can be viewed as an automaton recognizing the expression $\widehat{\alpha}$ over $\widehat{\Sigma}$ recursively defined by $\widehat{\emptyset} = \emptyset$, $\widehat{\epsilon} = \epsilon$, $\widehat{a} = a$, $\widehat{k\beta} = k\widehat{\beta}$, $\widehat{\beta k} = \widehat{\beta} k$, $\widehat{\beta + \gamma} = \epsilon^1_+\widehat{\beta} + \epsilon^2_+\widehat{\gamma}$, $\widehat{\beta \cdot \gamma} = \widehat{\beta} \cdot \widehat{\gamma}$ and $\widehat{\beta^*} = (\epsilon^1_\star \widehat{\beta})^* \epsilon^2_\star$.

For $i \in \mathrm{pos}_0(\alpha)$, we use the same notation $q_i$ (with $q_0 = I$) for the corresponding states in $A_T(\alpha)$, $\widehat{A_T(\alpha)}$ and $\mathrm{rmeps}(\widehat{A_T(\alpha)})$. For a state $q$ in $\mathrm{rmeps}(\widehat{A_T(\alpha)})$, we define by $L(q)$ the language recognized from $q$ considering $\mathrm{rmeps}(\widehat{A_T(\alpha)})$ as an unweighted automaton over pairs (symbol,weight). Lemma 3 follows from our marking of the $\epsilon$-transitions.

**Lemma 3.** *For $i \in \mathrm{pos}_0(\alpha)$, $L(q_i)$ uniquely defines a regular expression over $\Sigma$, denoted by $\delta_i$ (or $\delta^\alpha_i$ when there is an ambiguity).*

**Lemma 4.** *For all $i \in \mathrm{pos}_0(\alpha)$ and $j \in \mathrm{pos}(\alpha)$, we have for $p_j$, $q_i$ in $A_T(\alpha)$ that:*

$$(p_j, w) \in \mathrm{closure}(q_i) \text{ iff } (w, \delta_j) \in \partial_a(\delta_i). \tag{7}$$

*Proof.* The proof is by induction on the length of the regular expression. If $\alpha = a$, $\alpha = \epsilon$ or $\alpha = \emptyset$, then the properties trivially hold. Due to the lack of space, we will only treat the case $\alpha = \beta \cdot \gamma$, other cases can be treated similarly. Let $A = A_T(\alpha)$, $B = A_T(\beta)$ and $C = A_T(\gamma)$.

If $q_i$ is in $C$, then $\delta^\alpha_i = \delta^\gamma_i$ and $\mathrm{closure}_A(q_i) = \mathrm{closure}_C(q_i)$. Therefore, if $(w, p_j) \in \mathrm{closure}_A(q_i)$, $p_j$ is in $C$ and then $\delta^\alpha_j = \delta^\gamma_j$. Hence (7) recursively holds.

If $q_i$ is in $B$, then $\delta^\alpha_i = \delta^\beta_i \cdot \gamma$ and we have:

$$\partial_a(\delta^\alpha_i) = \partial_a(\delta^\beta_i) \cdot \gamma \cup \mathrm{null}(\delta^\beta_i) \cdot \partial_a(\gamma) \tag{8}$$

$$\mathrm{closure}_A(q_i) = \mathrm{closure}_B(q_i) \cup \mathrm{null}(\delta^\beta_i) \cdot \mathrm{closure}_C(I_C). \tag{9}$$

By induction, we have that $(p_j, w) \in \mathrm{closure}_B(q_i)$ iff $(w, \delta^\beta_j) \in \partial_a(\delta^\beta_i)$, and $(p_j, w) \in \mathrm{closure}_C(I_C)$ iff $(w, \delta^\gamma_j) \in \partial_a(\delta^\gamma_0) = \partial_a(\gamma)$. Hence (7) follows. □

Observe that $\delta_0 = \alpha$, hence lemma 4 implies that the $\delta_i$ are the derived terms of $\alpha$, more precisely, $i \mapsto \delta_i$ is a surjection from $\mathrm{pos}_0(\alpha)$ onto $D(\alpha)$. This leads us to the following result, where $\min_{\mathbb{B}}$ is unweighted minimization when each pair (label,weight) is treated as regular symbol and $\widehat{\mathrm{rmeps}}$ denotes the removal of the marked $\epsilon$'s.

**Theorem 6.** *We have $A_A(\alpha) = \widehat{\mathrm{rmeps}}(\min_{\mathbb{B}}(\mathrm{rmeps}(\widehat{A_T(\alpha)})))$.*

*Proof.* Note that $\text{rmeps}(\widehat{A_T(\alpha)})$ is deterministic. During minimization, two states $q_i$ and $q_j$ are equivalent iff $L(q_i) = L(q_j)$, *i.e.* $\delta_i = \delta_j$ (by lemma 3). Hence, there is a bijection between $D(\alpha)$ and the set of states of $\min_{\mathbb{B}}(\text{rmeps}(\widehat{A_T(\alpha)}))$ having an incoming transition with label in $\Sigma$, and hence between $D(\alpha)$ and the set of states of $A = \widehat{\text{rmeps}}(\min_{\mathbb{B}}(\text{rmeps}(\widehat{A_T(\alpha)})))$. Lemma 4 ensures that the transitions in $A$ is consistent with the definition of $A_A(\alpha)$. □

**Theorem 7.** *If $\mathbb{K}$ is $\epsilon$-$k$-closed, then $A_A(\alpha)$ can be computed in $O(m \log m + mn)$.*

Theorem 7 follows from the fact that $\text{rmeps}(\widehat{A_T(\alpha)})$ has $O(m)$ states and transitions. In the unweighted case, this complexity is a good as the more complicated and best known algorithm of [8].

In the weighted case, the use of minimization over (label,weight) pairs is sub-optimal since states that would be equivalent modulo a $\otimes$-multiplicative factor are not merged. When possible, using weighted minimization instead would lead to a smaller automaton in general. Hence, if $\mathbb{K}$ is closed, we can defined the *normalized Antimirov automaton* of $\alpha$ as $\widehat{\text{rmeps}}(\min_{\mathbb{K}}(\text{rmeps}(\widehat{A_T(\alpha)})))$. This automaton would always be smaller than the Antimirov automaton and the automaton of unitary derived terms of [16][5]. If $\mathbb{K}$ is $k$-closed, it can be constructed in $O(m \log m + mn)$.

*Remark* When the condition about $k$-closedness (resp. $\epsilon$-$k$-closedness for $\alpha$) of $\mathbb{K}$ is relaxed to the closedness of $\mathbb{K}$ (resp. that $\alpha$ is well-defined), all our construction algorithms can still be used by replacing the generic single-source shortest-distance algorithm with a generalization of the Floyd-Warshall algorithm [15, 20], leading to a complexity in $O(m^3)$. It is not hard however to maintain the quadratic complexity by modifying the generic single-source shortest-distance algorithm to take advantage of the special topology of the Thompson automaton.

In the unweighted case, every regular expression can be staightforwardedly rewritten in $\epsilon$-normal form such that $m = O(n)$. In that case, our $O(mn)$ or $O(m \log m + mn)$ complexities become $O(m + n^2)$ which is what is often reported in the literature.

## 6 Conclusion

We presented a simple and unified view of $\epsilon$-free automata representing unweighted and weighted regular expressions. We showed that standard unweighted and weighted epsilon-removal and minimization can be used to create the Glushkov, follow, and Antimirov automata and that the complexities of these algorithms match those of the best known algorithms. This provides a better understanding

[5] This automaton can be viewed in our approach as the result of a simpler form of reweighting than weight pushing, the reweighting used by weighted minimization.

of the $\epsilon$-free automata representing regular expressions. It also suggests using other combinations of epsilon-removal and minimization for creating $\epsilon$-free automata. For example, in some contexts, it might be beneficial to use reverse-epsilon-removal rather than epsilon-removal [19]. Note also that the Glushkov automaton can be constructed on-the-fly since Thompson's construction and epsilon-removal both admit an on-demand implementation.

## References


1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques and Tools*. Addison Wesley: Reading, MA, 1986.
2. V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.
3. G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48(3):117–126, 1986.
4. A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.
5. P. Caron and M. Flouret. Glushkov construction for series: the non commutative case. *International Journal of Computer Mathematics*, 80(4):457–472, 2003.
6. J.-M. Champarnaud, É. Laugerotte, F. Ouardi, and D. Ziadi. From regular weighted expressions to finite automata. In *Proceedings of CIAA 2003*, volume 2759 of *Lecture Notes in Computer Science*, pages 49–60. Springer-Verlag, 2003.
7. J.-M. Champarnaud, F. Nicart, and D. Ziadi. Computing the follow automaton of an expression. In *Proceedings of CIAA 2004*, volume 3317 of *Lecture Notes in Computer Science*, pages 90–101. Springer-Verlag, 2005.
8. J.-M. Champarnaud and D. Ziadi. Computing the equation automaton of a regular expression in $O(s^2)$ space and time. In *Proceedings of CPM 2001*, volume 2089 of *Lecture Notes in Computer Science*, pages 157–168. Springer-Verlag, 2001.
9. C.-H. Chang and R. Page. From regular expressions to DFA's using compressed NFA's. *Theoretical Computer Science*, 178(1-2):1–36, 1997.
10. D. Giammarresi, J.-L. Ponty, and D. Wood. Glushkov and Thompson constructions: a synthesis. `http://www.cs.ust.hk/tcsc/RR/1998-11.ps.gz`, 1998.
11. V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.
12. J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Proceedings of the International Symposium on the Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.
13. L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):146–162, 2003.
14. S. C. Kleene. Representations of events in nerve sets and finite automata. In C. E. Shannon, J. McCarthy, and W. R. Ashby, editors, *Automata Studies*, pages 3–42. Princeton University Press, 1956.
15. D. J. Lehmann. Algebraic structures for transitives closures. *Theoretical Computer Science*, 4:59–76, 1977.
16. S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *Theoretical Computer Science*, 332(1-3):142–177, 2005.
17. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1):39–47, 1960.

18. M. Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2, 1997.
19. M. Mohri. Generic $\epsilon$-removal and input $\epsilon$-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.
20. M. Mohri. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
21. G. Navarro and M. Raffinot. Fast regular expression search. In *Proceedings of WAE'99*, volume 1668 of *Lecture Notes in Computer Science*, pages 198–212. Springer-Verlag, 1999.
22. G. Navarro and M. Raffinot. *Flexible pattern matching*. Cambridge University Press, 2002.
23. J.-L. Ponty, D. Ziadi, and J.-M. Champarnaud. A new quadratic algorithm to convert a regular expression into automata. In *Proceedings of WIA'96*, volume 1260 of *Lecture Notes in Computer Science*, pages 109–119. Springer-Verlag, 1997.
24. M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
25. K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):365–375, 1968.

# A General algorithms

## A.1 Shortest distance

A generic single-source shortest-distance algorithm in weighted automata was presented in [20]. The algorithm is a generalization of the classical shortest-distance algorithms. It does not require the semiring to be idempotent. For a weighted automaton $A$ over $\mathbb{K}$, the condition for the algorithm to work is that $K$ must be $k$-closed for $A$, *i.e.* there exist $k \in \mathbb{N}$ such that for any cycle $c$ in $A$, $w[c]^* = w[c]^k$.

shortest-distance$(A, s)$

```
1  for each p ∈ Q do
2      d[p] ← r[p] ← 0̄
3  d[s] ← r[s] ← 1̄
4  S ← {s}
5  while S ≠ ∅ do
6      q ← head(S)
7      dequeue(S)
8      R ← r[q]
9      r[q] ← 0̄
10     for each e ∈ E[q] do
11         if d[n[e]] ≠ d[n[e]] ⊕ (R ⊗ w[e]) then
12             d[n[e]] ← d[n[e]] ⊕ (R ⊗ w[e])
13             r[n[e]] ← r[n[e]] ⊕ (R ⊗ w[e])
14             if n[e] ∉ S
15                 enqueue(S, n[e])
16 d[s] ← 1̄
```

**Fig. 3.** Pseudocode of the generic shortest-distance algorithm.

The algorithm is also generic in the sense that it works with any queue discipline. The pseudocode of the algorithm is given figure 3. The complexity of the algorithm depends on the queue discipline chosen for $S$, more precisely it is in:

$$O(|Q| + (T_\oplus + T_\otimes + C(A))|E| \max_{q \in Q} N(q) + (C(I) + C(X)) \sum_{q \in Q} N(q)) \qquad (10)$$

where $N(q)$ denotes the number of times state $q$ is extracted from the queue $S$, $C(X)$ the cost of extracting a state from $S$, $C(I)$ the cost of inserting a state in $S$, and $C(A)$ the cost of an assignment.

In the case of an acyclic automaton, using the topological order queue discipline, the complexity of the algorithm is linear, *i.e.*, $O(|Q| + |E|)$. In the case of the tropical semiring, using Fibonacci heaps, the complexity of the algorithm is $O(|E| + |Q| \log |Q|)$.

$\epsilon$-removal($A$)

```
1  for each p ∈ Q do
2      E[p] ← {e ∈ E[p] : i[e] ≠ ϵ}
3      for each (q, w) ∈ C[p] do ▷ C[p] = closure(p)
4          E[p] ← E[p] ∪ {(p, a, w ⊗ w′, r) : (q, a, w′, r) ∈ E[q] and a ≠ ϵ}
5          if q ∈ F then
6              F ← F ∪ {p}
7          ρ[p] ← ρ[p] ⊕ (w ⊗ ρ[q]
```

**Fig. 4.** Pseudocode of the $\epsilon$-removal algorithm.

### A.2 Epsilon removal

Let $A$ be a weighted automaton over $\mathbb{K}$ with $\epsilon$-transitions. Let $A_\epsilon$ be the automaton obtained by deleting all the transitions not labeled by $\epsilon$ from $A$. A general $\epsilon$-removal algorithm based on the generic shortest distance algorithm presented above was given in [19]. This algorithms works if the semiring $\mathbb{K}$ is $k$-closed for $A_\epsilon$.

The algorithm is divided in two steps. The first step consists of computing the $\epsilon$-closure of each state $p$ in $A$. Let $d_\epsilon[p, q]$ denote the $\epsilon$-distance from $p$ to $q$, for $p, q \in Q$:

$$d_\epsilon[p, q] = \bigoplus_{\pi \in P(p,q), i[\pi]=\epsilon} w[\pi]. \tag{11}$$

The $\epsilon$-closure of $p$ is then defined as

$$\text{closure}(p) = \{(q, d_\epsilon[p, q]) | d_\epsilon[p, q] \neq \overline{0}\}. \tag{12}$$

The $\epsilon$-closure of $p$ can be computed by using the generic shortest-distance algorithm on $A_\epsilon$ with source $p$.

The second step consist of, for each state $p$ having at least an incoming non-$\epsilon$ transition, deleting all the outgoing $\epsilon$-transitions of $p$, and adding out of $p$ all the non-$\epsilon$ transitions leaving each state $q \in \text{closure}(p)$ with their weight pre-$\otimes$-multiplied by $d_\epsilon[p, q]$. The pseudocode of this second step is given figure 4.

### A.3 Weight pushing

Weight pushing is an algorithm for normalizing the distribution of the weights along the paths of a weighted automata [18].

Let $A$ be a weighted automaton over $\mathbb{K}$ and assume that $\mathbb{K}$ is weakly left divisible and zero sum free. For every state $q \in Q$, assume that the shortest distance from $q$ to $F$:

$$d_F[q] = \bigoplus_{\pi \in P(q,F)} (w[\pi] \otimes \rho(n[\pi])) \tag{13}$$

is well defined in $\mathbb{K}$. The *weight pushing* algorithm consists of computing each $d_F[q]$ and of *reweighting* $A$ in the following way:

$$\begin{array}{lll} \forall e \in E \text{ such that } d_F[p[e]] \neq \overline{0}, & w[e] \leftarrow d_F[p[e]]^{-1} \otimes (w[e] \otimes d_F[n[e]]) & \\ \forall q \in I, & \lambda[q] \leftarrow \lambda[q] \otimes d_F[q] & (14) \\ \forall q \in F \text{ such that } d_F[q] \neq \overline{0}, & \rho[q] \leftarrow d_F[q]^{-1} \otimes \rho[q] & \end{array}$$

The complexity of the reweighting step is linear in the size of $A$ under the assumption that the cost of the $\otimes$ operation is constant. The first step can be achieve by applying the shortest-distance algorithm on the reverse of $A$, hence the complexity of this step is as discussed in section A.1.

Weight pushing has two interesting properties: (1) it does no change the weight of successful paths, (2) the resulting weighted automaton is *stochastic*, *i.e.* for any state $q$, the $\oplus$-sum of the weight of the outgoing transitions in $q$ is equal to $\overline{1}$.

### A.4 Weighted minimization

A weighted automaton $A$ is *deterministic* if no two transitions leaving any state share the same label and if it has a unique initial state. A deterministic weighted automaton is *minimal* if there exists no other deterministic automaton having a smaller number of states and realizing the same function.

A general weighted minimization was presented in [18]. Let $A$ be a weighted automaton over $\mathbb{K}$, the algorithm consists of the execution of the following steps:

1. weight pushing,
2. (unweighted) automata minimization, considering each pair $(\text{label}, \text{weight})$ as a single label.

Assuming that the conditions of application of weight pushing hold, the resulting weighted automaton, denoted by $\min(A)$, is minimal and equivalent to $A$. The complexity of the second step is in $O(|E| \log |Q|)$ using the Hopcroft algorithm [12].