A Unified Construction of the Glushkov, Follow, and Antimirov Automata

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. A number of different techniques have been introduced in the last few decades to create ϵ-free automata representing regular expressions such as the *Glushkov automata*, *follow automata*, or *Antimirov automata*. This paper presents a simple and unified view of all these construction methods both for unweighted and weighted regular expressions. It describes simpler algorithms with time complexities at least as favorable as that of the best previously known techniques, and provides a concise proof of their correctness. Our algorithms are all based on two standard automata operations: epsilon-removal and minimization. This contrasts with the multitude of complicated and special-purpose techniques previously described in the literature, and makes it straightforward to generalize these algorithms to the weighted case. In particular, we extend the definition and construction of follow automata to the case of weighted regular expressions over a closed semiring and present the first algorithm to compute weighted Antimirov automata.

1 Introduction

The construction of finite automata representing regular expressions has been widely studied due to their multiple applications to pattern-matching and many other areas of text processing [1, 21]. The most classical construction, Thompson's construction [13, 24], creates a finite automaton with a number of states and transitions linear in the length m of the regular expression. The time complexity of the algorithm is also linear, $O(m)$. But Thompson's automaton contains transitions labeled with the empty string ϵ which create a delay in pattern matching. Many alternative techniques have been introduced in the last few decades to create ϵ-free automata representing regular expressions, in particular, Glushkov automata [11], follow automata [12], and Antimirov automata [2].

The Glushkov automaton, or position automaton, was independently introduced by [11] and [16]. It has exactly $n+1$ states but may have up to n^2 transitions, where n is the number of occurrences of alphabet symbols appearing in the expression. Thus, its size is quadratic in that of the Thompson automaton for reasonable regular expressions for which $m = O(n)$. However, when using bit-parallelism for regular expression search, thanks to its smaller number of states, the Glushkov automaton can be represented with half the number of machine words required by the Thompson automaton [20, 21].

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

Table 1. Simple algorithms for the construction of Glushkov, follow, and Antimirov automata and their time complexity. T is the Thompson automaton. For an automaton A, $\overline{A}$ denotes the automaton derived from A 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 ϵ-transitions in T, making it deterministic (the ϵ-transitions marked are removed by the $\widehat{\text{rmeps}}$ operation).

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 its complexity is in $O(n^3)$. The algorithm described by [4] is based on an optimization of the recursive definition of the follow function and its complexity is in $O(m + n^2)$. 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 [22], 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]. 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 [12] for a simpler proof) to be the quotient of the Glushkov automaton for some equivalence relation. The complexity of the original construction algorithm of [2] is $O(m^5)$. [8] presented an algorithm whose complexity is $O(m^2)$.

Finally, the follow automaton was introduced by [12]. 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. The author gave an $O(m + n^2)$ algorithm where some ϵ-transitions are removed from the automaton at each step 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 [23]. 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 [15], but no explicit construction algorithm or complexity analysis was given by the authors.

This paper presents a simple and unified view of all these ϵ-free automata (Glushkov, follow, and Antimirov) both in the case of unweighted and weighted regular expressions. It describes simpler algorithms with time complexities at least as favorable as that of the best previously known techniques, and provides concise proofs. Our algorithms are all based on two standard automata operations: epsilon-removal and minimization, as summarized in Table 1.

This contrasts with the multitude of complicated and special-purpose techniques and proofs described 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 ϵ-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 ϵ-removal and minimization [17, 18]. It 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, helps us generalize follow automata 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, in the Glushkov case, only null-k-closed for the regular expression considered, the complexity of our algorithms is the same as in the unweighted case.

The paper is organized as follows. Section 2 introduces the definitions and a brief description of the elementary algorithms used in the following sections. Section 3 presents and analyzes our algorithm for the construction of the Glushkov automaton, Section 4 the same for the follow automaton, and Section 5 for the Antimirov automaton.

2 Preliminaries

Semirings. A system $(\mathbb{K}, \oplus, \otimes, \overline{0}, \overline{1})$ is a *semiring* when $(\mathbb{K}, \oplus, \overline{0})$ is a commutative monoid with identity element $\overline{0}$; $(\mathbb{K}, \otimes, \overline{1})$ is a monoid with identity element $\overline{1}$; $\otimes$ distributes over $\oplus$; and $\overline{0}$ is an annihilator for $\otimes$: for all $a \in \mathbb{K}, a \otimes \overline{0} = \overline{0} \otimes a = \overline{0}$. Thus, a semiring is a ring that may lack negation. Some familiar semirings include the boolean semiring $(\mathbb{B}, \vee, \wedge, 0, 1)$, the tropical semiring $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$, and the real semiring $(\mathbb{R}_+, +, \times, 0, 1)$.

A semiring $\mathbb{K}$ is said to be *closed* if for all $a \in \mathbb{K}$, the infinite sum $\bigoplus_{n=0}^{\infty} a^n$ is well-defined and in $\mathbb{K}$, and if associativity, commutativity, and distributivity apply to countable sums [19]. $\mathbb{K}$ is said to be k-closed if for all $a \in \mathbb{K}$, $\bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n$. More generally, we will say that $\mathbb{K}$ is *closed (k-closed) for an automaton* A, if the closedness (resp. k-closedness) axioms hold for all cycle weights of A. In some semirings, e.g., the probability semiring $(\mathbb{R}_+, +, \times, 0, 1)$, the equality $\bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n$ may hold for the cycle weights of A only approximately, modulo $\epsilon > 0$. A is then said to be *ϵ-k-closed* for that semiring.

Weighted automata. A *weighted automaton* A over a semiring $\mathbb{K}$ is a 7-tuple $(\Sigma, Q, E, I, \lambda, F, \rho)$ where: Σ 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; $\lambda : I \rightarrow \mathbb{K}$ the initial weight function; and $\rho : F \rightarrow \mathbb{K}$ the final weight function mapping F to $\mathbb{K}$.

We denote by $p[\pi]$ the origin and $n[\pi]$ the destination state of a path $\pi \in E^*$ in an automaton A. $i[\pi]$ denotes the label of π, and $w[\pi]$ its weight obtained by $\otimes$-multiplying the weights of its constituent transitions. We also denote by $P(p,q)$ the set of paths from p to q and by $P(I,x,F)$ the set of paths from the initial states I to the final states F labeled with $x \in \Sigma^*$. The weight associated by A to an input string $x \in \Sigma^*$ is obtained by summing the weights of these paths multiplied by their initial and final weights: $[\![A]\!](x) = \bigoplus_{\pi \in P(I,x,F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi])$. $[\![A]\!](x)$ is defined to be $\overline{0}$ when $P(I,x,F) = \emptyset$.

Shortest distance. 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 [19] if $\mathbb{K}$ is k-closed for A, or using a generalization of Floyd-Warshall [14, 19] if $\mathbb{K}$ is closed for A.

Epsilon-removal. The general ϵ-removal algorithm of [18] consists of first computing the ϵ-closure of each state p in A,

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

and then, for each state p, of deleting all the outgoing ϵ-transitions of p, and adding out of p all the non-ϵ 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 ϵ-cycles of A,[1] then the generic single-source shortest-distance algorithm [19] can be used to compute the ϵ-closures.

Weight-pushing and weighted minimization. Weight-pushing [17] is a normalization algorithm that redistributes 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 denote by push(A) the resulting automaton. The algorithm requires that $\mathbb{K}$ be 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 [17] 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 weight-pushing followed by unweighted minimization with each pair (label, weight) viewed as an alphabet symbol, leads to a minimal deterministic weighted automaton equivalent to A, denoted by min(A).

Regular expressions. A *weighted regular expression* over the semiring $\mathbb{K}$ is defined recursively by: $\emptyset$, ϵ and $a \in \Sigma$ are regular expressions, and if α and β are regular expressions then $k\alpha$, αk for $k \in \mathbb{K}$, $\alpha + \beta$, $\alpha \cdot \beta$ and α^* are also regular expressions. We denote by null(α) the weight associated by α to the empty string ϵ. A weighted regular expression α is well-defined iff for every subterm of the form β^*, null$(\beta)^*$ is well-defined and in $\mathbb{K}$. We will say that $\mathbb{K}$ is *null-k-closed for α* if there exist $k \geq 0$ such that for every subterm β^* of α, $\text{null}(\beta)^* = \bigoplus_{i=0}^{k} \text{null}(\beta)^i$. We denote by $|\alpha|$ the *length of* α, and by $|\alpha|_\Sigma$ the *width of* α, *i.e.*, the number of occurrences of alphabet symbols in α. Let $\text{pos}(\alpha) = \{1, 2, \ldots, |\alpha|_\Sigma\}$ be the set of (alphabet symbol) positions in α. An unweighted regular expression can be seen as a weighted expression over the boolean semiring $(\mathbb{B}, \vee, \wedge, 0, 1)$.

Thompson automaton. We denote by $A_T(\alpha)$ the *Thompson automaton* of α and by $I_{A_T(\alpha)}$ and $F_{A_T(\alpha)}$ its unique initial and final states. For $i \in \text{pos}(\alpha)$, we denote by p_i and q_i the states of $A_T(\alpha)$ such that the transition from p_i to q_i is labeled with the alphabet symbol at the i-th position in α. The states p_i

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

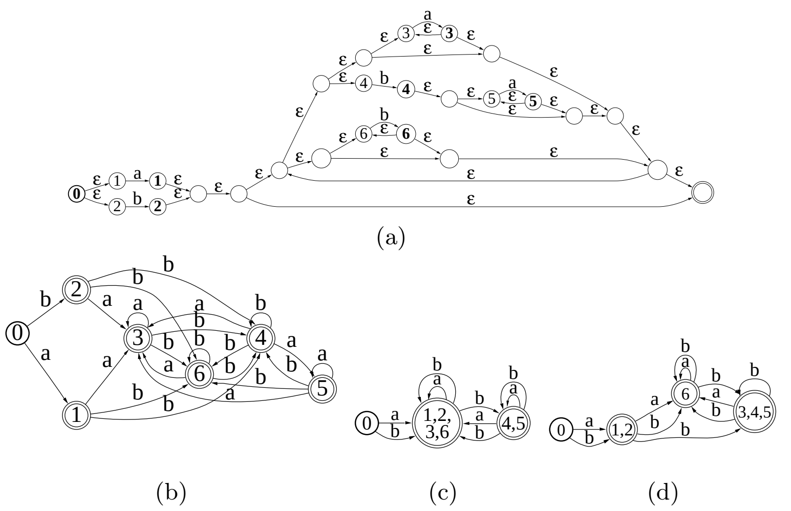

Fig. 1. (a) The Thompson automaton, (b) Glushkov automaton, (c) Follow automaton, and (d) Antimirov automaton of the regular expression $\alpha = (a+b)(a^* + ba^* + b^*)^*$, the running example used in [12]. In (d), state $\{0\}$ corresponds to the derived term α, $\{1,2\}$ to $\tau = (a^* + ba^* + b^*)^*$, $\{6\}$ to $a^*\tau$, and $\{3,4,5\}$ to $b^*\tau$.

(states q_i) are the only states having a non-ϵ outgoing (resp. incoming) transition. Figure 1(a) shows the Thompson automaton in the special case of the regular expression $\alpha = (a+b)(a^* + ba^* + b^*)^*$.

3 Glushkov Automaton

Let α be a weighted regular expression over the alphabet Σ and the semiring $\mathbb{K}$. The Glushkov automaton of α is an ϵ-free non-deterministic weighted automaton representing α that has an initial state plus one state for each position in α, *i.e.* each occurrence of an alphabet symbol in α. Figure 1(b) shows an example.

The formal definition of the Glushkov automaton is based on the functions null, first, last, and follow. Table 2 gives the recursive definition of these functions. $\text{null}(\overline{\alpha}) \in \mathbb{K}$ is the weight associated by $\overline{\alpha}$ to the empty string ϵ and is thus the final weight of the initial state of the automaton. $\text{last}(\overline{\alpha}) \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is the set of positions with the corresponding final weights where a non-empty string accepted by α can end. $\text{first}(\overline{\alpha}) \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is the set of positions with the corresponding weights that can be reached by reading one alphabet symbol from the initial state. Similarly, $\text{follow}(\overline{\alpha}, i) \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ is the set of positions with the corresponding weights that can be reached by reading one alphabet symbol from the position i.

In these definitions, the union of two weighted subsets $X, Y \subseteq \text{pos}(\overline{\alpha}) \times \mathbb{K}$ 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 any weighted subset $X \subseteq \text{pos}(\alpha) \times \mathbb{K}$, weight $k \in \mathbb{K}$, and position i, $k \otimes X$ denotes the weighted

$\overline{\alpha}$	$\mathrm{null}(\overline{\alpha})$	$\mathrm{first}(\overline{\alpha})$	$\mathrm{last}(\overline{\alpha})$	$\mathrm{follow}(\overline{\alpha}, i)$
$\emptyset$	$\overline{0}$	$\emptyset$	$\emptyset$	$\emptyset$
ϵ	$\overline{1}$	$\emptyset$	$\emptyset$	$\emptyset$
a_j	$\overline{0}$	$\{(j, \overline{1})\}$	$\{(j, \overline{1})\}$	$\emptyset$
$k\beta$	$k \otimes \mathrm{null}(\beta)$	$k \otimes \mathrm{first}(\beta)$	$\mathrm{last}(\beta)$	$\mathrm{follow}(\beta, i)$
βk	$\mathrm{null}(\beta) \otimes k$	$\mathrm{first}(\beta)$	$\mathrm{last}(\beta) \otimes k$	$\mathrm{follow}(\beta, i)$
$\beta + \gamma$	$\mathrm{null}(\beta) \oplus \mathrm{null}(\gamma)$	$\mathrm{first}(\beta) \cup \mathrm{first}(\gamma)$	$\mathrm{last}(\beta) \cup \mathrm{last}(\gamma)$	$\begin{cases} \mathrm{follow}(\beta, i) \text{ if } i \in \mathrm{pos}(\beta) \\ \mathrm{follow}(\gamma, i) \text{ if } i \in \mathrm{pos}(\gamma) \end{cases}$
$\beta \cdot \gamma$	$\mathrm{null}(\beta) \otimes \mathrm{null}(\gamma)$	$\mathrm{null}(\beta) \otimes \mathrm{first}(\gamma) \cup \mathrm{first}(\beta)$	$\mathrm{last}(\beta) \otimes \mathrm{null}(\gamma) \cup \mathrm{last}(\gamma)$	$\begin{cases} \mathrm{follow}(\beta, i) \cup \langle \mathrm{last}(\beta), i\rangle \otimes \mathrm{first}(\gamma) \text{ if } i \in \mathrm{pos}(\beta) \\ \mathrm{follow}(\gamma, i) \text{ if } i \in \mathrm{pos}(\gamma) \end{cases}$
β^*	$\mathrm{null}(\beta)^*$	$\mathrm{null}(\beta)^* \otimes \mathrm{first}(\beta)$	$\mathrm{last}(\beta) \otimes \mathrm{null}(\beta)^*$	$\mathrm{follow}(\beta, i) \cup \langle \mathrm{last}(\beta^*), i\rangle \otimes \mathrm{first}(\gamma)$

Table 2. Definition of the functions null, first, last, and follow. For convenience, 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 $\mathrm{null}(\overline{\alpha}) \neq \overline{0}$, $\mathrm{last}(\overline{\alpha})$ otherwise.

subset and $\langle X, i\rangle$ the weight defined by:

$$k \otimes X = \begin{cases} \{(i, k \otimes w) | (i, w) \in X\} & \text{if } k \neq \overline{0}, \\ \emptyset & \text{otherwise,} \end{cases} \quad \text{and } \langle X, i\rangle = \begin{cases} w & \text{if } \exists\, w\colon (i, w) \in X, \\ \overline{0} & \text{otherwise.} \end{cases}$$

$X \otimes k$ is defined similarly. Let $\overline{\alpha}$ denote the weighted regular expression obtained by marking each symbol of α with its position. The *Glushkov* or *position automaton $A_G(\alpha)$ of* α is defined by $A_G(\alpha) = (\Sigma, \mathrm{pos}_0(\alpha), E, 0, \overline{1}, F, \rho)$ where its states set is $\mathrm{pos}_0(\alpha) = \{0\} \cup \mathrm{pos}(\alpha)$ and its transition set

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

A state $i \in \mathrm{pos}_0(\alpha)$ is final iff there exist $w \in \mathbb{K}$ such that $(i, w) \in \mathrm{last}_0(\overline{\alpha})$ and when it is final $\rho(i) = w$. The following lemma shows that there exists a simple relationship between the first, last, and follow functions and the ϵ-closures of the states in the Thompson automaton that admit a non-ϵ incoming transition (states q_i).

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

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

Proof. Note that if $\alpha = a$, $\alpha = \epsilon$ or $\alpha = \emptyset$, then the properties trivially hold. The proof is by induction on the length of the regular expression and is given in the case $\alpha = \beta \cdot \gamma$. Other cases can be treated similarly.

Assume that the properties hold for all expressions shorter than α. Let $A = A_T(\alpha)$, $B = A_T(\beta)$ and $C = A_T(\gamma)$. If $\alpha = \beta \cdot \gamma$, then $\mathrm{closure}_A(I_A) = \mathrm{closure}_B(I_B) \cup [\![B]\!][\epsilon] \otimes \mathrm{closure}_C(I_A)$, thus, since $[\![B]\!][\epsilon] = \mathrm{null}(\beta)$, (i) holds by induction. If $j \in \text{pos}(\gamma)$, then $\text{closure}_A(q_j) = \text{closure}_C(q_j)$. Otherwise, if $j \in \text{pos}(\beta)$, then

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

Thus, by induction, both (ii) and (iii) hold. □

The following proposition follows directly from the lemma just presented.

Proposition 1. *Let α be a weighted regular expression. Then*

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

Proof. The only states potentially accessible in $\text{rmeps}(A_T(\overline{\alpha}))$ are the states q_i, $i \geq 0$, since they are the only states with non-ϵ incoming transitions. A state q_i is final with weight w in $\text{rmeps}(A_T(\overline{\alpha}))$ iff $(F_{A_T(\overline{\alpha})}, w) \in \text{closure}(q_i)$, that is, by Lemma 1, iff $(i, w) \in \text{last}_0(\overline{\alpha})$. (q_j, a_i, w, q_i) is a transition in $\text{rmeps}(A_T(\overline{\alpha}))$ iff $(p_i, w) \in \text{closure}(q_j)$, that is, by Lemma 1, iff $(i, w) \in \text{follow}(\overline{\alpha}, j)$. Thus, $A_G(\overline{\alpha}) = \text{rmeps}(A_T(\overline{\alpha}))$. □

This proposition suggests a natural algorithm to compute the Glushkov automaton. The following lemma helps determine its complexity.

Lemma 2. *Let A be the Thompson automaton of a weighted regular expression over a k-closed semiring and let s be a state of A. Then, the shortest-distance algorithm of [19] can be used to compute the shortest distances from the source state s to all states of A in linear time.*

Proof. We give a sketch of the proof. The complexity of the single-source shortest-distance algorithm of [19] depends on the queue discipline used, that is the order in which states are extracted from the queue. One can use a queue discipline that takes advantage of the specific structure of the Thompson automaton. Each subterm of the form $\beta + \gamma$ or β^* defines a sub-automaton with an entry state and an exit state. The appropriate queue discipline enforces that each sub-automaton be fully visited before being exited. The algorithm of [19] can also be modified to store the shortest-distance through the sub-automaton of a β^* subterm once it has been computed and avoid a subsequent revisit. With that queue discipline, the complexity of the algorithm is linear. □

Theorem 1. *Let α be a weighted regular expression over a semiring $\mathbb{K}$ null-k-closed for α and let $m = |\alpha|$ and $n = |\alpha|_\Sigma$. Then, the Glushkov automaton of α can be constructed in time $O(mn)$ by applying ϵ-removal to its Thompson automaton.*

Proof. If $\mathbb{K}$ is null-k-closed for α, then $\mathbb{K}$ is k-closed for all the paths considered during the computation of the ϵ-closures and, by Lemma 2, each ϵ-closure can be computed in linear time, that is 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 of [10] showed that the Glushkov automaton could be obtained by removing the ϵ-transitions from the Thompson automaton using a special-purpose ϵ-removal algorithm.

4 Follow Automaton

The *follow automaton* of an unweighted regular expression α, denoted by $A_F(\alpha)$ was introduced by [12]. Figure 1(c) shows an example. It is the quotient of $A_G(\alpha)$ by the equivalence relation $\equiv_F$ defined over $\text{pos}_0(\alpha)$ by:

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

Proposition 2. *For any regular expression α, the following identities hold:*

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

Note that it is mentioned in [12] 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 lemma shows that minimization has in fact a better complexity in this case. Observe that $A_G(\overline{\alpha})$ is deterministic.

Lemma 3. *The time complexity of Hopcroft's minimization algorithm applied to $A_G(\overline{\alpha})$ is linear in the size of $A_G(\overline{\alpha})$: it is in $O(n^2)$ where $n = |\alpha|_\Sigma$.*

Proof. We give 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 to split a subset once. The number of transitions in $A_G(\overline{\alpha})$ is at most n^2. □

The lemma holds in fact for all 1-local automata. This leads to a simple algorithm for constructing the follow automaton of a regular expression α based on:

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

The complexity of this algorithm is in $O(mn)$ which is the same as that of the more complicated and special-purpose algorithms of [12, 7]. When the semiring $\mathbb{K}$ is weakly divisible, zero-sum free, and closed, we can define the *follow automaton* of a weighted regular expression α as: $A_F(\alpha) = \overline{\min(A_G(\overline{\alpha}))}$.

Theorem 2. *Let α be a weighted regular expression over $\mathbb{K}$. If $\mathbb{K}$ is k-closed for the Thompson automaton of α, then the follow automaton of α can be computed in $O(mn)$ by applying epsilon-removal followed by weighted minimization to the Thompson automaton of α.*

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 ϵ-removal. The weighted automaton $\text{push}(A_G(\overline{\alpha}))$ is 1-local when considered as a finite automaton over pairs (label, weight), thus Lemma 3 can be applied. □

5 Antimirov Automaton

The definition of the Antimirov automaton of a regular expression is based on that of the *partial derivatives of regular expressions*, which are multisets of pairs of the form (w, α) where $w \in \mathbb{K}$ is a weight and α a weighted regular expression over $\mathbb{K}$. For any weight $k \in \mathbb{K}$ and any regular expression β, we define the following operations:

$$k \otimes (w, \alpha) = (k \otimes w, \alpha) \quad (w, \alpha) \otimes k = (w, \alpha k) \quad (w, \alpha) \cdot \beta = (w, \alpha \cdot \beta), \qquad (7)$$

which can be naturally extended to multisets of pairs (w, α). By multisets, we mean that $\{(w, \alpha)\} \cup \{(w, \alpha')\} = \{(w, \alpha), (w, \alpha')\}$. The *partial derivative of* α *with respect to* $a \in \Sigma$ is the multiset of pairs (w, α) defined recursively by:

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

The *partial derivative of* α *with respect to the string* $s \in \Sigma^*$ is denoted by $\partial_s(\alpha)$ and recursively defined by $\partial_{sa}(\alpha) = \partial_a(\partial_s(\alpha))$. Let $D(\alpha) = \{\beta : (w, \beta) \in \partial_s(\alpha)$ with $s \in \Sigma^*$ 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 $(\alpha + \beta) \cdot \gamma = \alpha \cdot \gamma + \beta \cdot \gamma$.[2]

The *Antimirov* or *partial derivatives automaton* $A_A(\alpha)$ of α is then the automaton defined by $A_A(\alpha) = (\Sigma, D(\alpha), E, \alpha, \overline{1}, F, \text{null})$ where $E = \{(\beta, a, w, \gamma) : w = \bigoplus_{(w', \gamma) \in \partial_a(\beta)} w'\}$ and $F = \{\beta \in D(\alpha) : \text{null}(\beta) \neq \overline{0}\}$. Figure 1(d) shows the Antimirov automaton for a specific regular expression.

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 ϵ-transitions of the Thompson automaton $A_T(\alpha)$ as follows: if $\alpha = \beta + \gamma$, we label by ϵ^1_+ (ϵ^2_+) the ϵ-transition from $I_{A_T(\alpha)}$ to $I_{A_T(\beta)}$ (resp. $I_{A_T(\gamma)}$); if $\alpha = \beta^*$, we label by ϵ^1_* (ϵ^2_*) the two ϵ-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 $i \in \text{pos}_0(\alpha)$, we use the same notation q_i (with $q_0 = I$) for the corresponding states in $A_T(\alpha)$, $\widehat{A_T(\alpha)}$ and $\text{rmeps}(\widehat{A_T(\alpha)})$. For a state q in $\text{rmeps}(\widehat{A_T(\alpha)})$, we define by $L(q)$ the language recognized from q considering $\text{rmeps}(\widehat{A_T(\alpha)})$ as an unweighted automaton over pairs (symbol,weight). Lemma 4 follows from our marking of the ϵ-transitions.

[2] These identities are the *trivial identities* considered in [15] 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 α in the corresponding normal form.

Lemma 4. *For $i \in \mathrm{pos}_0(\alpha)$, $L(q_i)$ uniquely defines a regular expression over Σ, denoted by δ_i (or δ_i^α in the presence of ambiguity).*

Lemma 5. *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:*

$$(p_j, w) \in \mathrm{closure}(q_i) \text{ iff } (w, \delta_j) \in \partial_a(\delta_i). \tag{8}$$

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. We give the proof in 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 $\delta_i^\alpha = \delta_i^\gamma$ 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 $\delta_j^\alpha = \delta_j^\gamma$. Hence (8) recursively holds.

If q_i is in B, then $\delta_i^\alpha = \delta_i^\beta \cdot \gamma$ and we have:

$$\partial_a(\delta_i^\alpha) = \partial_a(\delta_i^\beta) \cdot \gamma \cup \mathrm{null}(\delta_i^\beta) \otimes \partial_a(\gamma) \tag{9}$$

$$\mathrm{closure}_A(q_i) = \mathrm{closure}_B(q_i) \cup \mathrm{null}(\delta_i^\beta) \otimes \mathrm{closure}_C(I_C). \tag{10}$$

By induction, we have $(p_j, w) \in \mathrm{closure}_B(q_i)$ iff $(w, \delta_j^\beta) \in \partial_a(\delta_i^\beta)$, and $(p_j, w) \in \mathrm{closure}_C(I_C)$ iff $(w, \delta_j^\gamma) \in \partial_a(\delta_0^\gamma) = \partial_a(\gamma)$. Hence (8) follows. □

Note that $\delta_0 = \alpha$, thus Lemma 5 implies that the δ_i are the derived terms of α, 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 ϵ's.

Proposition 3. *We have $A_A(\alpha) = \widehat{\mathrm{rmeps}}(\min_{\mathbb{B}}(\mathrm{rmeps}(\widehat{A_T(\alpha)})))$.*

Proof. Note that $\mathrm{rmeps}(\widehat{A_T(\alpha)})$ is deterministic. During minimization, two states q_i and q_j are merged iff $L(q_i) = L(q_j)$, that is, by Lemma 4, iff $\delta_i = \delta_j$. Thus, there is a bijection between $D(\alpha)$ and the set of states of $\min_{\mathbb{B}}(\mathrm{rmeps}(\widehat{A_T(\alpha)}))$ having an incoming transition with label in Σ, and thus also between $D(\alpha)$ and the set of states of $A = \widehat{\mathrm{rmeps}}(\min_{\mathbb{B}}(\mathrm{rmeps}(\widehat{A_T(\alpha)})))$. Lemma 5 ensures that the transitions of A are consistent with the definition of $A_A(\alpha)$. □

Theorem 3. *Let α be a weighted regular expression over $\mathbb{K}$. If $\mathbb{K}$ is null-k-closed for α, then the Antimirov automaton of α can be computed in $O(m \log m + mn)$ using ϵ-removal and minimization.*

Theorem 3 follows from the fact that $\mathrm{rmeps}(\widehat{A_T(\alpha)})$ has $O(m)$ states and transitions. In the unweighted case, this complexity matches that of the more complicated and best known algorithm of [8].

In the weighted case, the use of minimization over (label,weight) pairs is suboptimal 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. For example, if $\mathbb{K}$ is closed, we can defined the *normalized Antimirov automaton* of α as $\widehat{\text{rmeps}}(\min_{\mathbb{K}}(\text{rmeps}(\widehat{A_T(\alpha)})))$. This automaton is always smaller than the Antimirov automaton and the automaton of unitary derived terms of [15].[3] When $\mathbb{K}$ is k-closed, it can be constructed in $O(m \log m + mn)$.

Remark. When the condition about k-closedness (null-k-closedness for α) of $\mathbb{K}$ is relaxed to the closedness of $\mathbb{K}$ (resp. that α 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 [14, 19], leading to a complexity of $O(m^3)$. 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 straightforwardly rewritten in ϵ-normal form such that $m = O(n)$. In that case, our $O(mn)$ and $O(m \log m + mn)$ complexities become $O(m + n^2)$ which coincides with what is often reported in the literature.

6 Conclusion

We presented a simple and unified view of ϵ-free automata representing unweighted and weighted regular expressions. We showed that standard unweighted and weighted epsilon-removal and minimization algorithms can be used to create the Glushkov, follow, and Antimirov automata and that the time complexity of our construction algorithms is at least as favorable as that of the best previously known algorithm.

This provides a better understanding of the ϵ-free automata representing regular expressions. It also suggests using other combinations of epsilon-removal and minimization for creating ϵ-free automata. For example, in some contexts, it might be beneficial to use reverse-epsilon-removal rather than epsilon-removal [18]. 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.

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

References

1. A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques and Tools.* Addison Wesley: Reading, MA, 1986.

[3] 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.

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. L. Ilie and S. Yu. Follow automata. *Information and Computation*, 186(1):146–162, 2003.
13. 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.
14. D. J. Lehmann. Algebraic structures for transitives closures. *Theoretical Computer Science*, 4:59–76, 1977.
15. S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *Theoretical Computer Science*, 332(1-3):142–177, 2005.
16. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1):39–47, 1960.
17. M. Mohri. Finite-State Transducers in Language and Speech Processing. *Computational Linguistics*, 23:2, 1997.
18. M. Mohri. Generic e-removal and input e-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143, 2002.
19. M. Mohri. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.
20. 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.
21. G. Navarro and M. Raffinot. *Flexible pattern matching.* Cambridge University Press, 2002.
22. 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.
23. M.-P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
24. K. Thompson. Regular expression search algorithm. *Communications of the ACM*, 11(6):365–375, 1968.