An Improved Algorithm for the Membership Problem for Extended Regular Expressions

Orna Kupferman* and Sharon Zuhovitzky

Hebrew University, School of Engineering and Computer Science, Jerusalem 91904, Israel
Email: {orna,sharonzu}@cs.huji.ac.il

Abstract. Extended regular expressions (ERE) define regular languages using union, concatenation, repetition, intersection, and complementation operators. The fact ERE allow intersection and complementation makes them exponentially more succinct than regular expressions. The membership problem for extended regular expressions is to decide, given an expression r and a word w, whether w belongs to the language defined by r. Since regular expressions are useful for describing patterns in strings, the membership problem has numerous applications. In many such applications, the words w are very long and patterns are conveniently described using ERE, making efficient solutions to the membership problem of great practical interest.

In this paper we introduce alternating automata with synchronized universality and negation, and use them in order to obtain a simple and efficient algorithm for solving the membership problem for ERE. Our algorithm runs in time $O(m \cdot n^2)$ and space $O(m \cdot n + k \cdot n^2)$, where m is the length of r, n is the length of w, and k is the number of intersection and complementation operators in r. This improves the best known algorithms for the problem.

1 Introduction

Regular languages of finite words are naturally defined by repeated applications of closure operators. *Regular expressions* (*RE*, for short) contain *union* ($\vee$), *concatenation* ($\cdot$), and *repetition* ($*$) operators and can define all the regular languages. It turned out that enriching RE with *intersection* ($\wedge$) and *complementation* ($\neg$) makes the description of regular languages more convenient. Indeed, there are evidences for the succinctness of *semi-extended regular expressions* (*SERE*, for short, which extend RE with intersection) with respect to RE, and for the succinctness of *extended regular expressions* (*ERE*, for short, which extend RE with both intersection and complementation) with respect to SERE. For example, specifying the regular language of all words that contain a non-overlapping repetition of a subword of length n requires an RE of length $\Omega(2^n)$ and can be done with an SERE of length $O(n^2)$ [Pet02]. Also, specifying the singleton language $\{1^n\}$ requires an SERE of length $\Theta(n)$ and can be done with an ERE of length $polylog(n)$ [KTV01]. In general, ERE are nonelementary more succinct than RE [MS73].

* Supported in part by BSF grant 9800096.

The membership problem for RE and their extensions is to decide, given an RE r and a word w, whether w belongs to the language defined by r. Since RE are useful for describing patterns in strings, the membership problem has many applications (see [KM95] for recent applications in molecular biology). In many of these applications, the words w are very long and patterns are described using SERE or ERE, making efficient solutions for the membership problem for them of great practical interest [Aho90].

Studies of the membership problem for RE have shown that an *automata-theoretic approach* is useful: a straightforward algorithm for membership in RE translates the RE to a nondeterministic automaton. For an RE of size m, the automaton is of size $O(m)$, thus for a word of length n, the algorithm runs in time $O(mn)$ and space $O(m)$ [HU79]. For SERE, a translation to nondeterministic automata involves an exponential blow-up, and until recently, the best known algorithms for membership in SERE were based on dynamic programing and ran in time $O(mn^3)$ and space $O(mn^2)$ [HU79,Hir89]. Myers describes a technique for speeding up the membership algorithms by $\log n$ [Mye92], but still one cannot expect an $O(mn)$ algorithm for membership in SERE, as the problem is LOGCFL-complete [Pet02], whereas the one for RE is NL-complete [JR91].

The translation of SERE to automata is not easy even when alternating automata are considered. The difficulty lies in expressions like $(r_1 \wedge r_2) \cdot r_3$, where the two copies of the alternating automaton that check membership of a guessed prefix in r_1 and r_2 should be synchronized in order to agree on the suffix that is checked for membership in r_3. Indeed, $(r_1 \wedge r_2) \cdot r_3$ is not equal to $(r_1 \cdot r_3) \wedge (r_2 \cdot r_3)$. In contrast, $(r_1 \vee r_2) \cdot r_3$ is equal to $(r_1 \cdot r_3) \vee (r_2 \cdot r_3)$, which is why RE can be efficiently translated to alternating automata, while SERE cannot [KTV01].

Several models of automata with *synchronization* are studied in the literature (c.f., [Hro86,Slo88,DHK$^+$89,Yam00b]). Essentially, in synchronous alternating automata, some of the states are designated as synchronization states, and the spawned copies of the automaton have to agree on positions in the input word in which they visit synchronization states. In [Yam00a], Yamamoto introduces *partially input-synchronized alternating automata*, and shows an algorithm for membership in SERE that is based on a translation of SERE into the new model. While the new algorithm improves the known algorithms – it runs in time $O(m \cdot n^2)$ and space $O(m \cdot n + k \cdot n^2)$, where k is the number of $\wedge$ operators, the model of partially input-synchronized alternating automata is needlessly complicated, and we found the algorithm that follows very cryptic and hard to implement. On the other hand, Yamamoto's idea of using alternating automata with some type of synchronization as an automata-theoretic framework for solving the membership problem for SERE seems like a very good direction.

In this paper, we introduce *alternating automata with synchronized universality and negation* (ASUN, for short). ASUN are simpler than the model of Yamamoto, but have a richer structure. In addition to the existential and universal states of usual alternating automata, ASUN have *complementation states*, which enable a dualization of the automaton's behavior. This enables us to translate an ERE of length m into an ASUN of size $O(m)$, where synchronized universality is used to handle conjunctions and synchronized negation is used to handle complementation. We show that the membership

problem for ASUN can be solved in time $O(m \cdot n^2)$ and space $O(m \cdot n + k \cdot n^2)$, where k is the number of $\wedge$ and $\neg$ operators. Thus, our bound coincides with that of Yamamoto's, but we handle a richer class of regular expressions, and the algorithm that follows is much simpler and easy to implement (in fact, we describe a detailed pseudo-code in one page).

2 Definitions

2.1 Extended Regular Expressions

Let Σ be a finite *alphabet*. A *finite word* over Σ is a (possibly empty) finite sequence $w = \sigma_1 \cdot \sigma_2 \cdots \sigma_n$ of concatenated letters in Σ. The length of a word w is denoted by $|w|$. The symbol ϵ denotes the empty word. We use $w[i, j]$ to denote the subword $\sigma_i \cdots \sigma_j$ of w. If $i > j$, then $w[i, j] = \epsilon$. *Extended Regular Expressions* (ERE) define languages by inductively applying union, intersection, complementation, concatenation, and repetition operators. Thus, ERE extend regular expressions with intersection and complementation operators. Formally, for an alphabet Σ, an ERE over Σ is one of the following.

- $\emptyset$, ϵ, or σ, for $\sigma \in \Sigma$.
- $r_1 \vee r_2$, $r_1 \wedge r_2$, $\neg r_1$, $r_1 \cdot r_2$, or r_1^*, for ERE r_1 and r_2.

We use $\mathcal{L}(r)$ to denote the language that r defines. For the base cases, we have $\mathcal{L}(\emptyset) = \emptyset$, $\mathcal{L}(\epsilon) = \{\epsilon\}$, and $\mathcal{L}(\sigma) = \{\sigma\}$. The operators $\vee$, $\wedge$, $\neg$, $\cdot$, and $*$ stand for union, intersection, complementation, concatenation, and repetition (also referred to as Kleene star), respectively. Formally,

- $\mathcal{L}(r_1 \vee r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$.
- $\mathcal{L}(r_1 \wedge r_2) = \mathcal{L}(r_1) \cap \mathcal{L}(r_2)$.
- $\mathcal{L}(\neg r_1) = \Sigma^* \setminus \mathcal{L}(r_1)$.
- $\mathcal{L}(r_1 \cdot r_2) = \{w_1 \cdot w_2 : w_1 \in \mathcal{L}(r_1) \text{ and } w_2 \in \mathcal{L}(r_2)\}$.
- Let $r_1^0 = \{\epsilon\}$ and let $r_1^i = r_1^{i-1} \cdot r_1$, for $i \geq 1$. Thus, $\mathcal{L}(r_1^i)$ contains words that are the concatenation of i words in $\mathcal{L}(r_1)$. Then, $\mathcal{L}(r_1^*) = \bigcup_{i \geq 0} r_1^i$.

2.2 Alternating Automata with Synchronized Universality

An *alternating automaton with synchronized universality* (*ASU*, in short) is a seven-tuple $\mathcal{A} = \langle \Sigma, Q, \mu, q_0, \delta, \psi, F \rangle$, where,

- Σ is a finite input alphabet.
- Q is a finite set of states.
- $\mu : Q \to \{\vee, \wedge\}$ maps each state to a branching mode. The function μ induces a partition of Q to the sets $Q_e = \mu^{-1}(\vee)$ and $Q_u = \mu^{-1}(\wedge)$ of *existential* and *universal* states, respectively.
- $q_0 \in Q$ is an initial state.
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is the transition function.

– $\psi : Q_u \to Q$ is a *synchronization function*.
– $F \subseteq Q_e$ is a set of final states.

ASU run on finite words over Σ. Consider a word $w = \sigma_1, \ldots, \sigma_n$. For technical convenience, we also refer to $\sigma_{n+1} = \epsilon$. During the run of $\mathcal{A}$ on w, it splits into several *copies*. A *position* of a copy of $\mathcal{A}$ is a pair $\langle q, i\rangle \in Q \times \{0, \ldots, n\}$, indicating that the copy is in state q, reading the letter σ_{i+1}. If $q \in Q_e$, we say that $\langle q, i\rangle$ is an *existential position*. Otherwise, it is a *universal position*. The run of $\mathcal{A}$ starts with a single copy in the *initial position* $\langle q_0, 0\rangle$. For $0 \leq i \leq n$, a position $\langle q', i'\rangle$ is a σ_{i+1}-*successor* of a position $\langle q, i\rangle$ if $q' \in \delta(q, \sigma_{i+1})$ and $i' = i + 1$. A position $\langle q', i'\rangle$ is an ϵ-*successor* of $\langle q, i\rangle$ if $q' \in \delta(q, \epsilon)$ and $i' = i$. Finally, $\langle q', i'\rangle$ is a *successor* of $\langle q, i\rangle$ if $\langle q', i'\rangle$ is a σ_{i+1}-successor or an ϵ-successor of $\langle q, i\rangle$. Note that nondeterministic automata with ϵ-moves are a special case of ASU where all states are existential, in which case the function ψ is empty.

Consider a copy of $\mathcal{A}$ in position $\langle q, i\rangle$. If q is an existential state, the copy can move to one of the states in $\delta(q, \sigma_{i+1}) \cup \delta(q, \epsilon)$, thus the new position of the copy is some σ_{i+1} successor or ϵ-successor of $\langle q, i\rangle$. If q is a universal state, the copy should move to all the states in $\delta(q, \sigma_{i+1}) \cup \delta(q, \epsilon)$. Thus, the copy splits into copies that together cover all σ_{i+1}-successors and ϵ-successors of $\langle q, i\rangle$. The computation graph of $\mathcal{A}$ on w embodies all the possible runs of $\mathcal{A}$ on w and is defined as follows.

Definition 1. *The computation graph of $\mathcal{A}$ on an input word $w = \sigma_1 \ldots \sigma_n$ is the directed graph $G = \langle V, E\rangle$, where $V = Q \times \{0, \ldots, n\}$, and $E(\langle q, i\rangle, \langle q', i'\rangle)$ iff $\langle q', i'\rangle$ is a successor of $\langle q, i\rangle$.*

Note that $|V| = m \cdot (n + 1)$, for $m = |Q|$. A *leaf* of G is a position $\langle q, i\rangle \in V$ such that for no $\langle q', i'\rangle \in V$ we have $E(\langle q, i\rangle, \langle q', i'\rangle)$. A *path* from $\langle q, i\rangle$ to $\langle q', i'\rangle$ in G is a sequence of positions $p_1, p_2, \ldots, p_k$, such that $p_1 = \langle q, i\rangle$, $p_k = \langle q', i'\rangle$ and $E(p_i, p_{i+1})$ for $1 \leq i < k$. A *run* of $\mathcal{A}$ on w is obtained from the computation graph by resolving the nondeterministic choices in existential positions. Thus, a run is obtained from G by removing all but one of the edges that leave each existential position. Formally, we have the following.

Definition 2. *Let $G = \langle V, E\rangle$ be the computation graph of $\mathcal{A}$ on w. A run of $\mathcal{A}$ on w is a graph $G_r = \langle V_r, E_r\rangle$ such that $V_r \subseteq V$, $E_r \subseteq E$ and the following hold.*

– $\langle q_0, 0\rangle \in V_r$.
– *Let $\langle q, i\rangle \in V_r$ be a universal position. Then for every position $\langle q', i'\rangle \in V$ such that $E(\langle q, i\rangle, \langle q', i'\rangle)$, we have $\langle q', i'\rangle \in V_r$ and $E_r(\langle q, i\rangle, \langle q', i'\rangle)$.*
– *Let $\langle q, i\rangle \in V_r$ be an existential position. Then either $i = n$ or there is a single position $\langle q', i'\rangle \in V_r$ such that $E_r(\langle q, i\rangle, \langle q', i'\rangle)$.*

Note that when a copy of $\mathcal{A}$ is in an existential position $\langle q, n\rangle$ and $\delta(q, \epsilon) \neq \emptyset$, the copy can choose between moving to an ϵ-successor of q or having q as its final state. In contrast, if $\langle q, n\rangle$ is universal, the copy must continue to all ϵ-successors of q. Note

also that the computation graph G and a run G_r may have cycles. However, these cycles may contain only ϵ-transitions.

Recall that the synchronization function ψ maps each universal state q to a state in Q. Intuitively, whenever a copy of $\mathcal{A}$ in state q is split into several copies, the synchronization function forces all these copies to visit the state $\psi(q)$ and to do so simultaneously. We now formalize this intuition. Let G_r be a run of an ASU $\mathcal{A}$ on an input word w. Let $q \in Q$ be a universal state and $s = \psi(q)$. Consider a position $\langle q, i\rangle$. We say that a position $\langle s, j\rangle$, for $j \geq i$, *covers* $\langle q, i\rangle$, if there is a path from $\langle q, i\rangle$ to $\langle s, j\rangle$ and for every position $\langle s', j'\rangle$ on this path, we have $s' \neq s$. In other words, $\langle s, j\rangle$ is the first instance of s on a path leaving $\langle q, i\rangle$. We say that $\langle q, i\rangle$ is *good* in G_r if there is exactly one position $\langle s, j\rangle$ that covers $\langle q, i\rangle$ and all the paths in G_r that leave $\langle q, i\rangle$ eventually reach $\langle s, j\rangle$.

A run $G_r = \langle V, E_r\rangle$ of an ASU $\mathcal{A}$ on an input word $w = \sigma_1 \dots \sigma_n$ is *accepting* if for all leaves $\langle q, i\rangle \in V$, we have $q \in F$ and $i = n$, and all the universal positions are good in G_r. A word w is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on w. The *language* of $\mathcal{A}$, denoted $L(\mathcal{A})$, is the set of all words accepted by the ASU $\mathcal{A}$.

Example 1. Let $\Sigma = \{0, 1\}$. Consider the language $L \subseteq \Sigma^*$, where a word $w \in \Sigma^*$ is in L iff the $(i+1)$-th letter in w is 0, for some $i = 7 \pmod{12}$. Thus, if we take $\tau = 0 + 1$, then $L = ((\tau^{12})^* \cdot \tau^7) \cdot 0 \cdot \tau^*$. A nondeterministic automaton that recognizes L has at least 13 states. For an integer n, we have $n = 7 \pmod{12}$ if $n = 1 \pmod 3$ and $n = 3 \pmod 4$. Thus, L can be expressed by the ERE $r = (((\tau\tau\tau)^* \cdot \tau) \wedge ((\tau\tau\tau\tau)^*\tau\tau\tau)) \cdot 0 \cdot \tau^*$. For example, the word $w_1 = 1^7 01$ is in L while the word $w_2 = 10101111$ is not. Note that w_2 satisfies both conjuncts of r, but not in the same prefix. We show now an ASU $\mathcal{A}$ with 10 states that recognizes L. $\mathcal{A} = \langle \{0, 1\}, \{q_0, \dots, q_9\}, \mu, q_0, \delta, \psi, \{q_9\}\rangle$, where q_0 is the only universal state, with $\psi(q_0) = q_8$, and the function δ is described in Figure 1.

A run of $\mathcal{A}$ on a word w splits in q_0 into two copies. One copy makes a nondeterministic move to q_8 after reading a prefix of length $1 \pmod 3$ and the other copy does the same after reading a prefix of length $3 \pmod 4$. If the two copies reach q_8 at the same place in the input word, then $\langle q_0, 0\rangle$ is good and if the next letter in the input is 0, the automaton moves to an accepting sink. If the two copies reach q_8 eventually but not at the same time, it means that both conjuncts were satisfied but not necessarily in the same prefix, and the run is not accepting.

2.3 Alternating Automata with Synchronized Universality and Negation

Alternating automata with synchronized universality and negation (*ASUN*, in short) extend ASU by having, in addition to existential and universal states, also *negation states*. It is easy to understand the task of negation states and how an ASUN runs on an input word by taking the *game-theoretic approach* to alternating automata. Let us first explain this approach for ASU. Consider an ASU $\mathcal{A} = \langle \Sigma, Q, \mu, q_0, \delta, \psi, F\rangle$ and an input word $w = \sigma_1 \cdots \sigma_n$. We can view the behavior of $\mathcal{A}$ on w as a game between two players: player 1, who wants to prove that $\mathcal{A}$ accepts w, and player 2, who wants

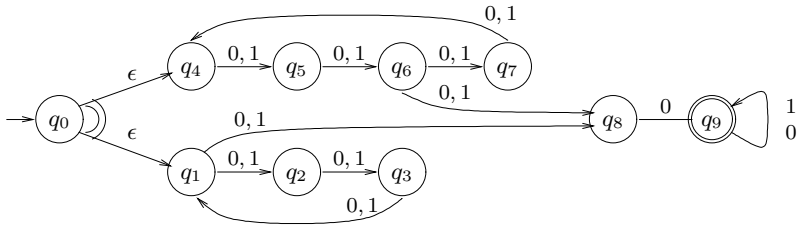

Fig. 1. An ASU for $(((0+1)^{12})^* \cdot (0+1)^7) \cdot 0 \cdot (0+1)^*$

to prove that $\mathcal{A}$ rejects w. For two positions $\langle q, i\rangle$ and $\langle s, j\rangle$, with $j \geq i$, we say that player 1 *wins the game from* $\langle q, i\rangle$ *to* $\langle s, j\rangle$, denoted $\langle q, i\rangle \rightarrow \langle s, j\rangle$, if $\mathcal{A}$ with initial state q and final state s accepts the word $w[i+1, j]$. Otherwise, player 1 *loses* the game from $\langle q, i\rangle$ to $\langle s, j\rangle$, denoted $\langle q, i\rangle \not\rightarrow \langle s, j\rangle$. We also use $\langle q, i\rangle \mapsto \langle s, j\rangle$ to indicate that $\mathcal{A}$ with initial state q and final state s accepts the word $w[i+1, j]$ with only a single visit to s. Note that $\mathcal{A}$ accepts w if there is $s \in F$ such that $\langle q_0, 0\rangle \rightarrow \langle s, n\rangle$.

The relation $\rightarrow$ can be defined inductively as follows.

- For $q \in Q_e$, we have that $\langle q, i\rangle \rightarrow \langle s, j\rangle$ iff $j = i$ and $s = q$, or $j = i$ and $s \in \delta(q, \epsilon)$, or $j = i+1$ and $s \in \delta(q, \sigma_{i+1})$, or there is $\langle q', i'\rangle$ such that $\langle q, i\rangle \rightarrow \langle q', i'\rangle$ and $\langle q', i'\rangle \rightarrow \langle s, j\rangle$.
- For $q \in Q_u$, we have that $\langle q, i\rangle \rightarrow \langle s, j\rangle$ iff there is $i \leq j' \leq j$ such that $\langle \psi(q), j'\rangle \rightarrow \langle s, j\rangle$ and for all q' and i', if $i' = i$ and $q' \in \delta(q, \epsilon)$, or $i' = i+1$ and $q' \in \delta(q, \sigma_{i+1})$, then $\langle q', i'\rangle \mapsto \langle \psi(q), j'\rangle$.

Intuitively, when $q \in Q_e$, we only have to find a successor of $\langle q, i\rangle$ from which an accepting run continues. On the other hand, when $q \in Q_u$, we have to find a position $\langle \psi(q), j'\rangle$ that covers $\langle q, i\rangle$, witnesses that $\langle q, i\rangle$ is good, and from which an accepting run continues.

The negation states of an ASUN dualize the winner in the game between the two players. Like universal states, negation states are mapped into synchronization states. Here, the task of a synchronization state is to mark the end of the scope of the negation, which may be before the end of the input word[1]. Formally, an ASUN is $\mathcal{A} = \langle \Sigma, Q, \mu, q_0, \delta, \psi, F\rangle$, where Σ, Q, q_0, and F are as in ASU, and

- $\mu : Q \rightarrow \{\vee, \wedge, \neg\}$ may now map states to $\neg$, and we use $Q_n = \mu^{-1}(\neg)$ to denote the set of *negation* states.

[1] Readers familiar with alternating automata know that it is easy to complement an alternating automaton by dualizing the function μ and the acceptance condition. A similar complementation can be defined for ASU, circumventing the need for negation states. We found it simpler to add negation states with synchronization, as they enable us to keep the structure of the ASUN we are going to associate with ERE very restricted (e.g., a single accepting state), which leads to a simple membership algorithm.

– The transition function $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to 2^Q$ is such that for all $q \in Q_n$, we have $\delta(q, \epsilon) = \{q'\}$, for some $q' \neq q$, and $\delta(q, \sigma) = \emptyset$ for all $\sigma \in \Sigma$. Thus, each negation state has a single ϵ-successor, and no other successors. For states in $Q_e \cup Q_u$, the transition function is as in ASU.
– The synchronization function $\psi : Q_u \cup Q_n \to Q$ now applies to both universal and negation states.

The computation graph G of an ASUN $\mathcal{A}$ on an input word w is defined exactly as the computation graph for ASU. We define acceptance by an ASUN in terms of the game between players 1 and 2. The relation $\to$ is defined as above for states in Q_e and Q_u. In addition, for every $q \in Q_n$ with $\delta(q, \epsilon) = \{q'\}$, we have that $\langle q, i\rangle \to \langle s, j\rangle$ iff there is $i \leq j' \leq j$ such that $\langle q', i\rangle \not\to \langle \psi(q), j'\rangle$ and $\langle \psi(q), j'\rangle \to \langle s, j\rangle$. Thus, $\mathcal{A}$ with initial state q and final state s accepts $w[i+1, j]$ if there is $i \leq j' \leq j$ such that $\mathcal{A}$ with initial state q' and final state $\psi(q)$ rejects $w[i+1, j']$ and $\mathcal{A}$ with initial state $\psi(q)$ and final state s accepts $w[j'+1, j]$. We then say that the position $\langle \psi(q), j'\rangle$ *covers* $\langle q, i\rangle$.

3 ASUN for ERE

Let Σ be a finite alphabet. Given an ERE r of length m over Σ, we build an ASUN $\mathcal{A}_r$ over Σ with $O(m)$ states such that $L(r) = L(\mathcal{A}_r)$. The construction is similar to the one used for translating regular expression into nondeterministic automata with ϵ-transitions [HMU00]. The treatment of conjunctions is similar to the one described by Yamamoto in [Yam00a], adjusted to our simpler type of automata. The treatment of negations is by negation states.

Theorem 1. *Given an ERE r of length m, we can construct an ASUN $\mathcal{A}_r$ of size $O(m)$ such that $L(r) = L(\mathcal{A}_r)$. Furthermore, $\mathcal{A}_r$ has the following properties:*

1. *$\mathcal{A}_r$ has exactly one accepting state, and there are no transitions out of the accepting state.*
2. *There are no transitions into the initial state.*
3. *The function ψ is one-to-one.*
4. *Every universal state has exactly two ϵ-successors, and no other successors.*
5. *For every existential state q, exactly one of the following holds: q has no successors, q has one or two ϵ-successors, or q has one σ-successor for exactly one $\sigma \in \Sigma$.*
6. *For every state q, exactly one of the following holds: q is the initial state, q is a σ-successor of a single other state, or q is an ϵ-successor of one or two other states.*

Proof: The construction of $\mathcal{A}_r$ is inductive, and we describe it in Figure 2. For the basic three cases of $r = \emptyset, \epsilon$, or σ, the only states of $\mathcal{A}_r$ are q_{in} and q_{fin}. For the cases $r = r_1 \vee r_2, r_1 \wedge r_2, r_1 \cdot r_2, r_1^*$, or $\neg r_1$, we refer to the ASUN $\mathcal{A}_1$ and $\mathcal{A}_2$ of the SERE r_1 and r_2. In particular, the state space of $\mathcal{A}_1$ is Q_1, it has an initial state q_1^{in} and final state q_1^{fin}, and similarly for $\mathcal{A}_2$.

The initial state of the ASUN associated with $r_1 \wedge r_2$ is universal, and the two copies has to synchronize in its final state; thus $\psi(q_{in}) = q_{fin}$. This guarantees that the two

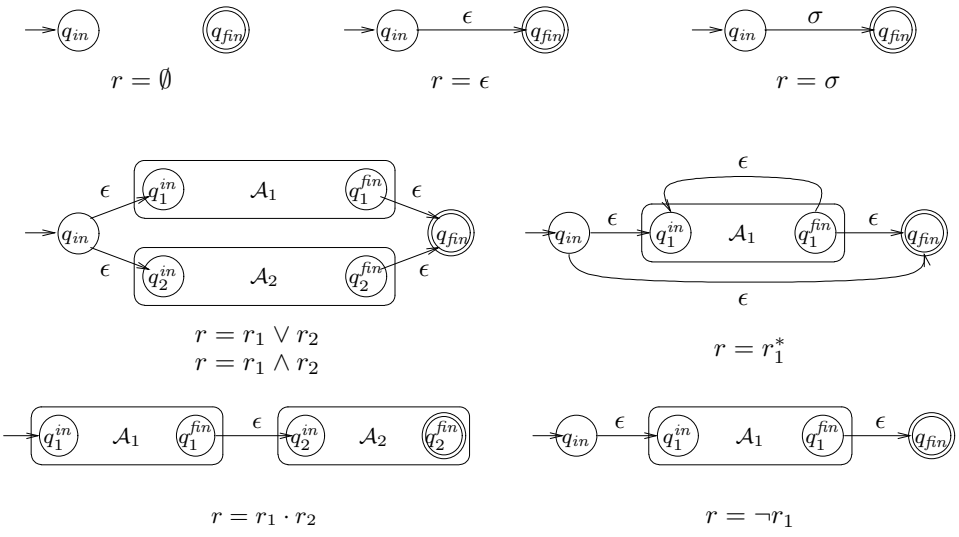

Fig. 2. The ASU $\mathcal{A}_r$ for the ERE r.

copies proceed on words of the same length. Similarly, the initial state of the ASUN associated with $\neg r_1$ is a negation state, and the scope of the negation is bounded to the ASUN $\mathcal{A}_1$; thus $\psi(q_{in}) = q_{fin}$. All the other states are existential. □

Each state of $\mathcal{A}_r$ is associated with a subexpression of r. In particular, the universal and negation states of $\mathcal{A}_r$ are associated with conjunctions and negations, respectively. We refer to $\wedge$ and $\neg$ as *special operators*, refer to states $q \in Q_u \cup Q_n$ as *special states*, and refer to positions $\langle q, i\rangle \in (Q_u \cup Q_n) \times \{0, \ldots, n\}$ as *special positions*. In order to analyze the structure of $\mathcal{A}_r$, we introduce the function $\varphi : Q \to Q_u \cup Q_n \cup \{\bot\}$. Intuitively, $\varphi(q)$, for a state $q \in Q$, is the special state associated with innermost special operator in r in which the subexpression associated with q is strictly nested. If no such special operator exists, then $\varphi(q) = \bot$. Note that we talk about strict nesting, thus $\varphi(q) \neq q$. The formal definition of φ is inductive, and we use the notations in Figure 2. Thus, if $r = \emptyset, \epsilon$, or σ, we refer to q_{in} and q_{fin}, and if r that has r_1 or r_2 as immediate subexpressions, we also refer to Q_1 and Q_2 and the functions φ_1 and φ_2 defined for the ASUN $\mathcal{A}_1$ and $\mathcal{A}_2$. Now, for $r = \epsilon, \emptyset, \sigma$, $r_1 \vee r_2$, $r_1 \cdot r_2$, or r_1^*, and a state $q \in \mathcal{A}_r$, we have

$$\varphi(q) = \begin{cases} \bot & \text{If } q = q_{in} \text{ or } q = q_{fin}. \\ \varphi_i(q) & \text{If } q \in Q_i, \text{ for } i \in \{1, 2\}. \end{cases}$$

Then, for $r = r_1 \wedge r_2$ or $r = \neg r_1$, and a state $q \in \mathcal{A}_r$, we have

$$\varphi(q) = \begin{cases} \bot & \text{If } q = q_{in} \text{ or } q = q_{fin}. \\ \varphi_i(q) & \text{If } q \in Q_i, \text{ for } i \in \{1, 2\}, \text{ and } \varphi_i(q) \neq \bot. \\ q_{in} & \text{If } q \in Q_i, \text{ for } i \in \{1, 2\}, \text{ and } \varphi_i(q) = \bot. \end{cases}$$

Let $\varphi^1(q) = \varphi(q)$, and let $\varphi^{i+1}(q) = \varphi(\varphi^i(q))$, for $i \geq 1$. Then, $\varphi^*(q) = \{\varphi^i(q) : i \geq 1\}$.

Lemma 1. *Consider the computation graph G of an ASUN $\mathcal{A}_r$ constructed in Theorem 1 on an input word w. Let q be a special state and let $\langle q, i\rangle$ be a position in G. The following hold.*

1. *If there is a path from $\langle q, i\rangle$ to $\langle q_{fin}, j\rangle$ in G then there is a position $\langle s, k\rangle$ along this path that covers $\langle q, i\rangle$.*
2. *If $\langle s, k\rangle$ covers $\langle q, i\rangle$ in G, then if there is another special position $\langle q', i'\rangle$ along the path between $\langle q, i\rangle$ and $\langle s, k\rangle$ then there is also a position $\langle s', k'\rangle$ along the path such that $\langle s', k'\rangle$ covers $\langle q', i'\rangle$.*
3. *Let p be a path in G. A position $\langle s, k\rangle$ on p may cover at most one special position on p.*
4. *If $\langle s, k\rangle$ covers $\langle q, i\rangle$ in G, then for every position $\langle q', i'\rangle$ along the path between $\langle q, i\rangle$ and $\langle s, k\rangle$, we have $q \in \varphi^*(q')$.*

4 The Membership Problem

Consider an ERE r. The membership problem is to decide, given $w \in \Sigma^*$, whether $w \in \mathcal{L}(r)$.

Theorem 2. *The membership problem for an ERE r of size m and a word w of length n is decidable in time $O(m \cdot n^2)$ and space $O(m \cdot n + k \cdot n^2)$, where k is the number of special operators in r.*

Proof: We describe an algorithm that runs in time $O(m{\cdot}n^2)$ and space $O(m{\cdot}n{+}k{\cdot}n^2)$, and determines the membership of w in $\mathcal{L}(r)$. A pseudo-code for the algorithm appears in Figure 3. Given an ERE r, we build the ASUN $\mathcal{A} = \langle \Sigma, Q, \mu, q_{in}, \delta, \psi, \{q_{fin}\}\rangle$ according to Theorem 1. The ASUN $\mathcal{A}$ has $O(m)$ states. Next, we build the computation graph G of $\mathcal{A}$ on w. As mentioned before, G has $O(m \cdot n)$ nodes. According to Theorem 1, every state $q \in Q$ has at most two ϵ-successors or exactly one σ-successor, for exactly one $\sigma \in \Sigma$. Therefore, G has $O(m \cdot n)$ edges.

The algorithm operates on G. It begins by determining a full order $\leq$ on the special positions of G. The full order is an extension of the partial order $\leq'$, where $\langle q', i'\rangle \leq' \langle q, i\rangle$ iff $\varphi(q') = q$. Note that according to Lemma 1(4), in every path in G from a special position $\langle q, i\rangle$ to a covering position $\langle s, j\rangle$ of it, if there is another special position $\langle q', i'\rangle$ on this path, then $\langle q', i'\rangle \leq \langle q, i\rangle$. In addition, according to Lemma 1(2), the path also includes some position $\langle s', j'\rangle$ that covers $\langle q', i'\rangle$.

The algorithm proceeds by calling the function *Find_Sync* for each of the special positions, by the order of $\leq$ (starting with the least element according to $\leq$; that is, innermost states are processed first). The function *Find_Sync*($\langle q, i\rangle$) constructs the set $S_{\langle q,i\rangle}$, which consists of indices of covering positions of $\langle q, i\rangle$. Recall that universal positions have two ϵ-successors, denoted *left-child* and *right-child*, while negation positions have exactly one ϵ-successor, denoted *left-child*. The construction of $S_{\langle q,i\rangle}$ begins

function *Membership_Check*(G)
  let $\langle q_1, i_1\rangle \leq \langle q_2, i_2\rangle \leq \cdots \leq \langle q_l, i_l\rangle$ be a full order on the special positions of G,
    such that $\leq$ extends the partial order $\leq'$ given by $\langle q, i\rangle \leq' \langle q', i'\rangle$ iff $\varphi(q) = q'$.
  for all $q \in Q$, $0 \leq i \leq n$ **do**
    $\langle q, i\rangle.index = -1$;
    $\langle q, i\rangle.visited = false$;
  for $j = 1$ to l **do** $S_{\langle q_j, i_j\rangle} =$ *Find_Sync*$(\langle q_j, i_j\rangle)$;
  return$(Find\_Path(\langle q_{in}, 0\rangle))$;

function *Find_Sync*$(\langle q_j, i_j\rangle)$
  if q_j is a universal state **then**
    $S_{\langle q_j, i_j, left\rangle} := \emptyset$; $S_{\langle q_j, i_j, right\rangle} := \emptyset$;
    Update(*left_child*$(\langle q_j, i_j\rangle), \langle q_j, i_j\rangle,$ *left*);
    Update(*right_child*$(\langle q_j, i_j\rangle), \langle q_j, i_j\rangle,$ *right*);
    return$(S_{\langle q_j, i_j, left\rangle} \cap S_{\langle q_j, i_j, right\rangle})$;
  else // q_j is a negation state
    $S_{\langle q_j, i_j, left\rangle} := \emptyset$; *Update*(*child*$(\langle q_j, i_j\rangle), \langle q_j, i_j\rangle,$ *left*);
    return$(\{i_j, \ldots, n\} \setminus S_{\langle q_j, i_j, left\rangle})$;

procedure *Update*$(\langle s, j\rangle, \langle q, i\rangle, d)$
  if $\langle s, j\rangle.index = i$ or $\langle s, j\rangle$ is a leaf **then return**;
  $\langle s, j\rangle.index := i$;
  if $\langle s, j\rangle$ is existential **then**
    if $|\delta(s, \epsilon)| = 2$ **then**
      Update$(left\_child(\langle s, j\rangle), \langle q, i\rangle, d)$;
      Update$(right\_child(\langle s, j\rangle), \langle q, i\rangle, d)$;
    else if $child(\langle s, j\rangle) = \langle \psi(q), j\rangle$ **then** $S_{\langle q,i,d\rangle} := S_{\langle q,i,d\rangle} \cup \{j\}$
      else *Update*$(left\_child(\langle s, j\rangle), \langle q, i\rangle, d)$;
  else // $\langle s, j\rangle$ is special
    for every j' in $S_{\langle s,j\rangle}$ **do** *Update*$(\langle \psi(q), j'\rangle, \langle q, i\rangle, d)$;
  return;

function *Find_Path*$(\langle q, i\rangle)$
  if $\langle q, i\rangle$ is accepting **then return**$(true)$;
  if $\langle q, i\rangle.visited$ or $\langle q, i\rangle$ is a leaf **then return**$(false)$;
  $\langle q, i\rangle.visited := true$;
  if $\langle q, i\rangle$ is existential **then**
    if $|\delta(q, \epsilon)| = 2$ **then**
      return$(Find\_Path(right\_child(\langle q, i\rangle))$ **or** $Find\_Path(left\_child(\langle q, i\rangle)))$;
    return$(Find\_Path(child(\langle q, i\rangle)))$;
  else: // $\langle q, i\rangle$ is special
    for every j in $S_{\langle q,i\rangle}$ **do**
      if $Find\_Path(\langle \psi(q), j\rangle)$ **then return**$(true)$;
    return$(false)$;

Fig. 3. The membership-checking algorithm.

by initializing two empty sets, $S_{\langle q,i,left\rangle}$ and $S_{\langle q,i,right\rangle}$ (in the case where q is a negation state, the second set is redundant). Next, the procedure *Update*$(\langle s,j\rangle, \langle q,i\rangle, d)$ is called, with d being either *left* or *right*. This procedure searches G in DFS manner for covering positions of $\langle q,i\rangle$ starting from the position $\langle s,j\rangle$. The indices of the covering positions are accumulated in the set $S_{\langle q,i,d\rangle}$. Thus, if q is a universal state, *Update* is called for the left and right children of $\langle q,i\rangle$, and the results of the two calls are intersected, forming the set $S_{\langle q,i\rangle}$, which consists of covering positions reachable on both sides. If q is a negation state, then only one call for *Update* is necessary. In this case, the negation is achieved by complementing the set retrieved by *Update* with respect to the set $\{i,\ldots,n\}$ of all potential indices of covering positions for $\langle q,i\rangle$. Note that during the search held by *Update*, if a special position $\langle q',i'\rangle$ is found then we already have $S_{\langle q',i'\rangle}$. Therefore the search may continue from positions in $S_{\langle q',i'\rangle}$, if there are any. This point is crucial for the efficiency of the algorithm.

After the calls for *Find_Sync* for all special positions are completed, the function *Find_Path* is called. This function starts at the initial position $\langle q_{in},0\rangle$ searching the graph for the accepting position $\langle q_{fin},n\rangle$ in a DFS manner. Like the *Update* procedure, whenever a special position $\langle q,i\rangle$ is found, the search continues from positions in $S_{\langle q,i\rangle}$, if there are any. The algorithm returns the result of the function *Find_Path*, which is *true* iff the accepting position was found.

For every position $\langle q,i\rangle$ of G, we keep a boolean flag $\langle q,i\rangle.visited$. *Find_Path* sets the flag for every position it visits, thus avoiding any type of repetitions. In addition, for every position $\langle q,i\rangle$ of G, we keep an integer variable $\langle q,i\rangle.index$. This variable is used in the *Update* procedure and it maintains the index of the special position that we are currently trying to cover. This allows the algorithm to avoid cycling and repetition of bad paths while trying to cover a certain special position. However, multiple checks of a position are allowed when done in different contexts, that is, while trying to cover two different special positions. The reason for allowing this kind of repetition is the possibility of having a position $\langle s,j\rangle$ that might be visited in paths from two special positions $\langle q,i\rangle$ and $\langle q,i'\rangle$. In this case we do not have previous knowledge about covering positions, and we need to go further with the check again. This makes our algorithm quadratic in n rather than linear in n.

The correctness of the algorithm follows from the following claim.

Claim. Let $S_{\langle q,i\rangle}$ be the set constructed for $\langle q,i\rangle$ in *Find_Sync*$(\langle q,i\rangle)$. For every $j \in \{0,\ldots,n\}$, we have that $\langle q,i\rangle \mapsto \langle \psi(q),j\rangle$ iff $j \in S_{\langle q,i\rangle}$.

It is left to show that the algorithm runs in time $O(m\cdot n^2)$ and space $O(m\cdot n+k\cdot n^2)$. For keeping the computation graph, the algorithm requires $O(m\cdot n)$ space. If there are k special operators in r, then there are k special states is $\mathcal{A}$. Since every state in $\mathcal{A}$ corresponds to at most $n+1$ positions in G, we have $O(k\cdot n)$ special positions in G. For each special position we keep at most two sets of at most $n+1$ indices. Therefor, the total space required for storing these sets is $O(k\cdot n^2)$, resulting in overall of $O(m\cdot n+k\cdot n^2)$ space.

Let q be a special state in $\mathcal{A}$. In each call to *Update*$(\langle s,j\rangle, \langle q,i\rangle, d)$, we have $\varphi(s)=q$. Therefore, there are $O(n^2)$ calls for $Update$ involving q and s. Note that the first call

to *Update*$(\langle s, j\rangle, \langle q, i\rangle, d)$ changes $\langle s, j\rangle.index$ to i. Hence, the next calls would return immediately as $\langle s, j\rangle.index = i$. We have $O(m)$ states s, thus the $Update$ procedure is called $O(m \cdot n^2)$ times. The function $Find\_Path$ is called at most twice for each position, as the second call returns immediately. Therefore, there are only $O(m \cdot n)$ calls to $Find\_Path$. As mentioned before, the construction of $\mathcal{A}$ and G can be done in time $O(m \cdot n)$. Hence, the overall running time of the algorithm is $O(m \cdot n^2)$. □

References

[Aho90] A.V. Aho. Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science*, pages 255–300, 1990.

[DHK+89] J. Dassow, J. Hromkovic, J. Karhumaki, B. Rovan, and A. Slobodova. On the power of synchronization in parallel computing. In *Proc. 14th International Symp. on Mathematical Foundations of Computer Science*, volume 379 of *Lecture Notes in Computer Science*, pages 196–206. Springer-Verlag, 1989.

[Hir89] S. Hirst. A new algorithm solving membership of extended regular expressions. Technical report, Basser Department of Computer Science, The University of Sydney, 1989.

[HMU00] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. Addison-Wesley, 2000.

[Hro86] J. Hrokovic. Tradeoffs for language recognition on parallel computing models. In *Proc. 13th Colloq. on Automata, Programming, and Languages*, volume 226 of *Lecture Notes in Computer Science*, pages 156–166. Springer-Verlag, 1986.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[JR91] T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40:25–31, 1991.

[KM95] James R. Knight and Eugene W. Myers. Super-pattern matching. *Algorithmica*, 13(1/2):211–243, 1995.

[KTV01] O. Kupferman, A. Ta-Shma, and M.Y. Vardi. Counting with automata. Submitted, 2001.

[MS73] A.R. Meyer and L.J. Stockmeyer. Word problems requiring exponential time: Preliminary report. In *Proc. 5th ACM Symp. on Theory of Computing*, pages 1–9, 1973.

[Mye92] G. Myers. A four russians algorithm for regular expression pattern matching. *Journal of the Association for Computing Machinery*, 39(4):430–448, 1992.

[Pet02] H. Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *Proc. 18th Symp. on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science. Springer-Verlag, 2002.

[Slo88] A. Slobodova. On the power of communication in alternating machines. In *Proc. 13th International Symp. on Mathematical Foundations of Computer Science*, volume 324 of *Lecture Notes in Computer Science*, pages 518–528, 1988.

[Yam00a] H. Yamamoto. An automata-based recognition algorithm for semi-extended regular expressions. In *Proc. 25th International Symp. on Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 699–708. Springer-Verlag, 2000.

[Yam00b] H. Yamamoto. On the power of input-synchronized alternating finite automata. In *Proc. 6th International Computing and Combinatorics Conference*, volume 1858 of *Lecture Notes in Computer Science*, pages 457–466, 2000.