Succinctness of regular expressions with interleaving, intersection and counting

Wouter Gelade

Hasselt University and transnational University of Limburg, School for Information Technology, Department WNI, Agoralaan Gebouw D, 3590 Diepenbeek, Belgium

ARTICLE INFO

Article history:
Received 1 October 2008
Received in revised form 3 February 2010
Accepted 20 April 2010
Communicated by M. Ito

Keywords:
Regular Expressions
Succinctness

ABSTRACT

In this paper, we study the succinctness of regular expressions (REs) extended with interleaving, intersection and counting operators. We show that in a translation from REs with interleaving to standard regular expressions a double exponential size increase cannot be avoided. We also consider the complexity of translations to finite automata. We give a tight exponential lower bound on the translation of REs with intersection to NFAs, and, for each of the three classes of REs, we show that in a translation to a DFA a double exponential size increase cannot be avoided. Together with known results, this gives a complete picture of the complexity of translating REs extended with interleaving, intersection or counting into (standard) regular expressions, NFAs, and DFAs.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Regular expressions are used in many applications such as text processors, programming languages [34], and XML schema languages [5,32]. These applications, however, usually do not restrict themselves to the standard regular expression using disjunction (+), concatenation (·) and star (∗), but also allow the use of additional operators. Although these operators mostly do not increase the expressive power of the regular expressions, they can have a drastic impact on succinctness, thus making them harder to handle. For instance, it is well known that expressions extended with the complement operator can describe certain languages non-elementary more succinct than standard regular expressions or finite automata [33].

In this paper, we study the succinctness of regular expressions extended with counting (RE(#)), intersection (RE(∩)), and interleaving (RE(&)) operators. The counting operator allows for expressions such as $a^{2,5}$, specifying that there must occur at least two and at most five a’s. These RE(#)’s are used in egrep [20] and Perl [34] patterns and in the XML schema language XML Schema [32]. The class RE(∩) is a well studied extension of the regular expressions, and is often referred to as the semi-extended regular expressions. The interleaving operator allows for expressions such as $a & b & c$, specifying that $a$, $b$, and $c$ may occur in any order, and is used, for instance, in the XML schema language Relax NG [5].

A problem we consider, is the translation of extended regular expressions into (standard) regular expressions. For RE(#) and RE(∩) the complexity of this translation has already been settled and is exponential [22] and double exponential [13], respectively. We show that also in constructing an expression for the interleaving of a set of expressions (an hence also for an RE(&)) a double exponential size increase cannot be avoided. This is the main technical result of the paper. Apart from a pure mathematical interest, the latter result has two important consequences. First, it prohibits an efficient translation...
from Relax NG (which allows interleaving) to XML Schema Definitions (which does not). However, as XML Schema is the widespread W3C standard, and Relax NG is a more flexible alternative, such a translation would be more than desirable. A second consequence concerns the automatic discovery of regular expression describing a set of given strings. The latter problem occurs in the learning of XML schema languages [1–3]. At present these algorithms do not take into account the interleaving operator, but for Relax NG this would be wise as this would allow to learn significantly smaller expressions.

It should be noted here that Gruber and Holzer independently obtained a similar result [18]. They show that any regular expression defining the language \((a_1b_1)^* \& (a_2b_2)^* \& \cdots \& (a_nb_n)^*\) must be of size at least double exponential in \(n\). Compared to the result in this paper, this gives a tighter bound \((2^{2^{2n}})\) instead of \(2^{2^{(n^3)}}\), and shows that the double exponential size increase already occurs for very simple expressions. On the other hand, the alphabet of the counterexamples grows linear with \(n\), whereas the alphabet size is constant for the languages in this paper. Later, they also adapted this result to obtain a (tighter) \(2^{(2^{2n}\ln n)}\) bound over a binary alphabet [18].

We also consider the translation of extended regular expressions to NFAs. For the standard regular expressions, it is well known that such a translation can be done efficiently [4]. Therefore, when considering problems such as membership, equivalence, and inclusion testing for regular expressions the first step is almost invariably a translation to a finite automaton. For extended regular expressions, such an approach is less fruitful. We show that an RE(\& \& \#) can be translated in exponential time into an NFA. However, it has already been shown by Kilpeläinen and Tuhkanen [22] and Mayer and Stockmeyer [24] that such an exponential size increase cannot be avoided for RE(\#) and RE(\&), respectively. For the translation from RE(\& \&) to NFAs, a \(2^{2^{2n}}\) lower bound is reported in [29], which we here improve to \(2^{2^{2n}}\).

As the translation of extended regular expressions to NFAs already involves an exponential size increase, it is natural to ask what the size increase for DFAs is. Of course, we can translate any NFA into a DFA in exponential time, thus giving a double exponential translation, but can we do better? For instance, from the results in [13] we can conclude that given a set of regular expressions, constructing an NFA for their intersection cannot avoid an exponential size increase. However, it is not too hard to see that also a DFA of exponential size accepting their intersection can be constructed. In the present paper, we show that this is not possible for the classes RE(\#), RE(\& \&), and RE(\& \& \#). For each class we show that in a translation to a DFA, a double exponential size increase cannot be avoided. An overview of all results is given in Table 1. Note that as we give upper bounds for the translation from RE(\& \& \#) and lower bounds for the smaller classes, most of these results imply matching upper and lower bounds.

**Table 1**

<table>
<thead>
<tr>
<th></th>
<th>NFA</th>
<th>DFA</th>
<th>RE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE(#)</td>
<td>(2^{2^{2n}}) [22]</td>
<td>(2^{2^{2n}}) (Proposition 7)</td>
<td>(2^{2n} ) [22]</td>
</tr>
<tr>
<td>RE(&amp;)</td>
<td>(2^{2^{2n}}) (Proposition 5)</td>
<td>(2^{2^{2n}}) (Theorem 8)</td>
<td>(2^{2^{2n}}) [18]</td>
</tr>
<tr>
<td>RE(&amp; &amp;)</td>
<td>(2^{2^{2n}}) [24]</td>
<td>(2^{2^{3n}}) (Theorem 10)</td>
<td>(2^{2^{2n}}) (Theorem 20)</td>
</tr>
<tr>
<td>RE(&amp; &amp; #)</td>
<td>(2^{2^{2n}}) (Proposition 4)</td>
<td>(2^{2^{2n}}) (Proposition 6)</td>
<td>(2^{2^{2n}}) (Proposition 11)</td>
</tr>
</tbody>
</table>

**Related work.** The different classes of regular expressions considered here have been well studied; in particular, RE(\& \&) and its membership [21, 23, 29] and equivalence and emptiness [10, 28, 30] problems. Also the classes RE(\#) [22, 27] and RE(\& \&) [12, 24] have received interest. Succinctness of regular expressions has been studied by Ehrenfeucht and Zeiger [8] and, more recently, by Ellul et al. [9], Gelade and Neven [13], Gruber and Holzer [15–18], and Gruber and Johannsen [19]. See the Ph.D. theses of Gruber [14] and Gelade [11] for an overview of many of the recently obtained results. Schott and Spehner give lower bounds for the translation of the interleaving of words to DFAs [31]. Also related, but different in nature, are the results on state complexity [36], in which the impact of the application of different operations on finite automata is studied.

**Outline.** In Section 2 we give the necessary definitions and present some basic results. In Sections 3–5 we study the translation of extended regular expressions to NFAs, DFAs, and regular expressions, respectively.

### 2. Definitions and basic results

#### 2.1. Regular expressions

By \(\mathbb{N}\) we denote the natural numbers without zero. For the rest of the paper, \(\Sigma\) always denotes a finite alphabet. A \(\Sigma\)-string (or simply string) is a finite sequence \(w = a_1a_2 \cdots a_n\) of \(\Sigma\)-symbols. We define the length of \(w\), denoted by \(|w|\), to be \(n\). We denote the empty string by \(\varepsilon\). The set of positions of \(w\) is \([1, \ldots, n]\) and the symbol of \(w\) at position \(i\) is \(a_i\).

By \(w_1 \cdot w_2\) we denote the concatenation of two strings \(w_1\) and \(w_2\). As usual, for readability, we denote the concatenation of \(w_1\) and \(w_2\) by \(w_1w_2\). The set of all strings is denoted by \(\Sigma^*\). A string language is a subset of \(\Sigma^*\). For two string languages \(L, L' \subseteq \Sigma^*\), we define their concatenation \(L \cdot L'\) to be the set \(\{ww' \mid w \in L, w' \in L'\}\). We abbreviate \(L \cdot L \cdots L\) (\(i\) times) by
Let $L$ be a regular language. Then any regular expression defining $L$ is of size at least $|\varepsilon|$, its inverse is not well-defined and thus also not deterministic.

The set of regular expressions over $\Sigma$, denoted by $RE$, is defined in the usual way: $\emptyset, \varepsilon$, and every $\Sigma$-symbol is a regular expression; and when $r_1$ and $r_2$ are regular expressions, then $r_1 \cdot r_2, r_1 + r_2,$ and $r_1^*$ are also regular expressions. By $RE(\& \cap \#)$ we denote the class of extended regular expressions, that is, $RE$s extended with interleaving, intersection and counting operators. So, when $r_1$ and $r_2$ are $RE(\& \cap \#)$-expressions then so are $r_1 \& r_2, r_1 \cap r_2,$ and $r_1^{[k,l]}$ for $k, l \in \mathbb{N}$ with $k \leq l$. By $RE(\&), RE(\cap), \text{and } RE(\#)$, we denote $RE$ extended solely with the interleaving, intersection and counting operator, respectively.

The language defined by an extended regular expression $r$, denoted by $L(r)$, is inductively defined as follows: $L(\emptyset) = \emptyset$; $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$; $L(r_1 + r_2) = L(r_1) \cup L(r_2)$; $L(r^*) = \bigcup_{n=0}^{\infty} L(r)^n$; $L(r_1 \& r_2) = L(r_1) \& L(r_2)$; and $L(r_1^{[k,l]}) = \bigcup_{n=k}^{l} L(r)^n$.

For a set $S = \{a_1, a_2, \ldots, a_n\} \subseteq \Sigma$, we abbreviate by $S$ the regular expression $a_1 + \ldots + a_n$. When $r^{[k,l]}$ is used in a standard regular expression, this is an abbreviation for $r^{(r + \varepsilon)^{l-k}}$.

We define the size of an extended regular expression $r$, denoted by $|r|$, as the number of $\Sigma$-symbols and operators occurring in $r$ plus the sizes of the binary representations of the integers. Formally, $|\varepsilon| = |a| = 1$, for $a \in \Sigma$, $|r_1 \cdot r_2| = |r_1| + |r_2|$, $|r_1 + r_2| = |r_1| + |r_2|$, $|r^*| = |r| + 1$, and $|r^{[k,l]}| = |r| + \log k + \log l$. Intuitively, the star height of a regular expression $r$, denoted by $sh(r)$ equals the number of nested stars in $r$. Formally, $sh(\emptyset) = sh(\varepsilon) = sh(a) = 0$, for $a \in \Sigma$, $sh(r_1 \cdot r_2) = sh(r_1 + r_2) = \max\{sh(r_1), sh(r_2)\}$,

and $sh(r^*) = sh(r) + 1$. The star height of a regular language $L$, denoted by $sh(L)$, is the minimal star height among all regular expressions defining $L$.

The latter two concepts are related through the following theorem due to Gruber and Holzer [15], which will allow us to reduce our questions about the size of regular expressions to questions about the star height of regular languages.

**Theorem 1 ([15]).** Let $L$ be a regular language. Then any regular expression defining $L$ is of size at least $2^{\frac{1}{2}(sh(L) - 1)} - 1$.2

### 2.2. Finite automata and graphs

A non-deterministic finite automaton (NFA) $A$ is a 4-tuple $(Q, q_0, \delta, F)$ where $Q$ is the set of states, $q_0$ is the initial state, $F$ is the set of final states and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. As usual, we denote by $\delta^* \subseteq Q \times \Sigma^* \times Q$ the reflexive–transitive closure of $\delta$. Then, $w$ is accepted by $A$ if $(q_0, w, q_f) \in \delta^*$ for some $q_f \in F$. The set of strings accepted by $A$ is denoted by $L(A)$. The size of an NFA is $|\delta|$. An NFA is deterministic (or a DFA) if for all $a \in \Sigma, q \in Q$, $\{((q, a, q') \in \delta \mid q' \in Q\} \leq 1$.

Note that we do not require DFAs to be complete, i.e., $\{(a, a, q') \in \delta \mid q' \in Q\}$ can be empty for some $q \in Q, a \in \Sigma$.

A state $q \in Q$ is useful if there exist strings $w, w' \in \Sigma^*$ such that $(q_0, w, q) \in \delta^*$, and $(q, w', q_f) \in \delta^*$, for some $q_f \in F$. An NFA is trim if it only contains useful states. For $q \in Q$, let $\text{symbols}(q) = \{a \mid \exists p \in Q, (p, a, q) \in \delta\}$. Then, $A$ is state-labeled if any $q \in Q$, $|\text{symbols}(q)| \leq 1$, i.e., all transitions to a single state are labeled with the same symbol. In this case, we also denote this symbol by $\text{symbol}(q)$. Further, $A$ is non-returning if symbols($q_0$) = $\emptyset$, i.e., $q_0$ has no incoming transitions. A language $L$ is bideterministic if there exists a DFA $A$, accepting $L$, such that the inverse of $A$ is again deterministic. Here, the inverse of $A$ is obtained by exchanging the initial and final state and inverting every transition of $A$. If $A$ has more than one final state, its inverse is not well-defined and thus also not deterministic.

A (directed) graph $G$ is a tuple $(V, E)$, where $V$ is the set of vertices and $E \subseteq V \times V$ is the set of edges. A graph $(U, F)$ is a subgraph of $G$ if $U \subseteq V$ and $F \subseteq E$. For a set of vertices $U \subseteq V$, the subgraph of $G$ induced by $U$, denoted by $G[U]$, is the graph $(U, F)$, where $F = \{(u, v) \mid u, v \in U \land (u, v) \in E\}$.

A graph $G = (V, E)$ is strongly connected if for every pair of vertices $u, v \in V$, both $u$ is reachable from $v$, and $v$ is reachable from $u$. A set of edges $V' \subseteq V$ is a strongly connected component (SCC) of $G$ if $G[V']$ is strongly connected and for every set $V''$, with $V' \subseteq V''$, $G[V'']$ is not strongly connected.

We now introduce the cycle rank of a graph $G = (V, E)$, denoted by $cr(G)$, which is a measure for the structural complexity of $G$. It is inductively defined as follows: (1) If $G$ is acyclic or empty, then $cr(G) = 0$, otherwise (2) if $G$ is strongly connected, then $cr(G) = \min_{V'} cr(G[V \setminus \{v\}]) + 1$, and otherwise (3) $cr(G) = \max_{V''} cr(G[V''])$.

We say that a graph $H$ is a minor of a graph $G$ if $H$ can be obtained from $G$ by removing edges, removing vertices, and contracting edges. Here, contracting an edge between two vertices $u$ and $v$, means replacing $u$ and $v$ by a new vertex, which inherits all incoming and outgoing edges of $u$ and $v$. Now, it has been shown by Cohen [6] that removing edges or nodes from a graph does not increase the cycle rank of a graph, while McNaughton [25] essentially showed that the same holds when contracting edges.

**Lemma 2 ([6,25]).** If a graph $H$ is a minor of a graph $G$, then $cr(H) \leq cr(G)$.

Let $A = (Q, q_0, \delta, F)$ be an NFA. The underlying graph $G$ of $A$ is the graph obtained by removing the labels from the transition edges of $A$, or more formally $G = (Q, E)$, with $E = \{(q, a, q') \mid 3a \in \Sigma, (q, a, q') \in \delta\}$. In the following, we often abuse notation and for instance say the cycle rank of $A$, referring to the cycle rank of its underlying graph.

---

2 In fact, in [15] the size of an expression only consists of the alphabet symbols, while we also count the number of operators. Thus, the size of an expression in this paper is always at least as big as in [15], and the lemma carries over to this setting.
There is a strong connection between the star height of a regular language, and the cycle rank of the NFAs accepting it, as witnessed by the following theorem. Theorem 3(1) is known as Eggan’s Theorem [7] and proved in its present form by Cohen [6]. Theorem 3(3) is due to McNaughton [25].

**Theorem 3.** For any regular language \( L \),

1. \( sh(L) = \min \{ cr(A) \mid A \text{ is an NFA accepting } L \} \).
2. \( sh(L) \cdot |\Sigma| \geq \min \{ cr(A) \mid A \text{ is a non-returning state-labeled NFA accepting } L \} \).
3. If \( L \) is bidenterministic, then \( sh(L) = cr(A) \), where \( A \) is the minimal trim DFA accepting \( L \).

**Proof.** We only have to prove (2). Thereto, let \( L \) be a regular language over an alphabet \( \Sigma \). By Eggan’s Theorem (Theorem 3(1)), we know that there exists an NFA \( A \), with \( L(A) = L \) and \( cr(A) = sh(L) \). We show that, given \( A \), we can construct a non-returning state-labeled NFA \( F^A \) equivalent to \( A \) such that \( cr(A) \cdot |\Sigma| \geq cr(F^A) \) from which the theorem follows.

Let \( A = (Q, q_0, \delta, F) \) be an NFA over \( \Sigma \), we construct \( F^A \) in two steps. First, we construct a non-returning NFA \( B = (Q^B, q_0, \delta^B, F^B) \), with \( L(B) = L(A) \), as follows: \( Q^B = Q \cup \{ q_0 \} \), \( \delta^B = \delta \cup \{ (q_0, a, q) \mid q \in Q, a \in \Sigma, (q_0, a, q) \in \delta \} \), and \( F^B = F \cup \{ q_0 \} \). Then, let \( F^B = F \) if \( q_0 \notin F \), otherwise. Intuitively, \( B \) is a \( A \) extended with a new initial state which only inherits the outgoing transitions of the old initial state. It should be clear that \( B \) is non-returning and \( L(B) = L(A) \). Furthermore, \( cr(B) = cr(A) \) because \( q_0 \) has no incoming transitions and it thus forms a separate strongly connected component in \( B \) whose cycle rank is 0. From the definitions it then follows that \( cr(B) = cr(A) \).

From \( B \), we now construct the non-returning state-labeled NFA \( F^A \) such that \( cr(F^A) \leq cr(B) \cdot |\Sigma| = cr(A) \cdot |\Sigma| \). Let \( F^A = (Q^A, q^A_0, \delta^A, F^A) \) be defined as

- \( Q^A = \{ a^q \mid q \in Q^B, a \in \Sigma \} \);
- \( q^A_0 = q_0^A \), for some \( a \in \Sigma \);
- \( \delta^A = \{ (a^q, b, p^b) \mid q, p \in Q^B, a, b \in \Sigma, (q, b, p) \in \delta^B \} \); and
- \( F^A = \{ a^q \mid q \in F^B, a \in \Sigma \} \).

That is, \( B^A \) contains \( |\Sigma| \) copies of every state \( q \) of \( B \), each of which captures all incoming transitions of \( q \) for one alphabet symbol. Obviously, \( B^A \) is a non-returning state-labeled NFA with \( L(B^A) = L(B^A) \).

We conclude by showing that \( cr(B) \cdot |\Sigma| \geq cr(B^A) \). In the following, we abuse notation and, for a set of states \( P \), write \( B[P] \) for the sub-automaton of \( B \) induced by \( P \), defined in the obvious way. Now, for a set of states \( P \) of \( B \), let \( P^B = \{ q^B \mid q \in P, a \in \Sigma \} \), and observe that \( B[Q \setminus P]^B = B^A[Q^B \setminus P^B] \) always holds. We now show \( cr(B) \cdot |\Sigma| \geq cr(B^A) \) by induction on the number of states of \( B \). If \( |Q| = 1 \) then either the single state does not contain a loop, such that \( cr(B) = cr(B^A) = 0 \), or the single state contains a self-loop, in which case \( cr(B) = 1 \), and as \( |Q^B| = |\Sigma| \), \( cr(B^A) \leq |\Sigma| \) holds, and hence \( cr(B) \cdot |\Sigma| \geq cr(B^A) \).

For the induction step, if \( B \) is acyclic, then, again, \( cr(B) = cr(B^A) = 0 \). Otherwise, if \( B \) is strongly connected, then \( cr(B) = cr(B[Q \setminus \{ q \}]) + 1 \), for some \( q \in Q^B \). Then, \( cr(B) \cdot |\Sigma| = cr(B[Q \setminus \{ q \}]) \cdot |\Sigma| + |\Sigma| \), by which the induction hypothesis gives us \( cr(B) \cdot |\Sigma| \geq cr(B^A[Q^B \setminus \{ q^B \}]) + |\Sigma| \). Finally, it is shown in [15] that removing a set of states \( P \) from a graph can never decrease its cycle rank by more than \( |P| \). Therefore, as \( |Q^B| = |\Sigma| \), it follows that \( cr(B^A[Q \setminus \{ q \}]) + |\Sigma| \geq cr(B^A) \), and hence \( cr(B) \cdot |\Sigma| \geq cr(B^A) \).

Otherwise, if \( B \) consists of several strongly connected components \( Q_1, \ldots, Q_k \), with \( k \geq 2 \), then \( cr(B) = \max_{i \leq k} cr(B(Q_i)) \). But now, by construction, every strongly connected component \( V \) of \( B^A \) is contained in \( Q^B \), for some \( i \in [1, k] \). Then, \( B^A[V] \) is a minor of \( B^A[Q^B] \), and hence it follows from Lemma 2 that \( cr(B^A) = \max_{i \leq k} cr(B^A[Q^B]) \leq \max_{i \leq k} cr(B^A[Q^B]) \). Now, by induction, \( cr(B(Q_i)) \cdot |\Sigma| \geq cr(B^A[Q^B]) \) for all \( i \in [1, k] \), from which it follows that \( cr(B) \cdot |\Sigma| \geq cr(B^A) \). \( \square \)

### 3. Succinctness w.r.t. NFAs

In this section, we study the complexity of translating extended regular expressions into NFAs. We show that such a translation can be done in exponential time, by constructing the NFA by induction on the structure of the expression.

**Proposition 4.** Let \( r \) be a RE(\&\&, \cup, #). An NFA \( A \) with at most \( 2^{\Theta(|r|)} \) states, such that \( L(r) = L(A) \), can be constructed in time \( 2^{\Theta(|r|)} \).

**Proof.** We construct \( A \) by induction on the structure of the formula. For the base cases, \( r = s, r = \emptyset \), and \( r = a \), for \( a \in \Sigma \), and the induction cases \( r = r_1 \cup r_2, r = r_1 \cap r_2, \) and \( r^\ast \) can thus be easily done using standard constructions. We give the full construction for the three special operators:

- If \( r = r_1 \cup r_2 \), for \( i \in [1, 2] \), let \( A_i = (Q^i, Q_0^i, \delta^i, F^i) \) accept \( L(r_i) \). Then, \( A = (Q = Q_1 \times Q_2, \delta = \delta_1 \cup \delta_2, F = F_1 \times F_2, F = \delta_1 \cup \delta_2) \) accepts \( r \).

- If \( r = r_1 \cap r_2 \), then \( A \) is defined exactly as for \( r_1 \cup r_2 \), except for \( \delta \) which now equals \( \delta_1 \cup \delta_2 \) except for \( \delta \) which now equals \( \delta_1 \cup \delta_2 \).

- If \( r = r_1^{[\ell]} \), let \( A_1 \) accept \( L(r_1) \). Then, let \( B_1 \) to \( B_{\ell} \) be \( \ell \) identical copies of \( A_i \) with disjoint sets of states. For \( i \in [1, \ell] \), let \( B_i = (Q_1, Q_0^i, \delta^i, F^i) \). Now, define \( A = B_1 \cup B_{\ell} \) accepting \( r_1^{[\ell]} \) as follows: \( Q = \bigcup_{i \leq \ell} Q_i \), \( F = \bigcup_{i \leq \ell} F_i \), and \( \delta = \bigcup_{i \leq \ell} \delta_1 \cup \delta_2 \cup \{ (q, a, q_0^{i+1}) \mid q_i \in Q_i \wedge q_i \in F_i \) such that \( q_i, a, p_i \in \delta_i \).
We note that the construction for the interleaving operator comes from [24], where it was already used for a translation from RE(∩) to NFAs. We argue that $A$ contains at most $2^{|\tau|}$ states. For $r = r_1 \cap r_2$ or $r = r_1 \& r_2$, by induction $A_1$ and $A_2$ contain at most $2^{|r_1|}$ and $2^{|r_2|}$ states and, hence, $A$ contains at most $2^{|\tau|} \cdot 2^{|r_1|} \cdot 2^{|r_2|} = 2^{|\tau|+|r_1|+|r_2|} \leq 2^{|\tau|}$ states. For $r = r_i^{[k,i]}$, similarly, $A$ contains at most $\ell \cdot 2^{|\tau|} = 2^{|\tau|+\log \ell} \leq 2^{|\tau|}$ states. Furthermore, as the intermediate automata never have more than $2^{|\tau|}$ states and we have to do at most $|\tau|$ such constructions, the total construction can be done in time $2^{O(|\tau|)}$. ☐

This exponential size increase cannot be avoided for any of the classes. For RE(∩) this is witnessed by the expression $a^{2^n \cdot 2^n}$ and for RE(∪) by the expression $a_1 & a_2 \& \cdots \& a_n$, as already observed by Kilpeläinen and Tuhkanen [22] and Mayer and Stockmeyer [24], respectively. For RE(∪), a $2^{O(\sqrt{\#})}$ lower bound has already been reported in [29]. The present tighter statement, however, will follow from Theorem 8 and the fact that any NFA with $n$ states can be translated into a DFA with at most $2^n$ states [35].

**Proposition 5.** For any $n \in \mathbb{N}$, there exist an RE(∩) $r^a$, an RE(∪) $r^b$, and an RE(∪) $r^c$, each of size $O(n)$, such that any NFA accepting $r^a$, $r^b$, or $r^c$ contains at least $2^n$ states.

4. Succinctness w.r.t. DFAs

In this section, we study the complexity of translating extended regular expressions into DFAs. First, from Proposition 4 and the fact that any NFA with $n$ states can be translated into a DFA with $2^n$ states in exponential time [35], we can immediately conclude the following.

**Proposition 6.** Let $r$ be a RE(∩, ∪, #). A DFA $A$ with at most $2^{|\tau|}$ states, such that $L(r) = L(A)$, can be constructed in time $2^{O(|\tau|)}$.

We show that, for each of the classes RE(∩), RE(∪), or RE(∪), this double exponential size increase cannot be avoided.

**Proposition 7.** For any $n \in \mathbb{N}$ there exists an RE(∩) $r_\tau$ of size $O(n)$ such that any DFA accepting $L(r_\tau)$ contains at least $2^n$ states.

**Proof.** Let $n \in \mathbb{N}$ and define $r_\tau = (a + b)^n (a + b)^{2^n \cdot 2^n}$. Here, $r_\tau$ is of size $O(n)$ since the integers in the numerical predicate are stored in binary. We show that any DFA $A = (Q, q_0, \delta, F)$ accepting $L(r_\tau)$ has at least $2^{2^n}$ states. Towards a contradiction, suppose that $A$ has less than $2^{2^n}$ states and consider all strings of length $2^n$ containing only $a$'s and $b$'s. As there are exactly $2^{2^n}$ such strings, and $A$ contains less than $2^{2^n}$ states, there must be two different such strings $w, w'$ and a state $q$ of $A$ such that $(q_0, w, q) \in \delta^*$ and $(q_0, w', q) \in \delta^*$. But now, as $w \neq w'$, there exists some $i \in [1, 2^n]$ such that the $i$th position of $w$ contains an $a$ and the $i$th position of $w'$ contains a $b$ (or the other way around, but that is identical). Therefore, $w' \neq w$, and $w' \neq w$ are both accepted or both not accepted by $A$, and hence $L(r_\tau) \neq L(A)$, which gives us the desired contradiction. ☐

We now move to regular expressions extended with the intersection operator. The succinctness of RE(∪) with respect to DFAs can be obtained along the same lines as the simulation of exponential space Turing machines by RE(∪) in [10].

**Theorem 8.** For any $n \in \mathbb{N}$ there exists an RE(∪) $r_\tau^n$ of size $O(n)$ such that any DFA accepting $L(r_\tau^n)$ contains at least $2^n$ states.

**Proof.** Let $n \in \mathbb{N}$. We start by describing the language $g_n$ which will be used to establish the lower bound. This will be a variation of the following language over the alphabet \( \{a, b\} \): \( \{ww' \mid |w| = 2^n\} \). It is well known that this language is hard to describe by a DFA. However, to define it very succinctly by an RE(∩), we need to add some additional information to it.

Thereto, we first define a marked number as a string over the alphabet \( \{0, 1\} \) defined by the regular expression \( (0 + 1)^* \cdot 10^* + 0^* \cdot \). i.e., a binary number in which the rightmost 1 and all following 0’s are marked. Then, for any $i \in [0, 2^n - 1]$ let enc(i) denote the $i$-bit marked number encoding $i$. These marked numbers were introduced in [10], where the following is observed: if $i, j \in [0, 2^n - 1]$ are such that $j = i + 1 \mod 2^n$, then the bits of $i$ and $j$ which are different are exactly the marked bits of $j$. For instance, for $n = 2$, enc(1) = 01 and enc(2) = 10 and they differ in both bits as both bits of enc(2) are marked. Further, let enc(i) denote the reversal of enc(i).

Now, for a string $w = a_0 a_1 \cdots a_{2^n - 1}$ define

\[
\text{enc}(w) = \text{enc}(0)a_0 \text{enc}(0)\text{enc}(1)a_1 \text{enc}(1)\cdots \text{enc}(2^n - 1)a_{2^n - 1} \text{enc}(2^n - 1)
\]

and, finally, define

\[
g_n = \{\#\text{enc}(w)\#\text{enc}(w) \mid w \in L((a + b)^n) \land |w| = 2^n\}.
\]

For instance, for $n = 2$, and $w = abba$, enc(w) = 00000100101010111 and hence #00000100101010111 #00000100101010111 \in g_2. Note that enc(0) equals 00 and not 00.

Now, let’s consider $\overline{g}_n$, the complement of $g_n$. Using a standard argument, similar to the one in the proof of Proposition 7, it is straightforward to show that any DFA accepting $\overline{g}_n$ must contain at least $2^n$ states. We conclude by constructing a regular expression $r_\tau^n$ of size $O(n)$ defining $\overline{g}_n$.

Note that $\Sigma = \{0, 1, n, a, b, S, \#\}$. We define $N = \{0, 1, \bar{1}\}, S = \{a, b\}, D = \{\#\}$ and for any set $U$, and $\sigma \in U$, let $U_\sigma = U \setminus \{\sigma\}$. Now, we construct a set of expressions, each capturing a possible mistake in a string. Then, $r_\tau^n$ simply is the disjunction of these expressions. The expressions are as follows.
All strings which do not start with #:
\[ \varepsilon + \Sigma_n \Sigma^*. \]

All strings in which two symbols at a distance \( n + 1 \) do not match:
\[ \Sigma^*(N \Sigma^n(S + D) + S \Sigma^n(S + N) + D \Sigma^n(D + N)) \Sigma^*. \]

All strings which do not end with \( \text{enc}(2^n - 1) = 1^{n-1} \bar{1} \):
\[ \Sigma^*(\Sigma_1 + \Sigma_1 \Sigma^{[1,n-1]}) \).

All strings in which \# occurs before any other number than \( \text{enc}^R(0) \) or where a $ occurs before \( \text{enc}^R(0) \):
\[ \Sigma^*(\Sigma^0 + \#(\Sigma^{[0,n-1]} \Sigma_0)) \Sigma^*. \]

All strings which contain more or less than 2 #-symbols
\[ \Sigma^*_a(\# + \varepsilon) \Sigma^*_a + \Sigma^*_a \Sigma^a \Sigma^* \Sigma^a. \]

All strings which contain a (non-reversed) binary number which is not correctly marked:
\[ \Sigma^* \Sigma^* \Sigma^*((0 + 1)(\overline{0} + S + \#) + (\overline{0} + \bar{1})N_0) \Sigma^*. \]

All strings in which the binary encodings of two numbers surrounding an \( a \) or \( b \) are not each others reverse. Thereto, we first define expressions \( r_i \), for all \( i \in [0,n] \), such that \( L(r_i) = \{ w \sigma w' \mid \sigma \in S \land w, w' \in \Sigma^* \land |w| = |w'| \land |w| \leq i \} \), inductively as follows: \( r_0 = S \) and for all \( i \in [1,n] \), \( r_i = r_0 + \Sigma r_{i-1} \). Then, the following is the desired expression:
\[ \Sigma^*(r_n \cap \bigcup_{\sigma \in \Sigma} \Sigma^\sigma \Sigma_n) \Sigma^*. \]

All strings in which the binary encodings of two numbers surrounding a $ or # do not differ by exactly one, i.e., there is a substring of the form \( \text{enc}(i) \Sigma \text{enc}(j) \) or \( \text{enc}(i) \# \Sigma \text{enc}(j) \) such that \( j \neq i + 1 \mod 2^n \). Exactly as above, we can inductively define \( r_n \) such that \( L(r_n) = \{ w \sigma w' \mid \sigma \in D \land w, w' \in \Sigma^* \land |w| = |w'| \land |w| \leq n \} \). Then, we obtain:
\[ \Sigma^*(r_n \cap ((0 + \overline{0}) \Sigma^*(0 + 1) + (1 + \bar{1}) \Sigma^*(1 + 0))) \Sigma^*. \]

All strings in which two \( a \) or \( b \) symbols are different but should be equal. We now define expressions \( s_i \), for all \( i \in [0,n] \) such that \( L(s_i) = \{ u w u w^R \mid u, v, w \in \Sigma^* \land |w| = i \} \). By induction, \( s_0 = \Sigma^* \#, S^* \), and for all \( i \in [1,n] \), \( s_i = \Sigma s_{i-1} \Sigma \cap \bigcup_{\sigma \in \Sigma} \Sigma^\sigma \Sigma^* \). Then, the following is the desired expression:
\[ \Sigma^*(s_n a b + b s_n a) \Sigma^*. \]

Now, a string is not in \( g_n \) if it is accepted by at least one of the previous expressions. Hence, \( r_n \), defined as the disjunction of all these expressions, defines exactly \( g_n \). Furthermore, notice that all expressions, including the inductively defined ones, are of size \( O(n) \), and hence \( r_n \) is also of size \( O(n) \). This concludes the proof. \( \Box \)

We can now extend the results for \( \text{RE}(\cap) \) to \( \text{RE}(\&). \) We do this by using a technique of Mayer and Stockmeyer [24] which allows, in some sense, to simulate an \( \text{RE}(\cap) \) by an \( \text{RE}(\&). \) We illustrate their technique in a simple case, for an expression \( r = r_1 \cap r_2 \), where \( r_1 \) and \( r_2 \) are normal regular expressions, not containing intersection operators. Let \( c \) be a symbol not occurring in \( r \), and for \( i = 1, 2 \), define \( r_i^c \) as the expression obtained from \( r_i \) by replacing any symbol \( a \) in \( r_1 \) by \( a c \). Then, it is easily seen that a string \( a_0 a_1 \cdots a_n \in L(r_1) \) iff \( a_0 a_1 c \cdots a_n c \in L(r_2^c) \). Now, consider the expression \( r^c = r_1^c \& r_2^c \). Then, a string \( a_0 a_1 \cdots a_n \in L(r_1) \cap L(r_2) \) iff \( a_0 a_1 c \cdots a_n c \in L(r_1^c) \cap L(r_2^c) \) iff \( a_0 c^2 a_1 c^2 \cdots a_n c^2 \in L(r^c) \). So, not all strings defined by \( r^c \) are of the form \( a_0 c^2 a_1 c^2 \cdots a_n c^2 \), but the set of strings of the form \( a_0 c^2 a_1 c^2 \cdots a_n c^2 \) defined by \( r^c \) corresponds exactly to the set of strings defined by \( r \). In this sense, \( r^c \) simulates \( r \).

The latter technique can be extended to general \( \text{RE}(\cap) \) and \( \text{RE}(\&). \) To formally define this, we need some notation. Let \( w = a_0 a_1 \cdots a_n \) be a string over an alphabet \( \Sigma \), and let \( c \) be a symbol not in \( \Sigma \). Then, for any \( i \in N \), define \( \text{pump}_i(w) = a_0 c^i a_1 c^i \cdots a_n c^i \).

**Lemma 9** ([24]). Let \( r \) be an \( \text{RE}(\cap) \) containing \( k \cap \)-operators. Then, there exists an \( \text{RE}(\& \& \cap \cap) \) of size at most \( |r|^2 \) such that for any \( w \in \Sigma^* \), \( w \in L(r) \) iff \( \text{pump}_k(w) \in L(s) \).

Using this lemma, we can now prove the following theorem.

**Theorem 10.** For any \( n \in N \) there exists an \( \text{RE}(\& \& \cap \cap) \) of size \( O(n^2) \) such that any DFA accepting \( L(r_n^k) \) contains at least \( 2^{2^n} \) states.

**Proof.** Let \( n \in N \) and consider the expression \( r_n^k \) of size \( O(n) \) constructed in Theorem 8 such that any DFA accepting \( L(r_n^k) \) contains at least \( 2^{2^n} \) states. Now, let \( r_n^k \) be the regular expression simulating \( r_n^k \) obtained from Lemma 9, such that for some \( k \in N \) and any \( w \in \Sigma^* \), \( w \in L(r_n^k) \) iff \( \text{pump}_k(w) \in L(r_n^k) \). Then, \( r_n^k \) is of size \( O(n^2) \) and, exactly as before, it is straightforward to show that any DFA accepting \( L(r_n^k) \) contains at least \( 2^{2^n} \) states. \( \Box \)
5. Succinctness w.r.t. regular expressions

In this section, we study the translation of extended regular expressions to (standard) regular expressions. First, for the class RE(#) it has already been shown by Kilpelainen and Tuhkanen [22] that this translation can be done in exponential time, and that an exponential size increase cannot be avoided. Furthermore, from Proposition 4 and the fact that any NFA with n states can be translated into a regular expression in time \(2^{n^2}n\) [9] it immediately follows that:

**Proposition 11.** Let \(r\) be a RE(\&, \(\cap\), #). A regular expression \(s\) equivalent to \(r\) can be constructed in time \(2^{2^{O(n^3)}}\).

Furthermore, from the results in [13] (see also Table 1(b)) it follows that in a translation from RE(\(\cap\)) to standard regular expressions, a double exponential size increase cannot be avoided.

Hence, it only remains to show a double exponential lower bound on the translation from RE(\&\&\&\&\&\&\&\&) to standard regular expressions, which is exactly what we will do in the rest of this section. Theoreo, we proceed in several steps and define several families of languages. First, we introduce the family of languages \((\mathcal{K}_n)_{n \in \mathbb{N}}\), on which all following languages will be based, and establish its star height. The star height of languages will be our tool for proving lower bounds on the size of regular expressions defining these languages. Then, we define the family \((\mathcal{L}_n)_{n \in \mathbb{N}}\) which is a binary encoding of \((\mathcal{K}_n)_{n \in \mathbb{N}}\) and show that these languages can be defined as the intersection of small regular expressions.

Finally, we define the family \((\mathcal{M}_n)_{n \in \mathbb{N}}\) which is obtained by simulating the intersection of the previously obtained regular expressions by the interleaving of related expressions, similar to the simulation of RE(\(\cap\)) by RE(\&) in Section 4. Bringing everything together, this then leads to the desired result: a double exponential lower bound on the translation of RE(\&) to RE.

As an intermediate corollary of this proof, we also obtain a double exponential lower bound on the translation of RE(\(\cap\)) to RE, similar to a result in [13]. We note, however, that the succinctness results for RE(\&) cannot be obtained by using the results in [13], and that, hence, the different lemmas which prove the succinctness of RE(\(\cap\)) are necessary to obtain the subsequent results on RE(\&).

5.1. \(\mathcal{K}_n\): The basic language

We first introduce the family \((\mathcal{K}_n)_{n \in \mathbb{N}}\) defined by Ehrenfeucht and Zeiger over an alphabet whose size grows quadratically with the parameter \(n\) [8]:

**Definition 12.** Let \(n \in \mathbb{N}\) and \(\Sigma_n = \{a_{i,j} \mid 0 \leq i, j \leq n - 1\}\). Then, \(\mathcal{K}_n\) contains exactly all strings of the form \(a_{0,0}, a_{0,i}, \ldots, a_{0,n-1}\) where \(k \in \mathbb{N} \cup \{0\}\).

An alternative definition of \(\mathcal{K}_n\) is through the minimal DFA accepting it. Theoreo, let \(A_n^\mathcal{K}\) be the minimal DFA accepting \(\mathcal{K}_n\) and \(\mathcal{E}\). Then, \(A_n^\mathcal{K}\) is deterministic as transition is labeled with a different symbol. Furthermore, \(A_n^\mathcal{K}\) is the minimal trim DFA accepting \(\mathcal{K}_n\). Hence, by Theorem 3(3), sh(\(\mathcal{K}_n\)) = cr(\(A_n^\mathcal{K}\)). We conclude by showing that, for any \(n \in \mathbb{N}\), cr(\(A_n^\mathcal{K}\)) = \(n\).

Thereto, first observe that the graph underlying \(A_n^\mathcal{K}\) is the complete graph (including self-loops) on \(n\) nodes, which we denote by \(K_n\). We proceed by induction on \(n\). For \(n = 1\), the graph is a single node with a self-loop and hence by definition cr(\(K_1\)) = 1. For the inductive step, suppose that cr(\(K_n\)) = \(n\) and consider \(K_{n+1}\) with node set \(V_{n+1}\). Since \(V_{n+1}\) consists of only one strongly connected component, cr(\(K_{n+1}\)) = \(1 + \min_{e \in V_{n+1}}\{cr(K_{n+1}[V_{n+1} \setminus \{v\}])\}\). However, \(K_{n+1}[V_{n+1} \setminus \{v\}] = K_n\), for any \(v \in V_{n+1}\), and hence by the inductive hypothesis cr(\(K_{n+1}\)) = \(n + 1\). \(\square\)

5.2. \(\mathcal{L}_n\): Succinctness of RE(\(\cap\))

In this section we want to construct a set of small regular expressions such that any expression defining their intersection must be large (that is, of double exponential size). Ideally, we would like to use the family of languages \((\mathcal{K}_n)_{n \in \mathbb{N}}\) for this as
we have shown in the previous section that they have a large star height, and thus by Theorem 1 cannot be defined by small expressions. Unfortunately, this is not possible as the alphabet of \((\mathcal{K}_n)_{n \in \mathbb{N}}\) grows quadratically with \(n\).

Therefore, we will introduce in this section the family of languages \((\mathcal{L}_n)_{n \in \mathbb{N}}\) which is a binary encoding of \((\mathcal{K}_n)_{n \in \mathbb{N}}\) over a fixed alphabet. Thereto, let \(n \in \mathbb{N}\) and recall that \(\mathcal{K}_n\) is defined over the alphabet \(\Sigma_n = \{a_{ij} \mid i, j \in [0, n - 1]\}\). Now, for \(a_{ij} \in \Sigma_n\), define the function \(\rho_n\) as

\[
\rho_n(a_{ij}) = \#\text{enc}(j)\#\text{enc}(i)\#\text{enc}(i + 1)\Delta \cdots \#\text{enc}(n - 1)\Delta,
\]

where \(\Delta \notin \Sigma_n\) and \(\text{enc}(k)\), for \(k \in \mathbb{N}\), denotes the \([\log(n)]\)-bit marked number encoding \(k\) as defined in the proof of Theorem 8. So, the encoding starts by the encoding of the second index, followed by an ascending sequence of encodings of all numbers from the first index to \(n - 1\). We extend the definition of \(\rho_n\) to strings in the usual way: \(\rho_n(a_{00}, \ldots, a_{k-1, n-1}) = \rho_n(a_{0a_1}), \ldots, \rho_n(a_{0k-1})\). We now ready to define \(\mathcal{L}_n\).

**Definition 14.** Let \(\Sigma = \{0, 1, \tilde{0}, \tilde{1}, \$, \#, \Delta\}\). For \(n \in \mathbb{N}\), \(\mathcal{L}_n = \{\rho_n(w) \mid w \in \mathcal{K}_n\}\).

For instance, for \(n = 3\), \(a_{01, 1_2} \in \mathcal{K}_3\) and hence \(\rho_3(a_{01, 1_2}) = \#01\tilde{0}0\Delta 0\tilde{1}1\Delta \#1\tilde{0}0\#11\Delta 1\tilde{1}\Delta \in \mathcal{L}_3\). We now show that this encoding does not affect the star height.

**Lemma 15.** For any \(n \in \mathbb{N}\), \(sh(\mathcal{L}_n) = n\).

**Proof.** Let \(n \in \mathbb{N}\). We first show that \(sh(\mathcal{L}_n) \leq n\). By Lemma 13, \(sh(\mathcal{K}_n) = n\), and hence there exists a regular expression \(r_\mathcal{K}\), such that \(L(r_\mathcal{K}) = \mathcal{K}_n\) and \(sh(r_\mathcal{K}) = n\). Let \(r_\mathcal{L}\) be the regular expression obtained from \(r_\mathcal{K}\) by replacing every symbol \(a_{ij}\) by \(\rho_n(a_{ij})\). Obviously, \(L(r_\mathcal{L}) = \mathcal{L}_n\) and \(sh(r_\mathcal{L}) = n\), and hence \(sh(\mathcal{L}_n) \leq n\).

We now show that \(sh(\mathcal{L}_n) \geq n\). The proof is along the same lines as the proof of Lemma 13. We first show that \(\mathcal{L}_n\) is bideterministic and can then determine its star height by looking at the minimal DFA accepting it. In fact, the reason for the slightly involved encoding of \(\mathcal{K}_n\) is precisely the bideterminism property.

To show that \(\mathcal{L}_n\) is bideterministic, we now construct the minimal DFA \(A_\mathcal{L}^n\) accepting \(\mathcal{L}_n\). Here, \(A_\mathcal{L}^n\) will consist of \(n\) identical sub-automata \(B_0^n\) to \(B_n^{n-1}\) defined as follows. For any \(i \in [0, n - 1]\), \(B_i^n = (Q^1, q_i^n, \delta_i, F^1)\) is the smallest automaton for which \(Q^1\) contains distinct states \(q_0^i, q_1^i, \ldots, q_i^i\) and \(p_0^i, p_1^i, \ldots, p_{i-1}^i\) such that for any \(j \in [0, n - 1]\),

- \((q_j^i, \text{enc}(j)) \in \delta_i^*\), and for all \(w \neq \text{enc}(j)\) \((q_j^i, w) \notin \delta_i^*\); and
- \((q_j^i, \#\text{enc}(j), p_j^i) \in \delta_i^*\), and for all \(w \neq \#\text{enc}(j)\) \((q_j^i, w, p_j^i) \notin \delta_i^*\).

As an example, Fig. 2 shows \(B_3^n\). Now, \(A_\mathcal{L}^n = (Q, q_0^n, \delta, F)\) is defined as \(Q = \bigcup_{i \leq n} Q^1\), \(F = \{q_0^{n-1}\}\) and \(\delta = \bigcup_{i \leq n} \delta_i \cup \{(p_j^i, \$, q_j^i) \mid i, j \in [0, n - 1]\}\). Fig. 3 shows \(A_\mathcal{L}^5\).

To see that \(A_\mathcal{L}^n\) indeed accepts \(\mathcal{L}_n\), notice that after reading a substring \#\text{enc}(i), for some \(i\), the automaton moves to sub-automaton \(B_i^n\). Then, after passing through \(B_i^n\) which ends by reading a new substring \#\text{enc}(j), the automaton moves to state \(q_j^i\) of sub-automaton \(B_i^n\). This ensures that the subsequent ascending sequence of numbers starts with \text{enc}(i). Hence, the automaton checks correctly whether the numbers which should be equal, are equal.

Furthermore, \(A_\mathcal{L}^n\) is bideterministic and minimal. To see that it is bideterministic, notice that all states except the \(q_j^i\) only have one incoming transition. Furthermore, the \(q_j^i\) each have exactly two incoming transitions, one labeled with \$ and one labeled with \#. Minimality follows immediately from the fact that \(A_\mathcal{L}^n\) is both trim and bideterministic.

Since \(\mathcal{L}_n\) is bideterministic and \(A_\mathcal{L}^n\) is the minimal trim DFA accepting \(\mathcal{L}_n\), it follows from Theorem 3(3) that \(sh(\mathcal{L}_n) = cr(A_\mathcal{L}^n)\). Therefore, it suffices to show that \(cr(A_\mathcal{L}^n) \geq n\). Thereto, observe that \(A_\mathcal{L}^n\), the minimal DFA accepting \(\mathcal{K}_n\), is a minor of \(A_\mathcal{K}^n\). Indeed, we can contract edges and remove nodes from \(A_\mathcal{K}^n\) such that the only remaining nodes are \(q_0^n\) to \(q_{n-1}^n\), and such that they form a complete graph. Now, since it is shown in Lemma 13 that \(cr(A_\mathcal{K}^n) = n\) and by Lemma 2 and the fact that \(A_\mathcal{K}^n\) is a minor of \(A_\mathcal{L}^n\), we know that \(cr(A_\mathcal{L}^n) \geq cr(A_\mathcal{K}^n)\), it follows that \(sh(\mathcal{L}_n) = cr(A_\mathcal{L}^n) \geq n\). This completes the proof. □

Furthermore, it can be shown that \(\mathcal{L}_n\) can be described as the intersection of a set of small regular expressions.

**Lemma 16.** For every \(n \in \mathbb{N}\), there are regular expressions \(r_1, r_2, \ldots, r_m\), with \(m = 4n + 3\), each of size \(O(n)\), such that \(\bigcap_{i \leq m} L(r_i) = \mathcal{L}_2^n\).
Theorem. Let $L^n$ be defined over the alphabet $\Sigma = \{0, 1, \overline{0}, \overline{1}, \#, \Delta\}$. Let $N = \{0, 1, \overline{0}, \overline{1}\}$, $D = \{\$, $\#, \Delta\}$ and for any $\sigma \in \Sigma$, let $\Sigma_\sigma = \Sigma \setminus \{\sigma\}$. The expressions are as follows. The format of the string has to be correct:

$$(\#N^n \$N^n \Delta(N^n \Delta)^+)^*.$$  

Every number should be properly marked:

$$(D^+ ((0 + 1)^{*} \overline{10}^n + \overline{0}^n))^*.$$  

Every number before a $\$$ should be equal to the number following the next $\$$. We define two sets of regular expressions. First, for all $i \in [0, n - 1]$, the $(i + 1)$th bit of the number before an even $\$", i.e., a $\$$ preceded by an even number of $\$$'s in the string, should be equal to the $(i + 1)$th bit of the number after the next $\$$. Second, for all $i \in [0, n - 1]$, the $(i + 1)$th bit of the number before an odd $\$, i.e., a $\$ which is not even, should be equal to the $(i + 1)$th bit of the number after the next $\$$. The expressions are

$$((\#N^i \$N^i N^i)=1)),$$

$$\Sigma^*_\sigma(\#N^i \$N^i N^i)=1)),$$

$$((\#N^i \$N^i N^i)=1)),$$

$$\Sigma^*_\sigma(\#N^i \$N^i N^i)=1)),$$

Every two (marked) numbers surrounding a $\Delta$ should differ by exactly one. Again, we define two sets of regular expressions. First, for all $i \in [0, n - 1]$, the $(i + 1)$th bit of the number before an even $\Delta$ should properly match the $(i + 1)$th bit of the next number:

$$(\#\Sigma^*_\sigma((\Delta + \$) \Sigma^i((0 + \overline{0}) \Sigma^i(1 + 0) + (1 + \overline{1}) \Sigma^i(0 + 1)) \Sigma^{n-i-1})^+((\Delta + \$) \Sigma^i + \varepsilon) \Delta)^*.$$  

Second, for all $i \in [0, n - 1]$, the $(i + 1)$th bit of the number before an odd $\Delta$ should properly match the $(i + 1)$th bit of the next number:

$$(\#\Sigma^*_\sigma \Sigma^i((\Delta + \$) \Sigma^i((0 + \overline{0}) \Sigma^i(1 + 0) + (1 + \overline{1}) \Sigma^i(0 + 1)) \Sigma^{n-i-1})^+((\Delta \Sigma^i + \varepsilon) \Delta)^*.$$  

Every ascending sequence of numbers should end with $\text{enc}(2^n - 1) = 1^n - 1$:

$$(\#\Sigma^*_\sigma 1^{n-1} \overline{1} \Delta)^*.$$

Although it is not our main interest, we can now obtain the following by combining Theorem 1, and Lemmas 15 and 16.

Corollary 17. For any $n \in \mathbb{N}$, there exists an RE($\cap$) of size $\Theta(n^2)$ such that any (standard) regular expression defining $L(r)$ is of size at least $2^{\frac{1}{2}}(2^n - 1) - 1.$
Let \( n \in \mathbb{N} \). By Lemma 16 there exists a linear number of regular expressions of linear size, such that their intersection defines \( L_{2^n} \). Hence, there also exists an RE(\( \cap \)) \( r \) of size \( \Theta(n^2) \) defining \( L_{2^n} \). Furthermore, by Lemma 15, \( \text{sh}(L_{2^n}) = 2^n \) and therefore by Theorem 1 any regular expression defining \( L_{2^n} \) is of size at least \( 2^{\frac{3^n}{2^{n-1}} - 1} \).

5.3. \( M_n \): Succinctness of RE(\( \& \))

In this section we will finally show that RE(\( \& \)) are double exponentially more succinct than standard regular expressions. We do this by simulating the intersection of the regular expressions obtained in the previous section, by the interleaving of related expressions, similar to the simulation of RE(\( \cap \)) by RE(\( \& \)) in Section 4. This approach will partly yield the following family of languages. For any \( n \in \mathbb{N} \), define

\[
M_n = \{ \text{pump}_{[\log n]+3}(w) \mid w \in L_n \}.
\]

As \( M_n \) is very similar to \( L_n \), we can easily extend the result on the star height of \( L_n \) (Lemma 15) to \( M_n \):

**Lemma 18.** For any \( n \in \mathbb{N} \), \( \text{sh}(M_n) = n \).

**Proof.** The proof is along the same lines as the proof of Lemma 15. Again, \( M_n \) is bideterministic as witnessed by the minimal DFA \( A_n^M \) accepting \( M_n \). In fact, \( A_n^M \) is simply obtained from \( A_n^L \) by replacing each transition over a symbol \( a \) by a sequence of states reading \( a^m c^m \), with \( m = 4[\log n] + 3 \). Some care has to be taken, however, for the states \( q_i^j \), as these have two incoming transitions: one labeled with \$ and one labeled with \$\$. Naively creating these two sequences of states leading to \( q_i^j \), we obtain a non-minimal DFA. However, if we merge the last \( m \) states of these two sequences (the parts reading \( c^m \)), we obtain the minimal trim DFA \( A_n^M \) accepting \( M_n \). Then, the proof proceeds exactly as the proof of Lemma 15. \( \square \)

However, the language we will eventually define will not be exactly \( M_n \). Therefore, we need an additional lemma, for which we first introduce some notation. For \( k \in \mathbb{N} \), and an alphabet \( \Sigma \), we define \( \Sigma^{(k)} \) to be the language defined by the expression \( (\bigcup_{\sigma \in \Sigma} \sigma^k)^* \), i.e., all strings which consist of a sequence of blocks of identical symbols of length \( k \). Further, for a language \( L \), define \( \text{index}(L) = \max(i \mid i \in \mathbb{N} \land \exists w, w' \in \Sigma^*, a \in \Sigma \text{ such that } w a w' \in L) \). Notice that \( \text{index}(L) \) can be infinite. However, we will only be interested in languages for which it is finite, as in the following lemma.

**Lemma 19.** Let \( L \) be a regular language, and \( k \in \mathbb{N} \), such that \( \text{index}(L) \leq k \). Then, \( \text{sh}(L) \cdot |\Sigma| \geq \text{sh}(L \cap \Sigma^{(k)}) \).

**Proof.** Let \( L \) be a regular language, and \( k \in \mathbb{N} \), such that \( \text{index}(L) \leq k \). We show that, given a non-returning state-labeled NFA \( A \) accepting \( L \), we can construct an NFA \( B \) such that \( L(B) = L \cap \Sigma^{(k)} \) and \( B \) is a minor of \( A \).

We show how this implies the lemma. Let \( A \) be any non-returning state-labeled NFA of minimal cycle rank accepting \( L \), and let \( B \) be as constructed above. First, since \( B \) is a minor of \( A \) it follows from Lemma 2 that \( \text{cr}(B) \leq \text{cr}(A) \). Furthermore, by Egan’s Theorem (Theorem 3(1)), \( \text{cr}(B) \geq \text{sh}(L(B)) = \text{sh}(L \cap \Sigma^{(k)}) \), and hence \( \text{cr}(A) \geq \text{sh}(L \cap \Sigma^{(k)}) \). Now, since \( A \) is a non-returning state-labeled NFA of minimal cycle rank, we can conclude from Theorem 3(2) that \( \text{sh}(L) \cdot |\Sigma| \geq \text{cr}(A) \), and thus \( \text{sh}(L) \cdot |\Sigma| \geq \text{sh}(L \cap \Sigma^{(k)}) \).

We conclude by giving the construction of \( B \). Thereto, let \( A = (Q, q_0, \delta, F) \) be a non-returning state-labeled NFA. Now, for any \( q \in Q \) and \( a \in \Sigma \), define the function \( \text{in}_a(q) \) as follows:

\[
\text{in}_a(q) = \max(i \mid i \in \mathbb{N} \cup \{0\} \land \exists w \in \Sigma^* \text{ such that } (q_0, w d^i, q) \in \delta^i)
\]

Notice that as \( i \) can be equal to zero and \( \text{index}(L) \) is finite, \( \text{in}_a(q) \) is well defined for any state which is not useless. Intuitively, \( \text{in}_a(q) \) represents the maximal number of \( a \) symbols which can be read when entering state \( q \).

Now, the following algorithm transforms \( A \), accepting \( L \), into \( B \), accepting \( L \cap \Sigma^{(k)} \). Repeat the following two steps, until no more changes are made:

1. Apply one of the following rules, if possible:
   - (a) If exists \( q, q' \in Q, a \in \Sigma \), with \( (q, a, q') \in \delta \), such that \( \text{in}_a(q') > \text{in}_a(q) + 1 \) then remove \( (q, a, q') \) from \( \delta \).
   - (b) If exists \( q, q' \in Q, a \in \Sigma \), with \( (q, a, q') \in \delta \) and \( q \neq q_0 \), such that \( \text{in}_{\text{symbol}(q)}(q) < k \) and \( \text{symbol}(q) \neq a \) then remove \( (q, a, q') \) from \( \delta \).
   - (c) If exists \( q \in F, q \neq q_0 \), such that \( \text{in}_{\text{symbol}(q)}(q) < k \) then make \( q \) non-final, i.e., remove \( q \) from \( F \).
2. Remove all useless states from \( A \), and recompute \( \text{in}_a(q) \) for all \( q \in Q, a \in \Sigma \).

It remains to show that \( B \), the automaton obtained when no more rules can be applied, is the desired automaton. That is, that \( B \) is a minor of \( A \) and that \( L(B) = L(A) \cap \Sigma^{(k)} \). It is immediate that \( B \) is a minor of \( A \) since \( B \) is obtained from \( A \) by only removing transitions or states.

To show that \( L(B) = L(A) \cap \Sigma^{(k)} \), we first prove that \( L(A) \cap \Sigma^{(k)} \subseteq L(B) \). Thereto, let \( A_1, A_2, \ldots, A_n \), with \( A_1 = A \) and \( A_n = B \), be the sequence of NFAs produced by the algorithm, where each \( A_i \) is obtained from \( A_{i-1} \) by applying exactly one rule and possibly removing useless states. It suffices to show that for all \( i \in [1, n - 1] \), \( L(A_i) \cap \Sigma^{(k)} \subseteq L(A_{i+1}) \cap \Sigma^{(k)} \), as \( L(A) \cap \Sigma^{(k)} \subseteq L(B) \cap \Sigma^{(k)} \subseteq L(B) \) then easily follows.

Before proving \( L(A_i) \cap \Sigma^{(k)} \subseteq L(A_{i+1}) \cap \Sigma^{(k)} \), we introduce some notation. For any \( q \in Q, a \in \Sigma \), we define

\[
\text{out}_a(q) = \max\{i \mid i \in \mathbb{N} \cup \{0\} \land \exists w \in \Sigma^*, q_f \in F \text{ such that } (q, a^i w, q_f) \in \delta^i\}.
\]
Here, \( \omega(a) \) is similar to \( \omega(a) \) and represents the maximal number of \( a \)'s which can be read when leaving \( q \). Since of course \( L(A_i) \subseteq L(A) \) holds for all \( i \in [1, n] \), also \( \text{index}(A_i) \leq k \). Therefore, for any \( A_i \), any state \( q \) of \( A_i \) and any \( a \in \Sigma 
abla \omega(q) + \omega(a) \leq k \).

We are now ready to show that \( L(A_i) \cap \Sigma^{(k)} \subseteq L(A_{i+1}) \cap \Sigma^{(k)} \). Thereto, we prove that for any \( w \in L(A_i) \cap \Sigma^{(k)} \), any accepting run of \( A_i \) on \( w \) is still an accepting run of \( A_{i+1} \) on \( w \). More precisely, we prove for every rule separately that if \( A_{i+1} \) is obtained from \( A_i \) by applying this rule, then the assumption that the removed state or transition is used in an accepting run of \( A_i \) on \( w \) leads to a contradiction.

For rule (1a), suppose transition \((q, a, q') \) is removed but \((q, a, q') \) occurs in an accepting run of \( A_i \) on \( w \), i.e., \( w = uav \) for \( u, v \in \Sigma^* \), \((q_0, u, q) \in \delta^* \) and \((q', v, q') \in \delta^* \), for some \( q_i \in \mathcal{F} \). Since \( w \in \Sigma^{(k)} \), there exists a \( j \in [0, k - 1] \) and \( u', v' \in \Sigma^* \) such that \( u = u'a \) and \( v = a^{-j}v' \). It immediately follows that \( \omega(q) \geq j \) and \( \omega(a) \geq k - j - 1 \). Then, since \( \omega(q) > \omega(q) + 1 \), we have \( \omega(q) + \omega(a) \geq k - j - 1 + 1 = k \). However, this contradicts Eq. (1).

For rules (1b) en (1c), we describe the obtained contradictions less formal. For rule (1b), if the transition \((q, a, q') \) with symbol \( q \neq a \) is used in an accepting run on \( w \), then \( q \) is entered after reading \( k \) symbol\( q \) symbols, and hence \( \text{index}(q) \geq k \). Contradiction. For rule (1c), again, if \( q \) is the accepting state of some run on \( w \), then \( q \) must be preceded by \( k \) symbol\( q \) symbols, and hence \( \text{index}(q) \geq k \), contradiction.

We now show \( L(B) \subseteq L(A) \cap \Sigma^{(k)} \). Thereto, let \( B = (Q^B, \Sigma^B, \delta_B, F_B) \). We first make the following observation. Let \((q, a, q') \) be a transition in \( \delta_B \). Since \( B \) does not contain useless states, it is easy to see that \( \omega(q) \geq \omega(q) + 1 \). As rule (1a) cannot be applied to transition \((q, a, q') \), we now obtain the following equality:

\[ \forall q, q' \in Q^B, a \in \Sigma, \text{ with } (q, a, q') \in \delta_B : \omega(a) + 1 = \omega(q'). \]

We are now ready to show that \( L(B) = L(A) \cap \Sigma^{(k)} \). Since \( L(B) \subseteq L(A) \) definitely holds, it suffices to show that \( L(B) = L(A) \cap \Sigma^{(k)} \), i.e., \( B \) only accepts strings in \( \Sigma^{(k)} \).

Towards a contradiction, suppose that \( B \) accepts a string \( w \notin \Sigma^{(k)} \). Then, there must exist an \( i \in [1, k - 1] \) and \( a \in \Sigma, u \in L(e + \Sigma^* (\Sigma \setminus \{a\}) \vee v \in L(e + (\Sigma \setminus \{a\}) \vee \Sigma^*) \) such that \( w = uav \). Furthermore, since \( w \in L(B) \), there also exist states \( p_0, p_1, \ldots, p_i \) such that \((q_0, u, p_0) \in \delta^* \) and \((p_i, v, q') \in \delta^* \) for some \( q_i \in \mathcal{F} \) and \((p_i, a, p_{i+1}) \in \delta_B \) for all \( j \in [0, i - 1] \).

We first argue that \( \omega(p_i) = k \). By Eq. (1) it suffices to show that \( \omega(p_i) \geq k \). Notice that, as \( i > 0 \), symbol\( p_i \) is a. We consider two cases. If \( v = e \), then \( p_0 = q_0 \). As \( A \) is non-returning and we did not introduce any new transitions, \( B \) also is non-returning and hence \( \omega(q_0) = 0 \) should hold. Otherwise, if \( u = uav \) for \( v \in \Sigma^* \) and \( b \neq a \), then \( p_0 \) has an incoming \( b \)-transition. As \( B \) is state-labeled, \( p_0 \) does not have an incoming transition, and hence, again, \( \omega(q_0) = 0 \) should hold. This concludes the proof. \( \square \)

Now, we are finally ready to prove the desired theorem:

**Theorem 20.** For every \( n \in \mathbb{N} \), there are regular expressions \( s_1, s_2, \ldots, s_m \) with \( m = 4n + 3 \), each of size \( \mathcal{O}(n) \), such that any regular expression defining \( L(s_1) \cap L(s_2) \cdots \cap L(s_m) \) is of size at least \( 2^{\pi^{(2n-8)}} - 1 \).

**Proof.** Let \( n \in \mathbb{N} \), and let \( r_1, r_2, \ldots, r_m \), with \( m = 4n + 3 \), be the regular expressions obtained in Lemma 16 such that \( \bigcap_{i=1}^m L(r_i) = L^2n \).

Now, similar to Lemma 9, it is shown in [24], that given \( r_1, r_2, \ldots, r_m \), it is possible to construct regular expressions \( s_1, s_2, \ldots, s_m \) such that (1) for all \( i \in [1, m] \), \( |s_i| \leq 2|r_i| \), and if we define \( \mathcal{N}^{(2n)} = L(s_1) \cap L(s_2) \cap \cdots \cap L(s_m) \), then (2) \( \text{index}(\mathcal{N}^{(2n)}) \leq m \), and (3) for every \( w \in \Sigma^* \), \( w \in \bigcap_{i=1}^m L(r_i) \) iff \( \text{pump}_m(w) \in \mathcal{N}^{(2n)} \). Furthermore, it follows immediately from the construction in [24] that any string in \( \mathcal{N}^{(2n)} \cap \Sigma^{(m)} \) is of the form \( a_1^{n_1}a_2^{n_2}a_3^{n_3} \cdots a_m^{n_m} \), i.e., \( \text{pump}_m(w) \) for some \( w \in \Sigma^* \).

Since \( \bigcap_{i=1}^m L(r_i) = L^2n, \mathcal{M}_{2n} = \{ \text{pump}_m(w) \mid w \in L^2n \} \), it hence follows that \( \mathcal{M}_{2n} = \mathcal{N}^{(2n)} \cap \Sigma^{(m)} \). As furthermore, by Lemma 18, \( \text{sh}(\mathcal{M}_{2n}) = 2^n \) and \( \text{index}(\mathcal{N}^{(2n)}) \leq m \), it follows from Lemma 19 that \( \text{sh}(\mathcal{M}_{2n}) \geq \frac{\text{sh}(\mathcal{M}_{2n})}{|\Sigma|} = 2^n = \frac{2^{2n}}{8} \). So, \( \mathcal{N}^{(2n)} \) can be described by the interleaving of the expressions \( s_1 \) to \( s_m \), each of size \( \mathcal{O}(n) \), but any regular expression defining \( \mathcal{M}_{2n} \) must, by Theorem 1, be of size at least \( 2^{\pi^{(2n-8)}} - 1 \). This concludes the proof. \( \square \)

**Corollary 21.** For any \( n \in \mathbb{N} \), there exists an \( \mathcal{O}(8^n) \) \( r_n \) of size \( \mathcal{O}(n^2) \) such that any regular expression defining \( L(r_n) \) must be of size at least \( 2^{\pi^{(2n^2-8)}} - 1 \).

This completes the paper. As a final remark, we note that all lower bounds in this paper make use of a constant size alphabet and can furthermore easily be extended to a binary alphabet. For any language over an alphabet \( \Sigma = \{a_1, a_2, \ldots, a_k\} \), we obtain a new language over the alphabet \( \{b, c\} \) by replacing, for any \( i \in [1, k] \), every symbol \( a_i \) by \( b^{\text{index}(a_i)+1} \). Obviously, the size of a regular expression for this new language is at most \( k + 1 \) times the size of the original expression, and the lower bounds on the number of states of DFAs trivially carry over. Furthermore, it is shown in [26] that this transformation does not affect the star height, and hence the lower bounds on the sizes of the regular expression also carry over.
Acknowledgements

I thank Frank Neven and the reviewer of TCS for helpful suggestions, and Hermann Gruber and Markus Holzer for informing me about their results on the succinctness of the interleaving operator.

References