Transformation Between Regular Expressions and ω-Automata

Christof Löding[1] **and Andreas Tollkötter**[2]

1 **RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Germany**
loeding@informatik.rwth-aachen.de
2 **RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Germany**
andreas.tollkoetter@rwth-aachen.de

Abstract

We propose a new definition of regular expressions for describing languages of ω-words, called ∞-regular expressions. These expressions are obtained by adding to the standard regular expression on finite words an operator ∞ that acts similar to the Kleene-star but can be iterated finitely or infinitely often (as opposed to the ω-operator from standard ω-regular expressions, which has to be iterated infinitely often). We show that standard constructions between automata and regular expressions for finite words can smoothly be adapted to infinite words in this setting: We extend the Glushkov construction yielding a simple translation of ∞-regular expressions into parity automata, and we show how to translate parity automata into ∞-regular expressions by the classical state elimination technique, where in both cases the nesting of the $*$ and the ∞ operators corresponds to the priority range used in the parity automaton. We also briefly discuss the concept of deterministic expressions that directly transfers from standard regular expressions to ∞-regular expressions.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Infinity Regular Expressions, Parity Automata

Digital Object Identifier 10.4230/LIPIcs.MFCS.2016.88

1 Introduction

Regular expressions play a central role in formal language theory. They are a widespread formalism for specifying patterns, and the tight connection to finite automata provides many algorithmic techniques for dealing with regular expressions (see any standard textbook on formal language theory, e.g., [7, 8]). The theory of regular languages can be lifted to languages of infinite words with a richer landscape of automaton models (see, e.g., [15, 16, 12, 8]). The basic model of Büchi automata, which accept if a run visits an accepting state infinitely often, is more expressive in its nondeterministic variant than in its deterministic one. To obtain a determinization theorem as in the case of finite words, one needs to introduce more complex acceptance conditions like the parity, Rabin, or Muller conditions. There are also regular expressions for infinite words, called ω-regular expressions. These are of the form $r_1 \cdot s_1^\omega + \cdots + r_n \cdot s_n^\omega$ for standard regular expressions r_i, s_i. An infinite word α matches such an expression if there is an index i such that α has a finite prefix matching r_i, followed by an infinite concatenation of finite words matching s_i. While this definition makes the translation between Büchi automata and ω-regular expressions easy, it appears to be ad hoc and made with this very purpose in mind. In particular, the non-inductive definition makes it difficult to embed ω-regular expressions into other specification formalisms for infinite words (or trees). Such formalisms usually rely on standard regular expressions like, e.g., PDL

with regular programs [5], regular temporal logic that extends LTL by regular expressions [10], or the industrial specification language PSL that also combines temporal logic with regular expressions [13].

In this paper, we propose a simple inductive definition of regular expressions for infinite words, which we refer to as ∞-regular expressions. These are obtained by adding to the standard regular expressions an operator ∞ that behaves similar to the Kleene-star $*$ but can be iterated both finitely or infinitely often. Consequently, our expressions define mixed sets of finite and infinite words. This also solves the problem of the concatenation of two expressions defining infinite words. In a concatenation $r \cdot s$ of two ∞-regular expressions, we keep the infinite words defined by r, and concatenate the finite words defined by r with all words defined by s. This combined use of finite and infinite words is certainly not new. For example, in [11] the rational subsets of the set of finite and infinite words are defined from the single letters by closure under union, concatenation, finite, and infinite iteration (applied to the correct type of words, respectively). However, we are not aware of any simple inductive definition of regular expressions for infinite words as we propose it here.

We show that constructions between regular expressions and finite automata smoothly extend to ∞-regular expressions and parity automata. The acceptance condition of a parity automaton is defined by assigning priorities (natural numbers) to the states. By convention, the even priorities represent "good" states and the odd priorities represent "bad" states. The highest priority that is visited infinitely often decides about acceptance: the run is rejecting if it is odd, and accepting if it is even. These priorities naturally reflect the nesting of the $*$-operator and the ∞-operator in our expressions, where the $*$-operators correspond to odd priorities (because they can only be iterated finitely often), and the ∞-operators to even priorities (because they can be iterated finitely or infinitely often).

Based on this idea, we adapt the Glushkov construction (see [1]) to ∞-regular expressions and parity automata. The Glushkov construction translates regular expressions into automata using the occurrences of letters in the expressions as states, and inserting the transitions between these occurrences according to the operators of the expression. We use the same transition structure, assigning priorities to the transitions that reflect the nesting of the iteration operators.

For the other direction, from automata into expressions, we adapt the classical state elimination technique (as described, e.g., in [14]) to parity automata. In order to capture the semantics of the priorities in this translation, the elimination of the states has to be done in order of increasing priorities. Otherwise the construction is the same as for finite words.

We also consider the class of deterministic (or one-unambiguous) ∞-regular expressions. As for the case of finite words [2], these are, intuitively speaking, expressions for which matching can be done deterministically without lookahead on the input. As for finite words, deterministic ∞-regular expressions are precisely those for which the Glushkov automaton is deterministic. While we do not have deep results on deterministic ∞-regular expressions, we believe that this class of expressions could be of interest, for example, as specification formalism in synthesis problems. In the automata-theoretic approach to reactive synthesis, determinization of automata on infinite words is one of the main bottlenecks (see [9]). So, it could be interesting to write at least parts of the specification in a formalism that easily translates into deterministic automata.

The paper is structured as follows. In Section 2 we introduce the ∞-regular expressions and define a hierarchy of expressions corresponding to the nesting of the iteration operators. In Section 3 we present the adaption of the Glushkov construction to our setting. In Section 4 we show how state elimination can be used to translate parity automata into ∞-regular

expressions. In Section 5 we define deterministic ∞-regular expressions, and we conclude in Section 6. Detailed proofs can be found in [17].

2 Definitions and Examples

We use Γ to denote an alphabet, that is, a finite set of symbols. As usual, Γ^* and Γ^ω denote the set of finite and infinite words over Γ. Furthermore, we let $\Gamma^\infty := \Gamma^* \cup \Gamma^\omega$ be the set of all finite and infinite words over Γ. Regular expressions are defined in the standard way, that is, they are built up inductively from single letters $a \in \Gamma$, the empty word ε, and $\emptyset$ by the operators $+$ (union), $\cdot$ (concatenation), and * (finite iteration). Since we consider different classes of regular expressions, we explicitly refer to these regular expressions as *$*$-regular expressions*.

The usual definition of regular expressions for infinite words uses expressions of the form $r_1 s_1^\omega + \cdots + r_n s_n^\omega$ (see [15]). We refer to these expressions as *ω-regular expressions*. An infinite word α matches such an expression if there is an index i such that α has a finite prefix matching r_i, followed by an infinite concatenation of finite words matching s_i.

We introduce a new class of expressions that define mixed sets of finite and infinite words.

▶ **Definition 1** (∞ regular expressions)**.** The set of *∞-regular expressions* over the alphabet Γ is defined inductively as follows: ε, $\emptyset$, and every $a \in \Gamma$ is an expression. Given two expressions r and s, then the following are ∞-regular expressions as well: $r + s$, $r \cdot s$, r^*, r^∞.

The semantics of such an expression r is a language $L(r) \subseteq \Gamma^\infty$ of both finite and infinite words, defined as follows.

- $L(a) = \{a\}$, $L(\varepsilon) = \{\varepsilon\}$, $L(\emptyset) = \emptyset$
- $L(r + s) = L(r) \cup L(s)$
- $L(r \cdot s) = L_\omega(r) \cup (L_*(r) \cdot L(s))$
- $L(r^*) = (L_*(r))^* \cdot (L_\omega(r) \cup \{\varepsilon\})$
- $L(r^\infty) = L(r^*) \cup (L_*(r))^\omega$

Here, L_* and L_ω denote the subset of finite and infinite words of L, respectively, i.e. $L_*(r) = L(r) \cap \Gamma^*$ and $L_\omega(r) = L(r) \cap \Gamma^\omega$.

The following examples give an idea how ∞-expressions can be used. Trying to write ω-regular expressions for the corresponding sets of infinite words shows that ∞-regular expressions can express some natural properties more directly.

- $L_\omega((a^*b)^\infty)$ = the set of words which contain infinitely many b
- $L_\omega((a^\infty b)^*)$ = the set of words which contain finitely many b
- $L_\omega(\,((b+c)^\infty a(a+c)^* b)^\infty\,)$ = the set of words in which every a is followed by a b

It is not very difficult to see that ∞-regular expressions subsume the expressive power of the other two classes of regular expressions, as stated in the next proposition.

▶ **Proposition 2.**

1. *A language $L \subseteq \Gamma^*$ is regular iff there is an ∞-regular expression describing it.*
2. *A language $L \subseteq \Gamma^\omega$ is ω-regular iff there is an ∞-regular expression describing it.*
3. *A language $L \subseteq \Gamma^\infty$ is described by some ∞-regular expression iff $L \cap \Gamma^*$ is regular and $L \cap \Gamma^\omega$ is ω-regular*

Proof.

1. If a language is regular, then there is a $*$-regular expression r describing it. ∞-regular expressions are a generalization of $*$-expressions, so r is also an ∞-regular expression describing the language. Given an ∞-expression describing a language $L \subseteq \Gamma^*$, we can

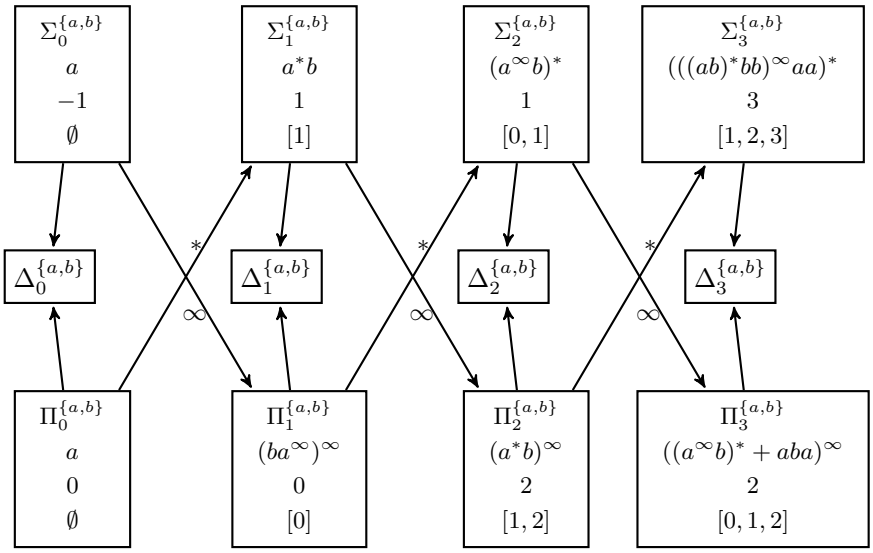


Figure 1 Illustration of the first four levels of the hierarchy $\mathcal{H}$. The second row is an example expression of the set and the third row shows the corresponding rank. The fourth row shows the interval of priorities which is used in the construction of a parity automaton from a regular expression in Section 3.

easily transform it into a $*$-regular expression by replacing every symbol r^∞ by r^*. This transformation does not change the set of finite words defined by the expression.

2. If $L \subseteq \Gamma^\omega$ is ω-regular, there is an ω-expression $r_1 s_1^\omega + \cdots + r_n s_n^\omega$ for it. By definition of the semantics of regular expressions, one can see that e^ω is equivalent to $e^\infty \cdot \emptyset$. Thus, L can be described by the ∞-expression $r_1 s_1^\infty \emptyset + \cdots + r_n s_n^\infty \emptyset$.
   To convert an ∞-regular expression into an ω-expression, one has to replace the different operators inductively. The proof is technical and is therefore left out. This result also follows directly from Theorem 10.
3. The third result is a consequence of the first two. Given an ∞-regular expression, we can transform it to obtain expressions for both $L \cap \Gamma^*$ and $L \cap \Gamma^\omega$ as shown above. If we know expressions r and s for $L \cap \Gamma^*$ and $L \cap \Gamma^\omega$, we can transform them to ∞-regular expression. Their union then describes L. ◀

For the translation between ∞-regular expressions and automata, the alternation of the two unary operators $*$ and ∞ plays a central role. Thus, we consider the hierarchy resulting from the nesting of these operators. The use of the names Π and Σ for the classes is borrowed from hierarchies like the Borel hierarchy or arithmetic hierarchy.

Let $\Sigma_0^\Gamma = \Pi_0^\Gamma = \{r \mid r \text{ is an } \infty\text{-regular expression over } \Gamma \text{ and contains no } * \text{ nor } \infty\}$, which are the "lowest" levels in this hierarchy as they do not use any degree of alternation. Inductively, we define Π_{n+1}^Γ to be the closure of Σ_n^Γ w.r.t. the operators $+$, $\cdot$, and ∞. Analogously, Σ_{n+1}^Γ is the closure of Π_n^Γ w.r.t. $+$, $\cdot$, and $*$. Finally, we set $\Delta_n^\Gamma = \Sigma_n^\Gamma \cap \Pi_n^\Gamma$.

The structure of the hierarchy is shown in Figure 1. The arrows indicate how the classes are built from other classes. For further illustration, here are some examples.

- Every $*$-regular expression lies in Σ_1^Γ.
- The expression $(a^*b)^\infty$ is in $\Pi_2^{\{a,b\}}$ but in no lower level.
- The expression $(a^\infty b)^*$ is in $\Sigma_2^{\{a,b\}}$ but in no lower level.
- The expression $a^* + b^\infty$ is in $\Delta_2^{\{a,b\}}$ but in no lower level.

Since the levels of the hierarchy are strictly growing, each expression is contained in infinitely many levels. We refer to the lowest level as the stage of an expression, as detailed in the following definition.

▶ **Definition 3** (Stage). For every ∞-regular expression r, there is a unique set Σ_n^Γ, Π_n^Γ or Δ_n^Γ, which is the lowest set in the hierarchy that contains r (Δ_n being lower than Σ_n and Π_n). We call that set the *stage* of r ($\mathrm{stg}(r)$).

For example, $(a^\infty b)^*$ is contained in $\Sigma_3^{\{a,b\}}$. Its stage, however, is $\Sigma_2^{\{a,b\}}$, as it is not the element of any level 1 set or $\Delta_2^{\{a,b\}}$.

Note that this hierarchy, which we call $\mathcal{H}$ in the following, only describes the expressions as syntactical objects, without regard to the semantic language. In fact, a semantic hierarchy of the same kind would collapse down to two levels (the hierarchy becomes strict in the context of deterministic expressions, as explained in Section 5).

▶ **Proposition 4.** *Let $L \subseteq \Gamma^\infty$ be ∞-regular. Then there is an ∞-regular expressions r describing L with $r \in \Pi_2^\Gamma$.*

Proof. This result is implied by the construction used in point 3 of Proposition 2. ◀

To better "categorize" the complexity of an expression in the hierarchy $\mathcal{H}$, we assign a single number to each class. Basically, this number corresponds to the highest priority that is used in Section 3 in the construction of parity automata from expressions of the corresponding level of the hierarchy.

▶ **Definition 5** (Rank). We assign a *rank* to every set in $\mathcal{H}$ (see also Figure 1)

$$\mathrm{rk}(\Sigma_n^\Gamma) = 2 \cdot \left\lfloor \frac{n+1}{2} \right\rfloor - 1, \quad \mathrm{rk}(\Pi_n^\Gamma) = 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor, \text{ and } \quad \mathrm{rk}(\Delta_n^\Gamma) = \min\{\mathrm{rk}(\Sigma_n^\Gamma), \mathrm{rk}(\Pi_n^\Gamma)\}.$$

We also define the rank of an expression r to be $\mathrm{rk}(r) = \mathrm{rk}(\mathrm{stg}(r))$. Note that every set Π_n^Γ has an even rank, and every Σ_n^Γ has an odd rank.

3 Glushkov Automaton

To convert a given ∞-regular expression to an automaton, we generalize the Glushkov construction which can be found in [1, 2] for finite words. We focus only on the set of ω-words defined by ∞-regular expression here. Since the automaton that we construct uses the same transition structure as the one from the classical construction for finite words, one can simply add a set of final states if one is interested in automata accepting both finite and infinite words. We start with the definition of parity automata, which is the model of automata for infinite words that we use (see, e.g., [6] for more information on parity automata).

▶ **Definition 6** (Parity automaton). A *parity automaton* is a tuple $\mathcal{A} = (Q, \Gamma, q_0, \delta, \gamma)$, where Q is the finite set of states, Γ is the alphabet, $q_0 \in Q$ is the initial state, $\delta : Q \times \Gamma \to 2^Q$ is the transition function, and γ is the priority assignment function. We use two different variants for γ; either $\gamma : Q \to \mathbb{N}$ assigns priorities to states, or $\gamma : Q \times Q \to \mathbb{N}$ assigns priorities to pairs of states, which corresponds to priority $\gamma(p, q)$ on all transitions from p to q.

A run of $\mathcal{A}$ on some word $w \in \Gamma^\infty$ is a word $\rho \in Q^\infty$ which "follows" δ, meaning $\rho(n+1) \in \delta(\rho(n), w(n))$ for all relevant n. $\mathcal{A}$ accepts an infinite word w if there is a run ρ on w which is accepting, meaning that the *highest* priority occurring infinitely often in ρ is *even*.

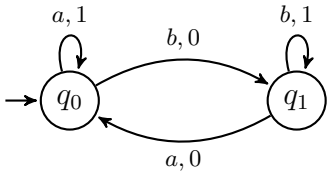


Figure 2 An automaton using transition priorities for the language $(a+b)^*(ab)^\omega$.

While the assignment of priorities to states is the usual definition, we use the assignment to transitions (pairs of states) in this section. An example of such an automaton is shown in Figure 2. It is easy to see that the two definitions of the priority function γ can be transformed into one another. From states to transitions, one can simply move the priorities to the outgoing transitions of a state. If the priorities are assigned to pairs of states, one can obtain an equivalent automaton with priorities on states by increasing the automaton by a factor of at most $|Q|$.

The states of the Glushkov automaton are the occurrences of letters in the expression. This is formalized using the definition of marking, given below.

▶ **Definition 7** (Marking)**.** Let r be an ∞-regular expression over Γ. We call $\sharp(r)$ a *marking of* r. It is defined as an ∞-regular expression over some subset $\Gamma' \subset \Gamma \times \mathbb{N}$ by replacing every $a \in \Gamma$ in r by a *unique* pair $(a, n) \in \Gamma \times \mathbb{N}$. We also write a_n instead of (a, n).

For the transitions of the Glushkov automaton the following definitions are used.

▶ **Definition 8.** Let $L \subseteq \Gamma^\infty$ be a language. We define

- $\text{first}_L = \{a \in \Gamma \mid \text{There is a } w \in L \text{ which begins with } a\}$
- $\text{last}_L = \{a \in \Gamma \mid \text{There is a } w \in L \text{ which ends with } a\}$
- $\text{follows}_L(b) = \{a \in \Gamma \mid \text{There is a } w \in L \text{ in which } a \text{ occurs directly after } b\}$

▶ **Definition 9** (Glushkov automaton)**.** Let r be an ∞-regular expression over the alphabet Γ. We define the *Glushkov automaton* $\mathcal{G}_r = (Q, \Gamma, q_0, \delta, \gamma)$. For that, let $\Gamma' \subset \Gamma \times \mathbb{N}$ be the set of symbols in $\sharp(r)$. The set of states $Q = \{q_0\} \cup \Gamma'$ contains one state for every symbol in r, as well as a distinct initial state. The transition function is defined as

$$\delta(q, a) = \begin{cases} \{a_n \in \Gamma' \mid a_n \in \text{first}_{L(\sharp(r))}\} & \text{if } q = q_0 \\ \{a_n \in \Gamma' \mid a_n \in \text{follows}_{L(\sharp(r))}(q)\} & \text{else.} \end{cases}$$

It allows the automaton to move from one symbol of $\sharp(r)$ to any succeeding symbol which matches with the read letter a. Note that for the case $q \neq q_0$, we use the fact that $\Gamma' \subset Q$, which is why follows(q) is well-defined. This definition of δ aligns with that for finite words from [2].

For the priority function, we use an assignment on edges: $\gamma : Q \times Q \to \mathbb{N}$. For any pair $p, q \in Q$ such that p or q equals q_0, or there is no transition between the two states, we can set $\gamma(p, q)$ to an arbitrary value, as it will occur at most once in any run. We will simply leave out the priority for these transitions in examples.

For the other cases, the priority is intuitively determined by the operators that induce the transition. To capture this formally, we need a few definitions that are illustrated with an example below. Let $a_n, b_m \in \Gamma'$ such that $b_m \in \delta(a_n, b)$. Let $R_* \ / \ R_\infty$ be the set of all sub-expressions of r of the form $s^* \ / \ s^\infty$, and

$$S = \{s \mid s \text{ is a sub-expression of } r \text{ with } a_n \in \text{last}_{L(\sharp(s))} \text{ and } b_m \in \text{first}_{L(\sharp(s))}\}.$$

Let $R'_* = \{s^* \in R_* \mid s \in S\}$, $R'_\infty = \{s^\infty \in R_\infty \mid s \in S\}$, and $R' = R'_* \cup R'_\infty$. These sets R'_* and R'_∞ can be seen as the "looping" expressions of the transition. The order $\prec$ on R' defined by $s \prec t$ if s is a sub-expression of t is a linear order because a_n and b_m exist exactly once in r.

We then set $\gamma(a_n, b_m) = \begin{cases} \blacklozenge & \text{if } R' = \emptyset \\ \mathrm{rk}(\max_\prec R'_\infty) & \text{if } R'_\infty \neq \emptyset \\ \mathrm{rk}(\min_\prec R'_*) & \text{else.} \end{cases}$

We use $\blacklozenge$ here as a dummy symbol for the minimal priority. These transitions never occur in a loop without a transition with $R' \neq \emptyset$.

This definition of γ ensures that the automaton always uses the "best looping transition" in r, meaning the highest even priority if possible, or the lowest odd priority otherwise.

We demonstrate the construction of $\mathcal{G}_r$ on the following example. Let the alphabet be $\Gamma = \{a, b, c\}$ and $r = (a((a + \varepsilon)b^\infty)^*)^\infty$. We want to note here that this is semantically not a useful expression and is only constructed to illustrate the definition. We use the marking $\sharp(r) = (a_1((a_2 + \varepsilon)b_1^\infty)^*)^\infty$. $\mathcal{G}_r$ is displayed in Figure 3.

This expression yields the sets $R_* = \{((a_2+\varepsilon)b_1^\infty)^*\}$ and $R_\infty = \{(a_1((a_2+\varepsilon)b_1^\infty)^*)^\infty, b_1^\infty\}$, which are all sub-expressions with a $*$ or an ∞ as the outermost operator.

Let $p = a_1$ and $q = a_2$. The set S for this pair of states is $S = \emptyset$, as there is no sub-expression of r that can start with a_2 and end with a_1. Hence, R' is empty as well and we set $\gamma(p, q) = \blacklozenge$. The intuitive explanation for this is that moving from a_1 to a_2 in the expression corresponds to a concatenation operation and not a looping operator.

Let $p = q = b_1$. In this case, $S = \{b_1, b_1^\infty, (a_2 + \varepsilon)b_1^\infty, ((a_2 + \varepsilon)b_1^\infty)^*\}$. This also gives non-empty sets $R'_* = \{((a_2 + \varepsilon)b_1^\infty)^*\}$ and $R'_\infty = \{b_1^\infty\}$. By definition, we assign $\gamma(p, q) = \mathrm{rk}(\max_\prec R'_\infty) = \mathrm{rk}(b_1^\infty) = 0$. The intuition is that there are two possible operators one could use to loop within the expression and follow a b_1 by another b_1. In this case, the priority reflects the "best" choice among these operators.

For the other priorities, note that moving back to a_1 from a_2 or b_1 is only possible via the outermost ∞-operator, hence these transitions have priority 2. Moving back between b_1 and a_2 is done through the $*$-operator, leading to priority 1. The transitions corresponding to concatenation operations and are assigned $\blacklozenge$ (which would be replaced by the minimal priority 0 used in the automaton).

Another significant example is the transition from a_2 to b_1. It might look like it would assigned the value $\blacklozenge$ as it corresponds to a concatenation operator but in fact the definition gives $\gamma(a_2, b_1) = 1$. That is because the transition can also be completed by the $*$-operator as follows: Read an a as a_2, skip b_1^∞ because it contains the empty word, use the $*$-operator to "loop around" to the beginning of the sub-expression, skip $(a_2 + \varepsilon)$ because it contains the empty word, finally read b as part of b_1^∞.

Not only does the Glushkov automaton accept the correct language, we can also find a relation between the rank of the expression r and the number of priorities in $\mathcal{G}_r$. This is captured by the following theorem.

▶ **Theorem 10.** *Let r be an ∞-regular expression.*
- *$\mathcal{G}_r$ accepts exactly $L_\omega(r)$*
- *$\mathcal{G}_r$ uses priorities up to at most $rk(r)$*
- *$\mathcal{G}_r$ uses $\mathcal{O}(|r|)$ many states*

Proof. The only idea of this proof is to convince oneself that the usage of "loops" in the expression, as it is described in the construction and the example, is correct. The formal

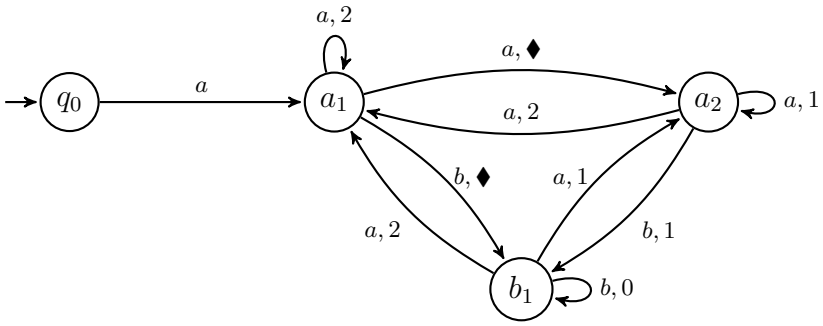


Figure 3 $\mathcal{G}_r$ for $r = (a((a + \varepsilon)b^\infty)^*)^\infty$.

proof is carried out by an equivalent inductive definition of the Glushkov automaton, which then admits an inductive correctness proof. The technical details are lengthy and can be found in [17]. ◀

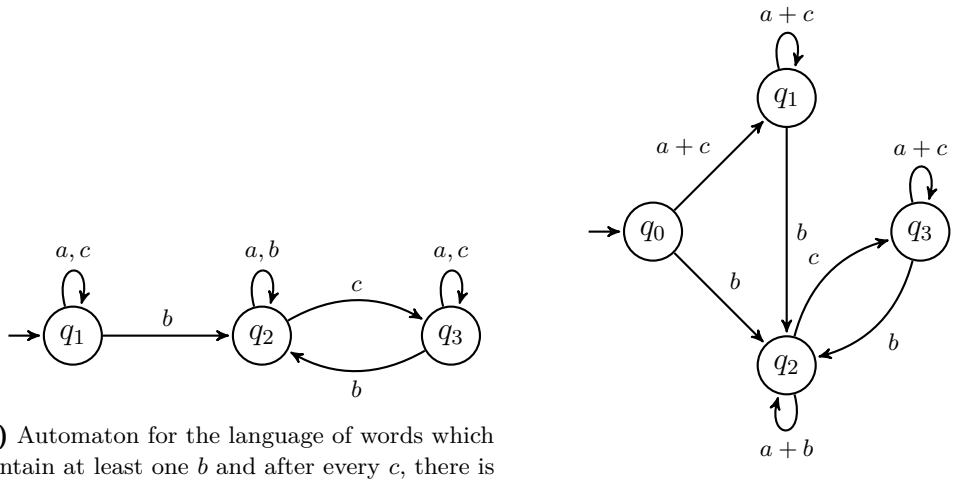
4 State Elimination

We now establish an algorithm for the reverse operation, i.e. converting a parity automaton to an equivalent ∞-regular expression. We use a variation of the *state elimination algorithm* which is known for regular languages of finite words [3] (see also [14]). In this section we work with priorities assigned to states.

`// Initialize new automaton`
$Q_0 := Q \dot\cup \{q_0'\}$;
$R_0 : Q_0 \times Q_0 \to \{r \mid r \text{ is an } \infty\text{-reg.exp.}\}$;
for $p, q \in Q_0$ **do**
  $$R_0(p,q) := \begin{cases} \sum\{a \in \Gamma \mid q \in \delta(p,a)\} & \text{if } p,q \in Q \\ \sum\{a \in \Gamma \mid q \in \delta(q_0,a)\} & \text{if } p = q_0', q \in Q \\ \emptyset & \text{if } q = q_0' \end{cases}$$
end
Let $\prec \subseteq Q \times Q$ be a linear order such that $x \prec y$ implies $\gamma(x) \leq \gamma(y)$;
for $i = 0$ *to* $|Q| - 1$ **do**
  `// Eliminate state with minimal priority`
  $q_e := \min_\prec(Q_i \setminus \{q_0'\})$;
  $$r_e := \begin{cases} (R_i(q_e,q_e))^\infty & \text{if } \gamma(q_e) \text{ is even} \\ (R_i(q_e,q_e))^* & \text{if } \gamma(q_e) \text{ is odd} \end{cases} ;$$
  $Q_{i+1} := Q_i \setminus \{q_e\}$;
  $R_{i+1} : Q_{i+1} \times Q_{i+1} \to \{r \mid r \text{ is an } \infty\text{-reg.exp.}\}$;
  for $p, q \in Q_{i+1}$ **do**
    $R_{i+1}(p,q) := R_i(p,q) + (R_i(p,q_e) \cdot r_e \cdot R_i(q_e,q))$
  end
end
return $R_{|Q|}(q_0', q_0')$;

Algorithm 1: State elimination algorithm

The formal algorithm is described as Algorithm 1. It takes as input a parity automaton



(a) Automaton for the language of words which contain at least one b and after every c, there is a b later on. The priorities are $\gamma(q_1) = \gamma(q_3) = 1$ and $\gamma(q_2) = 2$.

(b) The initialized transition structure defined by Q_0 and R_0.

Figure 4 Initialization step of Algorithm 1.

$\mathcal{A} = (Q, \Gamma, q_0, \delta, \gamma)$ with priorities assigned to states and returns an ∞-regular expression r with $L_\omega(r) = L(\mathcal{A})$. We explain its operations by an example.

The state elimination for parity automata is shown in Algorithm 1. It computes intermediate automata in which the transitions are labeled by ∞-regular expressions. These transitions are represented by mappings R_i that assign ∞-regular expressions to pairs of states. In the initialization, the algorithm adds a new initial state to the automaton, and R_0 maps each pair (p, q) of states to the disjunction of labels of transitions from p to q.

Consider the parity automaton shown in Figure 4a with priorities $\gamma(q_1) = \gamma(q_3) = 1$ and $\gamma(q_2) = 2$. It recognizes the language of words which contain at least one b and after every c, there is a b later on. The result of the initialization is shown in Figure 4b, where transitions labeled $\emptyset$ are omitted. The new initial state is called q_0 in the picture.

After this initialization, all states except for q_0 (q_0' in the algorithm) are eliminated in order of ascending priority. Here our algorithm differs from the original one for NFAs, for which the order of elimination is arbitrary. The resulting expression is then the label on the remaining loop on the new initial state.

The first step of this elimination process can be found in Figure 5, in which q_3 has been eliminated. As q_3 had odd priority, the $*$-operator was used for the iteration of $R_0(q_3, q_3) = a + c$. If the priority was even, all occurrences of $(a + c)^*$ would have to be replaced by $(a + c)^\infty$ instead. This is the second difference of our algorithm from the known one. The new function of expressions is then called R_1. A specific example is $R_1(q_1, q_2) = R_0(q_1, q_2) + R_0(q_1, q_3) \cdot R_0(q_3, q_3)^* \cdot R_0(q_3, q_2) = b + \emptyset \cdot (a + c)^* \cdot b$. Note here that unlike state elimination of NFAs, it is important to consider $\emptyset$-transitions in the calculation because $r^\infty\emptyset$ is the set of infinite words defined by r^∞, and thus $r^\infty\emptyset \neq \emptyset$.

The algorithm would continue by deleting q_1 followed by q_2 to create R_2 and R_3 respectively. The result is then found in $R_3(q_0, q_0)$, the self-loop of the new initial state.

The state elimination algorithm allows us to establish a similar relation between operator nesting and number of priorities like the Glushkov construction did. Concerning the complexity, the expression can be exponential in the size of the automaton, just like in the classical counterpart of the algorithm.

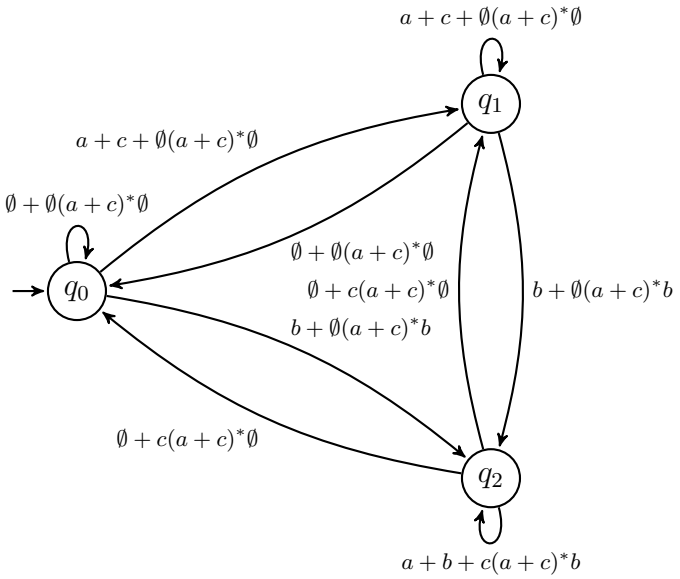


Figure 5 The transition structure defined by Q_1 and R_1, after the elimination of the first state q_3.

▶ **Theorem 11.** *Let $\mathcal{A} = (Q, \Gamma, q_0, \delta, \gamma)$ be a parity automaton with $\gamma(Q) = \{0, \dots, n\}$ or $\gamma(Q) = \{1, \dots, n\}$ and let r be the result of the state elimination algorithm with input $\mathcal{A}$.*

- *$\mathcal{A}$ accepts exactly $L_\omega(r)$*
- *$rk(r)$ is at most n*
- *$|r| \in \mathcal{O}(4^{|\Gamma| \cdot |Q|})$*

Proof. The statement about the size of r can be seen directly from the definition. In the first iteration, all edges are marked by regular expressions of size at most Γ. In every step, each expression is replaced by the concatenation of four expressions from the previous round, giving the bound on the size.

For the correctness proof one shows that after each step the automata accept the same language (with an appropriate definition of the semantics for parity automata with ∞-regular expressions on transitions). Details are available in [17]. ◀

5 Deterministic Regular Expressions

Deterministic regular expressions for finite words play a role in specification languages for XML documents. They have been studied in detail in [2] (where they are called one-unambiguous expressions). We extend the definition to ∞-regular expressions and state some basic properties for this class of expressions.

▶ **Definition 12** (Deterministic expressions)**.** Let r be an ∞-regular expression over Γ and let $\sharp(r)$ be a marking (see Definition 7) over an alphabet $\Gamma' \subset \Gamma \times \mathbb{N}$. We call r *deterministic* if it satisfies the following property: Let $u \in (\Gamma')^*$, $v, w \in (\Gamma')^\infty$, and $x = a_n, y = b_m \in \Gamma'$, such that $uxv, uyw \in L(\sharp(r))$. If $a = b$, then $n = m$.

A more natural description of this subclass defines it as those expressions, such that after every occurrence of some symbol a in the expression r and every $b \in \Gamma$, there is at most one occurrence of b in r which can succeed this occurrence of a.

For example, let $r_1 = aa^*$ and $r_2 = a^*a$. Possible markings are $\sharp(r_1) = a_1a_2^*$ and $\sharp(r_2) = a_1^*a_2$. That shows that r_1 is deterministic, but r_2 is not. For every word $u \in \{a\}^*$, it is clear which letter in u is mapped to which occurrence of a in r_1. However, that is not the case for r_2, as $u = \varepsilon$ is both a prefix of a_2 and a_1a_2. In other words, the first a of an input string could be mapped to the first or the second occurrence of a in r_2.

It is notable that these expressions are strictly less powerful than the class of all ∞-regular expressions, and they are also distinct from the deterministic Büchi-definable languages, which define a proper subclass of the ω-regular languages (see [15]). The proof is just a simple extension of a similar result for finite words: The language $(a+b)^*b(a+b)$ cannot be described by a deterministic $*$-regular expression. This is stated in [2], and can be verified using their decision procedure for definability by deterministic $*$-regular expressions.

▶ **Proposition 13.** *The language $(a+b)^*b(a+b)c^\omega$ can be recognized by a deterministic Büchi automaton but not by a deterministic ∞-regular expression.*

Proof. Assume there is a deterministic expression r for this language. We can replace every occurrence of c with ε to obtain a deterministic $*$-regular expression r' for $(a+b)^*b(a+b)$, which is not possible, as shown in [2]. The determinism of r' easily follows from the fact that the c only appear at the end of words. It is easy to specify a deterministic Büchi automaton for the language. ◀

Furthermore, a direct consequence of the definition of deterministic expressions is the following relation to the Glushkov automaton.

▶ **Proposition 14.** *Let r be an ∞-regular expression. r is deterministic iff the Glushkov automaton $\mathcal{G}_r$ is deterministic.*

Proof. As the transition structure is the same as for $*$-regular expressions, the same proof from [2] is valid here. ◀

There is however no such relation between the deterministic structure of a parity automaton and the resulting expression from the state elimination algorithm. In fact, the state elimination algorithm will produce a non-deterministic expression for almost all input automata.

In the context of deterministic ∞-regular expressions, the hierarchy $\mathcal{H}$ as defined in Section 2 becomes more interesting, as shown by the following results.

▶ **Proposition 15.** *For every $n > 0$, there is an alphabet Γ and deterministic ∞-regular expressions $r \in \Sigma_n^\Gamma$ and $s \in \Pi_n^\Gamma$ such that there are no deterministic ∞-regular expressions for $L(r)$ or $L(s)$ in $\Sigma_{n-1}^\Gamma \cup \Pi_{n-1}^\Gamma$.*

Proof. For all $0 \leq i < j$ we define $\Gamma_{i,j} = \{i, \dots, j\} \subset \mathbb{N}$ and

$$L_{i,j} = \{\alpha \in \Gamma_{i,j}^\omega \mid \text{The highest number occurring infinitely often in } \alpha \text{ is even}\}.$$

Let $r_{i,i} = \begin{cases} i^\infty & \text{if } i \text{ is even} \\ i^* & \text{if } i \text{ is odd} \end{cases}$ and $r_{i,j} = \begin{cases} (r_{i,j-1} \cdot j)^\infty & \text{if } j \text{ is even} \\ (r_{i,j-1} \cdot j)^* & \text{if } j \text{ is odd} \end{cases}$.

These regular expressions contain every symbol only once and are therefore deterministic. Their subset of ω-words also describes the corresponding language $L_{i,j}$. We use the well-known result that there is no deterministic parity automaton $\mathcal{A}$ such that $\mathcal{A}$ uses at most $j-i$ many priorities and $L(\mathcal{A}) = L_{i,j}$. A proof of this statement is given in [17].

For the claim stated in the proposition itself, let $n > 0$. We use $\Gamma = \{0, \ldots, n+1\}$, $r = r_{1,n+1}$, and $s = r_{0,n}$. Assume there is a deterministic ∞-regular expression $t \in \Sigma^{\Gamma}_{n-1} \cup \Pi^{\Gamma}_{n-1}$ with $L_\omega(r) = L_\omega(t)$. (Argumentation for s is analogous.) Then $\mathrm{rk}(t) \leq n-1$ by definition of the rank function, so the Glushkov construction finds a deterministic parity automaton for $L(r)$ which uses only priorities from 0 to $n-1$, i.e. at most n many priorities. This is a contradiction as $L(r) = L_{1,n+1}$ which we established not to be recognizable by any deterministic parity automaton with at most n priorities. ◀

As a consequence of the proof of Proposition 15 we obtain that in particular the languages of deterministic ∞-regular expressions are not captured by deterministic Büchi automata (more generally by deterministic parity automata with a fixed number of priorities). The language $L_{0,1} = \{\alpha \in \{0,1\}^\omega \mid \alpha \text{ contains only finitely many ones}\}$ is such an example.

▶ **Corollary 16.** *There are ω-languages which can be described by a deterministic ∞-regular expression but cannot be recognized by a deterministic Büchi automaton.*

The result gives more meaning to the hierarchy $\mathcal{H}$. While the class of all ∞-regular expressions breaks down to the set Π_2, the restriction of the hierarchy to deterministic expressions is strict.

6 Conclusion

We have given a simple inductive definition of the class of ∞-regular expressions that can be used to define languages of infinite words (or mixed languages of finite and infinite words). We adapted two classical algorithms for the transformation between automata and expressions such that the nesting of the two looping operators in the expressions corresponds to the priorities of an equivalent parity automaton. We have also shown that the hierarchy obtained from the nesting of the two operators, while collapsing to the second level in the general case, becomes strict for deterministic expressions.

Concerning deterministic expressions, there are some interesting questions for future research. For finite words it is decidable whether a given regular language can be defined by a deterministic expression [2]. Since the decision procedure uses the existence of a unique minimal DFA for each regular language, the methods cannot be adapted directly, and it is an open question whether the decidability result carries over to infinite words. The strictness of the deterministic hierarchy of operator nestings also naturally induces the problem of deciding whether a given deterministic ∞-regular expression has an equivalent one on a given level. For deterministic parity automata the corresponding question is decidable [4] but the exact relation between the hierarchy for deterministic ∞-regular expression and deterministic parity automata needs to be understood in more detail.

References

1 Anne Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 120(2):197–213, 1993. doi:10.1016/0304-3975(93)90287-4.

2 Anne Brüggemann-Klein and Derick Wood. One-unambiguous regular languages. *Inf. Comput.*, 140(2):229–253, 1998. doi:10.1006/inco.1997.2688.

3 Janusz A Brzozowski and Edward J McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, 12(2):67–76, 1963.

4 Olivier Carton and Ramón Maceiras. Computing the Rabin index of a parity automaton. *RAIRO Theoretical Informatics and Applications*, 33(6):495–506, 1999.

5 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

6 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

7 John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and their Relation to Automata*. Addison-Wesley, 1969.

8 Bakhadyr Khoussainov and Anil Nerode. *Automata theory and its applications*, volume 21 of *Progress in Computer Science and Applied Logic*. Birkhäuser, 2001.

9 Orna Kupferman and Moshe Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*, pages 531–542. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.66.

10 Martin Leucker and César Sánchez. Regular linear temporal logic. In *Theoretical Aspects of Computing – ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2007. doi:10.1007/978-3-540-75292-9_20.

11 Dominique Perrin and Jean-Éric Pin. Semigroups and automata on infinite words. In *NATO Advanced Study Institute Semigroups, Formal Languages and Groups*. Kluwer academic publishers, 1995.

12 Dominique Perrin and Jean-Éric Pin. *Infinite words : automata, semigroups, logic and games*. Pure and applied mathematics. Academic, London, San Diego (Calif.), 2004. URL: http://opac.inria.fr/record=b1100620.

13 IEEE Standard for Property Specification Language (PSL) IEEE Std 1850-2005 (2005).

14 Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

15 Wolfgang Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 133–192. Elsevier Science Publishers, 1990.

16 Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997. URL: http://dl.acm.org/citation.cfm?id=267871.267878.

17 Andreas Tollkötter. Transformations between regular expressions and ω-automata. Bachelor Thesis, RWTH Aachen, Germany, 2015. Available on https://www.lii.rwth-aachen.de/images/Mitarbeiter/pub/loeding/tollkoetter15-bsc.pdf.