Regular Expressions into Finite Automata

Anne Brüggemann-Klein*

February 19, 1996

Abstract

It is a well-established fact that each regular expression can be transformed into a non-deterministic finite automaton (NFA) with or without ϵ-transitions, and all authors seem to provide their own variant of the construction. Of these, Berry and Sethi [BS86] have shown that the construction of an ϵ-free NFA due to to Glushkov [Glu61] is a *natural* representation of the regular expression, because it can be described in terms of the Brzozowski derivatives [Brz64] of the expression. Moreover, the Glushkov construction also plays a significant role in the document processing area: The SGML standard [ISO86], now widely adopted by publishing houses and government agencies for the syntactic specification of textual markup systems, uses *deterministic* regular expressions, i.e. expressions whose Glushkov automaton is deterministic, as a description language for document types.

In this paper, we first show that the Glushkov automaton can be constructed in time quadratic in the size of the expression, and that this is worst case optimal. For deterministic expressions, our algorithm has even linear run time. This improves on the cubic time methods suggested in the literature [BEGO71, ASU86, BS86]. A major step of the algorithm consists in bringing the expression into what we call *star normal form*. This concept is also useful for characterizing the relationship between two types of unambiguity that have been studied in the literature. Namely, we show that, modulo a technical condition, an expression is strongly unambiguous [SS88] if and only it is weakly unambiguous [BEGO71] and in star normal form. This leads to our third result, a quadratic time decision algorithm for weak unambiguity, that improves on the bi-quadratic method introduced by Book et al. [BEGO71].

*Institut für Informatik, Universität Freiburg, Rheinstr. 10–12, 7800 Freiburg, Germany.

1 Introduction

Regular expressions play a prominent role in practical applications. In syntactic specifications of programming languages they describe lexical tokens, and in text manipulation systems they describe textual patterns that trigger processing actions [ASU86, DO87]. They have become the basis of standard utilities such as the scanner generator lex and the text tools awk and egrep [AKW88, Sta89]. Regular expressions provide an appropriate notation for regular languages in text-based user interfaces, whereas finite automata are the preferred data structure for programming purposes.

Two distinct methods have been devised to translate a regular expression into a nondeterministic finite automaton (NFA). In a two-step approach, the standard method first translates a regular expression in linear time into a non-deterministic finite automaton with ϵ-transitions. Then the ϵ-transitions are eliminated in quadratic time [HU79, ASU86, Woo87, SS88].

The alternative method formalizes the notion of a symbol in a word being matched by an occurrence of the symbol in the expression [Glu61, BEGO71, ASU86, BS86]. It is based on the fact that, if a word is denoted by an expression, it must be possible to spell out that word by tracing an appropriate "path" through the expression. For example, the word $abba$ is denoted by the expression $a(a+b)^*a$ because it corresponds to the path that starts at the first a in $a(a+b)^*a$, then visits the b twice and finally arrives at the third occurrence of a. Of course, the structure of the expression restricts the positions adjacent symbols of a word can be matched with. For instance, if the i^{th} symbol in a word is matched by the second a in $a(a+b)^*a$, then the $i+1^{\text{st}}$ symbol cannot be matched with the first a. These restrictions were first formalized by Glushkov [Glu61].

It has been noticed by a number of authors that a regular expression E defines in a natural way an NFA M_E, the *Glushkov automaton* of E, whose states correspond to the occurrences of symbols in E and whose transitions connect positions that can be consecutive on a path through E [BEGO71, ASU86]. Recently, Berry and Sethi have shown that the Glushkov construction M_E is related in a natural way to the Brzozowski derivatives of E [Brz64, BS86].

None of the cited papers considers, however, the time complexity of constructing M_E. A straightforward implementation takes time cubic in the size of the expression, as opposed to the quadratic time of the standard construction.

In this paper we provide a quadratic time algorithm (Theorem 3.6) that is worst case optimal and output sensitive. To this end, we first transform an expression E in linear time into an expression $E^{\bullet}$ in what we denote by *star normal form* whose Glushkov automaton is identical to M_E (Theorem 3.1). Then we show how, for expressions in star normal form, the Glushkov automaton can be constructed in quadratic time (Lemma 2.3).

For some practical applications full regular expressions are considered too powerful and syntactic restrictions are imposed. Well known examples are the Unix tools vi and grep [DO87]. In the text processing area, the ISO Standard for SGML (Standard General Markup Language) provides a syntactic meta-language for the definition of textual markup systems. Such markup systems facilitate the electronic interchange of electronic documents and provide a standard basis for accessing and displaying them. In the SGML context, the only valid regular expressions are those for which the Glushkov automaton is deterministic. The languages recognized by *deterministic* regular expressions have been characterized [BW91]. Here we show that for a deterministic expression a *deterministic* finite automaton can be constructed in linear time (Theorem 3.7). This implies that parsing tables of linear size can be generated for the context-free grammars SGML uses to describe document types.

When transforming language descriptions from one type to another, say from regular expressions to finite automata, it is, from an applications point of view, important to preserve unambiguity, since only for unambiguous representations of a language can the meaning of a word in the language be derived from the representation. Indeed, this was the motivation for Book et al. to investigate the NFA M_E. They showed that a regular expression E is unambiguous if and only if M_E is unambiguous.

An ϵ-NFA M is *unambiguous* if for each word w there is at most one path through the state diagram of M that spells out w [ASU86]. A regular expression E is *unambiguous* if, for each word w, there is at most one path through E that matches w [BEGO71]. Thus, in unambiguous ϵ-NFAs semantic procedures can be attached to transitions, and in unambiguous regular expressions semantic procedures can be attached to occurrences of symbols.

We call the kind of ambiguity for regular expressions as defined above *weak*, as opposed to another definition given by Sippu and Soisalon-Soininen [SS88]. Their *strong* unambiguity allows semantic procedures to be attached not only to the symbols but also to the operators in a regular expression. To give an example, the expression $(a^* + b^*)^*$ is trivially weakly unambiguous because each symbol occurs only once. Thus, any symbol in a word can be matched by exactly one position in the expression. In contrast, the word aa is denoted by $(a^* + b^*)^*$ as a single application of the outer star and a two-fold application of the inner one, or, alternatively, as a two-fold application of the outer star and two single applications of the inner one. Thus, $(a^* + b^*)^*$ is *not* strongly unambiguous.

The two notions of unambiguity are related via our notion of star normal form. In Theorem 4.4 we show that, essentially, an expression is strongly unambiguous if and only if it is weakly unambiguous and in star normal form.

Finally, we turn to the decision problem for weak unambiguity. Unambiguity of ϵ-NFAs can be reduced in linear time to the LR(0) property for context-free grammars, which has quadratic time complexity [SS88]. Strong unambiguity of expressions can be reduced in linear time to unambiguity of ϵ-NFAs via Thompson's construction [ASU86]. Thus, there is a quadratic time algorithm to decide whether an expression is strongly unambiguous. On the other hand, weak unambiguity can as well be reduced to unambiguity of NFAs via the Glushkov construction, but because the reduction is quadratic in time and size, this yields a bi-quadratic decision algorithm for weak unambiguity. Alternatively, Book et al. suggest testing a regular expression E for weak ambiguity by transforming M_E into a Mealy automaton that is then tested for information losslessness. The latter can be done using an algorithm by Huffman or Evans, given, for example, in the textbook of Hennie [Hen68]. This algorithm boils down to testing, for any two different states p and q of M_E that can be reached from the initial state by means of a common word w, whether there is a state r and transitions from p to r and q to r on a common symbol a. Essentially, M_E and, hence, E is (weakly) unambiguous if no such pair of states can be found. A straight-forward implementation of this algorithm is bi-quadratic in the size of E, too.

Applying the technique developed for the quadratic time construction of the Glushkov automaton, we can transform a regular expression E into $E^\bullet$ in star normal form in linear time. This transformation preserves weak unambiguity and, for expressions in star normal form, weak and strong unambiguity are essentially the same. Thus, we provide the first algorithm for deciding weak unambiguity in quadratic time (Theorem 4.6).

2 Definitions

In this section, we define the Glushkov NFA M_E for a regular expression E. A straight-forward implementation of the construction runs in time cubic in the size of E. We show that the implementation can be modified to run in quadratic time, provided that E is in star normal form. In the next section, we show that a regular expression can be transformed into star normal form, in linear time, while leaving the Glushkov automaton intact. Together, this implies that the Glushkov automaton can be constructed from an expression in quadratic time.

Let Σ be a finite alphabet of symbols. Uppercase letters such as E, F, and G denote regular expressions and $\mathrm{L}(E)$ denotes the language specified by a regular expression E. To indicate different positions or occurrences of the same symbols in an expression, we mark symbols with subscripts. For example, the regular expression $(a+b)^*a(ab)^*$ is written as $(a_1+b_1)^*a_2(a_3b_2)^*$. With this approach the subscripted symbols a_i and b_j are called *positions* and the set of subscripted symbols in an expression E written in this form is denoted by $pos(E)$. Subscripting implies for expressions $F+G$ and FG that $pos(F)$ and $pos(G)$ are disjoint.

We use x, y, z as variables for positions and a, b, c for elements of Σ. Finally, for a position x let $\chi(x)$ be the corresponding symbol of Σ.

The following two definitions are due to Glushkov [Glu61], who used them to define a DFA recognizing $\mathrm{L}(E)$. Three functions capture the notion of a position in a regular expression matching a symbol in a word. These functions are: $\mathit{first}(E)$, the set of positions that match the first symbol of some word in $\mathrm{L}(E)$; $\mathit{last}(E)$, the dual set for last positions and symbols; and $\mathit{follow}(E, x)$, the set of positions that can follow position x in a path through E.

Definition 2.1 We can define $\mathit{first}(E)$ and $\mathit{last}(E)$ inductively.

$$[E = \epsilon \text{ or } \emptyset] \quad \mathit{first}(E) = \mathit{last}(E) = \emptyset.$$

$$[E = x] \quad \mathit{first}(E) = \mathit{last}(E) = \{x\}.$$

$$[E = F + G] \quad \mathit{first}(E) = \mathit{first}(F) \cup \mathit{first}(G), \quad \mathit{last}(E) = \mathit{last}(F) \cup \mathit{last}(G).$$

$$[E = FG] \quad \mathit{first}(E) = \begin{cases} \mathit{first}(F) \cup \mathit{first}(G) & \text{if } \epsilon \in \mathrm{L}(F), \\ \mathit{first}(F) & \text{otherwise,} \end{cases}$$
$$\mathit{last}(E) = \begin{cases} \mathit{last}(F) \cup \mathit{last}(G) & \text{if } \epsilon \in \mathrm{L}(G), \\ \mathit{last}(G) & \text{otherwise.} \end{cases}$$

$$[E = F^*] \quad \mathit{first}(E) = \mathit{first}(F),\ \mathit{last}(E) = \mathit{last}(F).$$

Definition 2.2 The function $\mathit{follow}(E, .)$ maps positions of E to subsets of positions of E.

$[E = \epsilon \text{ or } \emptyset]$ E has no positions.

$$[E = x] \quad \mathit{follow}(E, x) = \emptyset.$$

$$[E = F + G] \quad follow(E,x) = \begin{cases} follow(F,x) & \text{if } x \in pos(F), \\ follow(G,x) & \text{if } x \in pos(G). \end{cases}$$

$$[E = FG] \quad follow(E,x) = \begin{cases} follow(F,x) & \text{if } x \in pos(F) \setminus last(F), \\ follow(F,x) \cup first(G) & \\ & \text{if } x \in last(F), \\ follow(G,x) & \text{if } x \in pos(G). \end{cases}$$

$$[E = F^*] \quad follow(E,x) = \begin{cases} follow(F,x) & \text{if } x \in pos(F) \setminus last(F), \\ follow(F,x) \cup first(F) & \\ & \text{if } x \in last(F). \end{cases}$$

Using the functions *first*, *last*, and *follow* several authors have defined the Glushkov NFA M_E recognizing $\mathrm{L}(E)$ [BEGO71, ASU86, BS86]. Berry and Sethi have shown that M_E is a natural representation of E [BS86].

Definition 2.3 We define the Glushkov automaton

$$M_E = (Q_E \dot{\cup} \{q_I\}, \Sigma, \delta_E, q_I, F_E)$$

as follows.

1. $Q_E = pos(E)$, i.e. the states of M_E are the positions of E plus a new, initial state, q_I.
2. $\delta_E(s_I) = first(E)$.
3. $\delta_E(x,a) = \{y \mid y \in follow(E,x),\ \chi(y) = a\}$, for $x \in pos(E)$, $a \in \Sigma$.
4. $F_E = \begin{cases} last(E) \cup \{s_I\} & \text{if } \epsilon \in \mathrm{L}(M_E), \\ last(E) & \text{otherwise.} \end{cases}$

Proposition 2.1 $\mathrm{L}(M_E) = \mathrm{L}(E)$.

The inductive definition suggests a computation of *first*, *last*, and *follow* that is cubic in the size of E. First, we describe this canonical method. Then, we refine the method to achieve quadratic time complexity.

Let n be the size of E. We begin by converting E into a syntax tree. The external nodes are labeled with $\emptyset$, ϵ, and the occurrences of symbols, and the internal nodes are labeled with one of the operators $+$, $\cdot$, or *. Since the regular expressions are generated by an LL(1) grammar, this can be done in time $O(n)$ [HU79]. Each node ν of the syntax tree corresponds to a subexpression E_ν of E.

At each node ν of the syntax tree we provide variables

nullable(ν): Boolean;
first(ν), *last*(ν): $2^{pos(E)}$;
follow(ν, x): $2^{pos(E)}$;

for $x \in pos(E_\nu)$. The variable *nullable*(ν) indicates whether the subexpression E_ν corresponding to ν contains the empty word, *first*(ν) and *last*(ν) hold the first and last positions of E_ν, and *follow*(ν, x) holds the positions of E_ν following x in E_ν.

We perform a postorder traversal of the syntax tree and at each node ν, the variables for ν are computed. More precisely, at each node ν the following code is executed.

case
    ν is a node labeled $\emptyset$:
        nullable(ν) := *false*;
        first$(\nu) := \emptyset$;
        last$(\nu) := \emptyset$;
    ν is a node labeled ϵ:
        nullable(ν) := *true*;
        first$(\nu) := \emptyset$;
        last$(\nu) := \emptyset$;
    ν is a node labeled x:
        nullable(ν) := *false*;
        follow$(\nu, x) := \emptyset$;
        first$(\nu) := \{x\}$;
        last$(\nu) := \{x\}$;
    ν is a node labeled $+$:
        nullable(ν) := *nullable*(*leftchild*) **or** *nullable*(*rightchild*);

```
        first(ν) := first(leftchild) ∪ first(rightchild);                (⋆)
        last(ν) := last(leftchild) ∪ last(rightchild);                   (⋆)
    ν is a node labeled ·:
        nullable(ν) := nullable(leftchild) and nullable(rightchild);
        for each x in last(left child) do
            follow(ν, x) := follow(leftchild, x) ∪ first(rightchild);    (⋆⋆)
        if nullable(left child) then
            first(ν) := first(leftchild) ∪ first(rightchild)             (⋆)
        else
            first(ν) := first(leftchild);
        if nullable(right child) then
            last(ν) := last(leftchild) ∪ last(rightchild)                (⋆)
        else
            last(ν) := last(rightchild);
    ν is a node labeled *:
        nullable(ν) := true;
        for each x in last(child) do
            follow(ν, x) := follow(child, x) ∪ first(child);             (⋆⋆⋆)
        first(ν) := first(child);
        last(ν) := last(child);
end case;
```

Lemma 2.2 *The following invariant holds after node ν has been visited.*

1. *$nullable(\nu)$ is true if and only if $\epsilon \in \mathrm{L}(E_\nu)$.*
2. *$first(\nu) = first(E_\nu)$, $last(\nu) = last(E_\nu)$.*
3. *$follow(\nu, x) = follow(E_\nu, x)$ for $x \in pos(E_\nu)$.*

Especially, for the root note ν_0

1. *$first(\nu_0) = first(E)$, $last(\nu_0) = last(E)$.*
2. *$follow(\nu_0, x) = follow(E, x)$, for $x \in pos(E)$.*

If sets are represented as ordered lists, then the union of two sets can be implemented in time linear in the size of the sets. Since all sets are at most of size n, the algorithm to compute $\mathit{first}(E)$, $\mathit{last}(E)$, and $\mathit{follow}(E,x)$, for $x \in \mathit{pos}(E)$, takes time $\mathrm{O}(n^3)$.

The first observation on the way to a better time bound is that all unions labeled $(\star)$ or $(\star\star)$ are disjoint. This is, because $\mathit{pos}(F) \cap \mathit{pos}(G) = \emptyset$ if $F+G$ or FG are subexpressions of E. Only the unions labeled $(\star\star\star)$ are not necessarily disjoint. A starred subexpression H^* of E adds the elements of $\mathit{first}(H)$ to $\mathit{follow}(H,x)$ for $x \in \mathit{last}(H)$, but some elements of $\mathit{first}(H)$ may already belong to $\mathit{follow}(H,x)$ for some $x \in \mathit{last}(H)$, as the expression $(a^*b^*)^*$ illustrates.

Our general strategy is as follows: We only consider expressions for which all unions, including the ones of type $(\star\star\star)$, are disjoint. Such expressions are in star normal form. Then we show that our algorithm runs in time $\mathrm{O}(\mathrm{size}(M_E))$ for expressions E in star normal form. Finally, in the next section, we show why the restriction to star normal form is justified.

Definition 2.4 A regular expression E is in *star normal form* if for each starred subexpression H^* of E the *SNF-condition* holds, namely

$$\mathit{follow}(H, \mathit{last}(H)) \cap \mathit{first}(H) = \emptyset.$$

Lemma 2.3 *Let E be a regular expression in star normal form. Then, M_E can be computed from E in time $\mathrm{O}(\mathrm{size}(E) + \mathrm{size}(M_E))$.*

Proof Let E be in star normal form. First, let us look at the unions labeled $(\star)$. They have the general form $X := Y \cup Z$ where Y and Z are disjoint. Furthermore, Y and Z will never again be referred to by the program. Thus, we can represent sets as unordered lists and we can implement the union in constant time as list concatenation without copying, possibly destroying the binding of Y and Z to its values in the process.

The unions of type $(\star\star)$ and $(\star\star\star)$ also have the form $X := Y \cup Z$ where Y and Z are disjoint. In these cases, Z is referred to several times in a **for**-loop and, thus, must be preserved. Hence, we implement the union as copying the elements of Z one by one to the end of Y. The run time is proportional to the size of Z.

Finally, we have to estimate the run time of the algorithm against the size of M_E. The crucial observation is that for any subexpression F of a subexpression G of E and for any $x \in pos(F)$, we have

$$follow(F, x) \subseteq follow(G, x) \subseteq follow(E, x).$$

Since all unions are disjoint, the run time spent with instruction $(\star\star)$ or $(\star\star\star)$ in a node ν and for a position x is proportional to the number of positions in $follow(E_\nu, x)$ that are not present in any of the subexpressions of E_ν. Thus, the total run time spent with $(\star\star)$ and $(\star\star\star)$ is proportional to

$$\sum_{x \in pos(E)} |follow(E, x)|,$$

which is less or equal to the number of transitions in M_E. □

3 The star normal form

The goal of this section is to transform a regular expression E, in linear time, into an expression $E^\bullet$ in star normal form such that $M_E = M_{E^\bullet}$ holds.

Theorem 3.1 *For each regular expression E, there is a regular expression $E^\bullet$ such that*

1. $M_{E^\bullet} = M_E$.
2. *$E^\bullet$ is in star normal form.*
3. *$E^\bullet$ can be computed from E in linear time.*

As an intermediate step, we show that a starred expression E^* can be transformed into an expression $E^{\circ *}$ with identical Glushkov automaton, such that the SNF-condition of Definition 2.4 is fulfilled at least at the outermost level, namely for E°. The crucial observation is that, if we remove from M_E all "feedback" transitions leading from final states (apart from q_I) to states that q_I is directly connected to, and if we make s_I non-final, then the resulting NFA is the Glushkov automaton of an expression E° with $\mathit{follow}(E^\circ, \mathit{last}(E^\circ)) \cap \mathit{first}(E^\circ) = \emptyset$. Furthermore, all "feedback" transitions deleted from M_E in M_{E° are re-introduced in $M_{E^{\circ *}}$. Thus, we have $M_{E^{\circ *}} = M_{E^*}$.

Definition 3.1 We define E° inductively as follows.

$[E = \emptyset \text{ or } \epsilon]$ Let $E^\circ = \emptyset$.

$[E = a]$ Let $E^\circ = E$.

$[E = F + G]$ $E^\circ = F^\circ + G^\circ$.

$$[E = FG] \quad E^\circ = \begin{cases} FG & \text{if } \epsilon \notin \mathrm{L}(F),\ \epsilon \notin \mathrm{L}(G). \\ F^\circ G & \text{if } \epsilon \notin \mathrm{L}(F),\ \epsilon \in \mathrm{L}(G). \\ FG^\circ & \text{if } \epsilon \in \mathrm{L}(F),\ \epsilon \notin \mathrm{L}(G). \\ F^\circ + G^\circ (!) & \text{if } \epsilon \in \mathrm{L}(F),\ \epsilon \in \mathrm{L}(G). \end{cases}$$

$[E = F^*]$ $E^\circ = F^\circ$.

Lemma 3.2

1. $\mathrm{length}(E^\circ) \leq \mathrm{length}(E)$.
2. $\epsilon \notin \mathrm{L}(E^\circ)$.
3. $\mathit{pos}(E^\circ) = \mathit{pos}(E)$.
4. $\mathit{first}(E^\circ) = \mathit{first}(E)$,
   $\mathit{last}(E^\circ) = \mathit{last}(E)$.
5. $\mathit{follow}(E^\circ, x) = \mathit{follow}(E, x)$, *for all* $x \in \mathit{pos}(E) \setminus \mathit{last}(E)$.
6. $\mathit{follow}(E^\circ, x) = \mathit{follow}(E, x) \setminus \mathit{first}(E)$, *for all* $x \in \mathit{last}(E)$, *especially* $\mathit{follow}(E^\circ, \mathit{last}(E^\circ)) \cap \mathit{first}(E^\circ) = \emptyset$.
7. $\mathit{follow}(E^{\circ *}, x) = \mathit{follow}(E^*, x)$, *for all* $x \in \mathit{pos}(E)$.
8. $M_{E^*} = M_{E^{\circ *}}$.

Proof The first claims are straightforward inductions on E. Claims 5 and 6 are proved by induction on E. We only show the induction step for concatenation.

$[E = FG]$ Case 1: $\epsilon \notin \mathrm{L}(F)$, $\epsilon \notin \mathrm{L}(G)$.
follow(E, x) and *first*(E) are disjoint for any position $x \in$ *last*(E). This implies that *follow*$(E^\circ, x) =$ *follow*$(E, x) =$ *follow*$(E, x) \setminus$ *first*(E).

Case 2: $\epsilon \notin \mathrm{L}(F)$, $\epsilon \in \mathrm{L}(G)$.
For any $x \in$ *last*(G), again *follow*(E, x) and *first*(E) are disjoint because $\epsilon \notin \mathrm{L}(F)$. The other cases follow from the induction hypothesis.

Case 3: $\epsilon \in \mathrm{L}(F)$, $\epsilon \notin \mathrm{L}(G)$.
last$(E^\circ) =$ *last*(G°) because $\epsilon \notin \mathrm{L}(G^\circ)$.

Case 4: $\epsilon \in \mathrm{L}(F)$, $\epsilon \in \mathrm{L}(G)$.
This case follows directly from the induction hypothesis.

Claims 7 and 8 follow directly from 5 and 6. □

Substituting an expression H^* with $H^{\circ *}$ leaves the Glushkov automaton of H^* intact. Furthermore,

$$\mathit{follow}(H^\circ, \mathit{last}(H^\circ)) \cap \mathit{first}(H^\circ) = \emptyset.$$

Thus, if we substitute in E each starred subexpression H^* with $H^{\circ *}$, proceeding bottom up in E, we can expect to get an expression $E^\bullet$ in star normal form with $M_E = M_{E^\bullet}$.

Definition 3.2

$[E = \emptyset, \epsilon, \text{or } a]$ $E^\bullet = E$.

$[E = F + G]$ $E^\bullet = F^\bullet + G^\bullet$

$[E = FG]$ $E^\bullet = F^\bullet G^\bullet$.

[$E = F^*$] $E^\bullet = F^{\bullet\circ *}$.

Lemma 3.3

1. $\mathrm{L}(E) = \mathrm{L}(E^\bullet)$.
2. $\mathrm{length}(E^\bullet) \leq \mathrm{length}(E)$.
3. $pos(E^\bullet) = pos(E)$.
4. $first(E^\bullet) = first(E)$,
   $last(E^\bullet) = last(E)$.
5. $follow(E^\bullet, x) = follow(E, x)$, *for* $x \in pos(E)$.
6. $s_I \in F_{E^\bullet}$ *if and only if* $s_I \in F_E$.

Claims 4 and 5 imply the first part of Theorem 3.1, namely $M_{E^\bullet} = M_E$.

Our next claim is that $E^\bullet$ is in star normal form. The proof is by induction on the length of E. The interesting case is the star in the induction step.

[$E = F^*$] We have $E^\bullet = F^{\bullet\circ *}$. The SNF-condition holds for $F^{\bullet\circ}$ (Lemma 3.2), and the induction hypothesis implies that $F^{\circ\bullet}$ is in star normal form. To complete the proof, we only have to show that $F^{\circ\bullet} = F^{\bullet\circ}$.

Lemma 3.4

1. $E^{\circ\circ} = E^\circ$.
2. $E^{\bullet\circ} = E^{\circ\bullet}$.
3. $E^{\bullet\bullet} = E^\bullet$.

Proof By induction on E. We show the induction step for the star.

[$E = F^*$] 1. $E^{\circ\circ}$ is identical to $F^{\circ\circ}$ by definition, which, in turn, is F° by induction hypothesis or E° by definition.

2. $E^{\bullet\circ} = F^{\bullet\circ*\circ} = F^{\bullet\circ\circ}$ by definition. Claim 1 gives $F^{\bullet\circ}$, which is $F^{\circ\bullet}$ by the induction hypothesis, or $E^{\circ\bullet}$ by definition.

3. $E^{\bullet\bullet} = F^{\bullet\circ*\bullet} = F^{\bullet\circ\bullet\circ*}$ by definition. Claim 2 gives $F^{\bullet\bullet\circ\circ*}$, which is $F^{\bullet\circ*}$ by the induction hypothesis and Claim 1. This, in turn, is $E^{\bullet}$ by definition.

$\square$

Finally, we show that $E^{\bullet}$ can be computed from E in linear time. $E^{\bullet}$ is built up from $H^{\bullet}$ and $H^{\bullet\circ}$ for subexpressions H of E. Thus, we compute $H^{\bullet}$ and $H^{\bullet\circ}$ simultaneously, during a postorder traversal through the syntax tree of E. The following lemma, together with the recursive definition of $E^{\bullet}$, makes sure that at each node only a constant amount of time is spent. This completes the proof of Theorem 3.1.

Lemma 3.5

$$[E = \emptyset \text{ or } \epsilon] \quad \emptyset^{\bullet\circ} = \emptyset = \epsilon^{\bullet\circ}.$$

$$[E = a] \quad E^{\bullet\circ} = E.$$

$$[E = F + G] \quad E^{\bullet\circ} = F^{\bullet\circ} + G^{\bullet\circ}.$$

$$[E = FG] \quad E^{\bullet\circ} = \begin{cases} F^{\bullet}G^{\bullet} & \textit{if } \epsilon \notin \mathrm{L}(F),\ \epsilon \notin \mathrm{L}(G). \\ F^{\bullet\circ}G^{\bullet} & \textit{if } \epsilon \notin \mathrm{L}(F),\ \epsilon \in \mathrm{L}(G). \\ F^{\bullet}G^{\bullet\circ} & \textit{if } \epsilon \in \mathrm{L}(F),\ \epsilon \notin \mathrm{L}(G). \\ F^{\bullet\circ} + G^{\bullet\circ} & \textit{if } \epsilon \in \mathrm{L}(F),\ \epsilon \in \mathrm{L}(G). \end{cases}$$

$$[E = F^*] \quad E^{\bullet\circ} = F^{\bullet\circ}.$$

Proof By induction on E. In the concatenation step one has to observe that $\mathrm{L}(E) = \mathrm{L}(E^{\bullet})$, and in the star step one has to apply Lemma 3.4. $\square$

Example By definition, we have

$$(a^*b^*)^{*\bullet} = (a^*b^*)^{\bullet\circ*}.$$

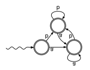

Figure 1 The Glushkov automaton corresponding to $(a^*b^*)^*$ and its star normal form $(a+b)^*$.

Repeated application of Lemma 3.5 yields

$$\begin{aligned}(a^*b^*)^{\bullet\circ *} &= (a^{*\bullet\circ} + b^{*\bullet\circ})^* \\ &= (a^{\bullet\circ} + b^{\bullet\circ})^* \\ &= (a+b)^*.\end{aligned}$$

Hence, $(a+b)^*$ is the star normal form of $(a^*b^*)^*$. Both expressions have the same Glushkov NFA, which is shown in Figure 1.

Putting the results of the previous two sections together, we get:

Theorem 3.6 *The Glushkov automaton M_E can be computed from a regular expression E in time linear in* $\text{size}(E) + \text{size}(M_E)$.

Proof First, we compute $E^\bullet$ from E in linear time. According to Theorem 3.1, $E^\bullet$ fulfills the precondition of Lemma 2.3. Hence, the NFA $M_{E^\bullet}$ can be computed from $E^\bullet$ in time linear in $\text{size}(E^\bullet) + \text{size}(M_{E^\bullet})$. But $M_{E^\bullet}$ is identical to M_E. □

Since each transition leading to a state $x \in pos(E)$ in M_E has the label $\chi(x)$, the size of M_E is quadratic in the size of E. Our result is worst case optimal, because the size of a minimal NFA equivalent to E is $\Omega(\text{size}(E)^2)$ [SS88].

People writing document grammars in the SGML context are especially interested in regular expressions whose Glushkov automaton is a DFA. For such expressions, the Glushkov automaton can be constructed in linear time.

Definition 3.3 A regular expression E is *deterministic* if the corresponding NFA M_E is deterministic.

Theorem 3.7 *It can be decided in linear time whether a regular expression E is deterministic. If E is deterministic, then the deterministic finite automaton M_E can be computed from E in linear time.*

Proof For a deterministic expression E, the size of M_E is linear in the size of E. $\square$

4 Ambiguity in automata and expressions

Two types of unambiguity of regular expressions have been defined in the literature. An expression E is weakly unambiguous [BEGO71] if each word of E can be traced uniquely with a path through E, whereas E is strongly unambiguous [SS88] if each word of E can be uniquely decomposed into subwords according to the syntactic structure of E. The relationship between the two concepts of unambiguity has not been investigated so far. It turns out in this section, that the missing link is the star normal form defined above. Thus, modulo a technical condition on the empty word, an expression E is strongly unambiguous if and only if it is weakly unambiguous and in star normal form (Theorem 4.4).

First, we define weak and strong unambiguity. From now on we consider only regular expressions that don't use $\emptyset$ as a syntactic constituent and, hence, consider only non-empty regular languages.

Definition 4.1

1. An ϵ-NFA M is *unambiguous* if, for each word w, there is at most one path from the initial state to a final state that spells out w.

2. A regular expression E is *weakly unambiguous* if and only if the NFA M_E is unambiguous.

Note that a path through M_E is uniquely determined by the sequence $x_1, \ldots, x_n$ of positions in $pos(E)$ it passes through because all transitions leading to state $x \in pos(E)$ are labeled with $\chi(x)$ and no transition in M_E leads to the initial state.

Definition 4.2 We define for languages L, L':

1. The concatenation of L, L' is *unambiguous* if $v, w \in L$, $v', w' \in L'$, and $vv' = ww'$ imply $v = w$ and $v' = w'$.
2. The star of L is *unambiguous* if $v_1, \ldots, v_m \in L$, $w_1, \ldots, w_n \in L$, $m, n \geq 0$, and $v_1 \ldots v_m = w_1 \ldots w_n$ imply $m = n$ and $v_i = w_i$ for $1 \leq i \leq m$.

Definition 4.3 We define inductively when a regular expression E is *strongly unambiguous*.

[$E = \epsilon$ or a] E is strongly unambiguous.

[$E = F + G$] E is strongly unambiguous if F and G are strongly unambiguous and $\mathrm{L}(F)$ and $\mathrm{L}(G)$ are disjoint.

[$E = FG$] E is strongly unambiguous if F and G are strongly unambiguous and the concatenation of $\mathrm{L}(F)$ and $\mathrm{L}(G)$ is unambiguous.

[$E = F^*$] E is strongly unambiguous if F is strongly unambiguous and the star of $\mathrm{L}(F)$ is unambiguous.

Strong unambiguity can be defined in terms of automata as well.

Definition 4.4 Let M'_E be the ϵ-NFA recognizing $\mathrm{L}(E)$ according to any of the standard textbook constructions [ASU86, HU79, Woo87, SS88, AO83].

Lemma 4.1 *E is strongly unambiguous if and only if M'_E is unambiguous.*

Proof Sippu and Soisalon-Soininen have shown this for their construction [SS88]. The other variants are similar. □

Lemma 4.2 *If E is strongly unambiguous, then E is weakly unambiguous.*

Proof Elimination of ϵ-transitions using the algorithm of Sippu and Soisalon-Soininen [SS88] transforms M'_E into M_E. Thus, different paths in M_E spelling out a word w correspond to different paths in M'_E doing the same. Therefore, unambiguity of M'_E implies unambiguity of M_E. Now Lemma 4.1 can be applied. □

Now we investigate under which circumstances weakly unambiguous expressions are also strongly unambiguous. A direct comparison is facilitated through the following inductive definition of weak unambiguity.

Lemma 4.3

[$E = \epsilon$ or a] *E is weakly unambiguous.*

[$E = F + G$] *E is weakly unambiguous if and only if F and G are weakly unambiguous and at most the empty word ϵ is both in $\mathrm{L}(F)$ and $\mathrm{L}(G)$.*

[$E = FG$] *E is weakly unambiguous if and only if F and G are weakly unambiguous and the concatenation of $\mathrm{L}(F)$ and $\mathrm{L}(G)$ is unambiguous.*

[$E = F^*$] *Let $\mathit{follow}(F, \mathit{last}(F)) \cap \mathit{first}(F) = \emptyset$, $\epsilon \notin \mathrm{L}(F)$. Then, E is weakly unambiguous if and only if F is weakly unambiguous and the star of $\mathrm{L}(F)$ is unambiguous.*

Proof

[$E = F + G$] Since Glushkov automata have no ϵ-transitions, the only path denoting the empty word is the empty path. Furthermore, any path through F or through G is also a path through E, and any non-empty path through F is different from any path through E.

[$E = FG$] Let's assume that E is weakly unambiguous. Since $\mathrm{L}(F)$ and $\mathrm{L}(G)$ are non-empty, each path through F or G can be completed to a path through E. Thus, F and G are weakly unambiguous. Each decomposition of a word $w \in \mathrm{L}(F)\mathrm{L}(G)$ in the form $w = vw = v'w'$, $v, v' \in \mathrm{L}(F)$, $w, w' \in \mathrm{L}(G)$, corresponds to paths $x_1 \ldots x_m y_1 \ldots y_n$ and $x'_1 \ldots x'_{m'} y'_1 \ldots y'_{n'}$ of E, where the x-positions belong to F and the y-positions to G. Since E is weakly unambiguous, the paths through E are identical. Since the positions of F and G are disjoint, we have $m = m'$ and $n = n'$, i.e. $v = v'$, $w = w'$. Thus, the concatenation of $\mathrm{L}(F)$ and $\mathrm{L}(G)$ is unambiguous.

This proves one direction, the other one is obvious.

$[E = F^*]$ Since $\epsilon \notin \mathrm{L}(E)$, the empty word is uniquely decomposed into a sequence of words in $\mathrm{L}(F)$.

Any non-empty path through M_E is determined by a sequence of positions $x_1, \ldots, x_n$, $n \geq 1$, which consists of a sequence of paths through M_F. Because F fulfills the SNF-condition, the starting positions of those paths are uniquely determined. Hence, if E is weakly unambiguous, then the star of F is unambiguous.

Again, the other direction is obvious.

□

Thus, weak and strong unambiguity have exactly the same inductive definition for expressions E in star normal form, provided that no subexpression of E denotes the empty word ambiguously. We call the last condition *epsilon normal form.*

Definition 4.5 We define by induction on E when E is in *epsilon normal form.*

$[E = \epsilon \text{ or } a]$ E is in epsilon normal form.

$[E = F + G]$ E is in epsilon normal form if F and G are in epsilon normal form and $\epsilon \notin \mathrm{L}(F) \cap \mathrm{L}(G)$.

[$E = FG$] E is in epsilon normal form if F and G are in epsilon normal form.

[$E = F^*$] E is in epsilon normal form if F is in epsilon normal form and $\epsilon \notin \mathrm{L}(F)$.

Theorem 4.4 *E is strongly unambiguous if and only if*

1. *E is weakly unambiguous,*
2. *E is in star normal form, and*
3. *E is in epsilon normal form.*

Proof Lemma 4.3 implies that for expressions in star and epsilon normal form, weak and strong unambiguity are identical. It remains to show that strongly unambiguous expressions are in star and in epsilon normal form. The crucial point in the induction is dealt with in the next lemma. $\square$

Lemma 4.5 *If E^* is strongly unambiguous, then E fulfills the SNF-condition.*

Proof We assume that there exist $x \in \mathit{last}(E)$, $y \in \mathit{follow}(E, x) \cap \mathit{first}(E)$. Since E does not contain $\emptyset$ as a syntactic constituent, the final state x of M_E can be reached in M_E from the initial state s_I via intermediate states $x_1, \ldots, x_n$, $n \geq 0$, and some final state $z \in \mathit{last}(E)$ can be reached from y via intermediate states $y_1, \ldots, y_m$, $m \geq 0$. Now the state sequence $x_1, \ldots, x_n, x, y, y_1, \ldots, y_m, z$ describes a path through M_E, because $y \in \mathit{follow}(E, x)$. But this path is also the composition of two paths through M_E, because $x \in \mathit{last}(E)$, $y \in \mathit{first}(E)$. This makes the star of $\mathrm{L}(E)$ ambiguous. $\square$

In the previous section, we have transformed expressions into star normal form, in linear time. This transformation preserves epsilon normal form. Thus, Theorem 4.4 reduces weak unambiguity to strong unambiguity in linear time. This yields a quadratic decision algorithm for weak unambiguity of expressions in epsilon normal form.

Theorem 4.6 *Regular expressions in epsilon normal form can be tested for weak unambiguity in quadratic time.*

Proof Let E be in epsilon normal form. E can be transformed into star normal form $E^\bullet$ without changing the Glushkov automaton, in linear time. Furthermore, $E^\bullet$ is also in epsilon normal form.

Unfortunately, it is possible that $E^\bullet$ contains $\emptyset$ as a syntactic constituent, even if E does not. The usual linear time elimination of $\emptyset$ from $E^\bullet$, however, resulting in an expression $E^{\bullet\emptyset}$, preserves star and epsilon normal form and leaves the Glushkov automaton intact, i.e. $M_E = M_{E^\bullet} = M_{E^{\bullet\emptyset}}$. Now Theorem 4.4 can be applied to the $\emptyset$-free expression $E^{\bullet\emptyset}$. Thus, E is weakly unambiguous if and only if $E^{\bullet\emptyset}$ is, i.e. if and only if $E^{\bullet\emptyset}$ is strongly unambiguous. Finally, strong unambiguity of expressions can be decided in quadratic time [SS88]. $\square$

References

[AKW88] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language.* Addison-Wesley, Reading, Massachusetts, 1988.

[AO83] Jürgen Albert and Thomas Ottmann. *Automaten, Sprachen und Maschinen.* Bibliographisches Institut, Mannheim, 1983.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools. Addison-Wesley Series in Computer Science*, Addison-Wesley, Reading, Massachusetts, 1986.

[BEGO71] Ronald Book, Shimon Even, Sheila Greibach, and Gene Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Computers*, C-20(2):149–153, February 1971.

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

[BS86] Gerard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.

[BW91] Anne Brüggemann-Klein and Derick Wood. Parser generators for document grammars. Submitted for publication, 1991.

[DO87] Dale Dougherty and Tim O'Reilly. *UNIX Text Processing.* Hayden Books, Indianapolis, 1987.

[Glu61] V.M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.

[Hen68] Frederick C. Hennie. *Finite-State Models for Logical Machines.* John Wiley, New York, 1968.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation. Addison-Wesley Series in Computer Science*, Addison-Wesley, Reading, Massachusetts, 1979.

[ISO86] ISO 8879. Information processing—text and office systems—standard generalized markup language (SGML). October 1986. International Organization for Standardization.

[SS88] Seppo Sippu and Eljas Soisalon-Soininen. *Parsing Theory.* Volume 1, Languages and Parsing, of *EATCS Monographs on Theoretical Computer Science*, Springer-Verlag, Berlin, 1988.

[Sta89] Gottfried Staubach. *UNIX-Werkzeuge zur Textmusterverarbeitung.* Springer-Verlag, Berlin, 1989.

[Woo87] Derick Wood. *Theory of Computation.* John Wiley, New York, 1987.