

International Journal of Foundations of Computer Science
© World Scientific Publishing Company

Balanced-by-Construction Regular and ω -Regular Languages

Luc Edixhoven

*Department of Computer Science
Open University, The Netherlands
Centrum Wiskunde & Informatica (CWI)
The Netherlands
led@ou.nl*

Sung-Shik Jongmans

*Department of Computer Science
Open University, The Netherlands
Centrum Wiskunde & Informatica (CWI)
The Netherlands
ssj@ou.nl*

Received (Day Month Year)

Accepted (Day Month Year)

Communicated by (xxxxxxxxxx)

Paren_n is the typical generalization of the Dyck language to multiple types of parentheses. We generalize its notion of balancedness to allow parentheses of different types to freely commute. We show that balanced regular and ω -regular languages can be characterized by syntactic constraints on regular and ω -regular expressions and, using the shuffle on trajectories operator, we define grammars for balanced-by-construction expressions with which one can express every balanced regular and ω -regular language.

Keywords: Dyck language; shuffle on trajectories; regular languages.

Author accepted manuscript of an article published in the International Journal of Foundations of Computer Science, <https://dx.doi.org/10.1142/S0129054122440026>.
©World Scientific Publishing Company

1. Introduction

The Dyck language of balanced parentheses is a textbook example of a context-free language. Its typical generalization to multiple types of parentheses, Paren_n , is central in characterizing the class of context-free languages, as shown by the Chomsky-Schützenberger theorem [1]. Many other generalizations of the Dyck language have been studied over the years. The *unrestricted* Dyck language requires only that the number of left parentheses equals the number of right parentheses,

with no restriction on their ordering. Prodinger [15] generalizes this unrestricted Dyck language by considering factorizations with arbitrary words instead of with single parentheses. Labelle and Yeh [7] allow a larger alphabet where every alphabet symbol is associated with some rational number. A word is then balanced if the value of its letters sums to 0 and if it has no prefix with a negative value. The Dyck language is then the special case with a binary alphabet, where the opening bracket has value 1 and the closing bracket value -1 . This generalization is further studied by Duchon [3]. Moortgat [12] also considers a larger alphabet, in which the symbols are ordered. A word is balanced if every prefix of it contains symbol σ_i at least as many times as symbol σ_{i+1} . Finally, Liebehenschel [8] considers a generalization with multiple types of parentheses where the parentheses are not paired by type but by some similarity relation. Depending on this relation some type of left parenthesis may match multiple types of right parentheses and vice-versa.

In this paper we consider a generalization with multiple types of parentheses. The parentheses are paired by type and for each type a balanced word must contain the same number of opening and closing parentheses, with no prefix containing more closing than opening parentheses. The difference with Paren_n is that, while Paren_n requires parentheses of different types to be properly nested, we impose no such restriction: parentheses of different types may freely commute. For example, Paren_n allows $[_1 [_2]_2]_1$ but rejects $[_1 [_2]_1]_2$. We consider both to be balanced.

This notion of balancedness is of particular interest in the context of distributed computing, where different components communicate by exchanging messages: if we assign a unique type of parentheses to every communication channel between two participants and interpret opening (resp. closing) parentheses as sending (resp. receiving) messages, then balancedness characterizes precisely all sequences of communication with no lost or orphan messages. Specifically, we are interested in specifying languages that are balanced by construction, which correspond to communication protocols that are free of lost messages and orphan messages. More precisely, we aim to answer the question: can we define balanced atoms and a set of balancedness-preserving operators with which one can express all balanced languages?

Our main result is that we answer this question positively for the classes of regular and ω -regular languages. Our contributions are as follows:

- In Section 2 we show how balancedness of regular languages corresponds to syntactic properties of finite automata and regular expressions.
- In Section 3 we show that, by using a parameterized shuffle operator, we can define an infinite grammar of balanced-by-construction expressions with which one can express all balanced regular languages.
- In Section 4 we extend these results to ω -regular languages, ω -automata and ω -regular expressions.
- In Section 5 we discuss the questions of whether an infinite grammar of balanced-by-construction expressions exists for balanced context-free languages, and whether a finite grammar of balanced-by-construction expres-

sions exists for regular languages. Both of these questions appear not to have a straightforward answer.

Detailed proofs appear in a technical report [5].

This paper is an extended version of the paper presented at DLT 2021 [4]. Most notably, we have made an effort to increase the clarity and accessibility of the proofs and the general line of thought throughout the paper. We also discuss a couple of topics for future work, which did not appear in the original paper. In particular:

- We use significantly more examples throughout the paper.
- Sections 2 and 4 now use automata as a didactic stepping stone from balanced languages to balanced expressions.
- The proofs in Sections 3 and 4 have been made more accessible by adding an analogy using jigsaw pieces.
- Section 5 is new altogether.

Notation $\mathbb{N} = \{1, 2, \dots\}$, $\mathbb{N}_0 = \{0, 1, \dots\}$, \mathbb{Z} is the set of integers and $\aleph_0 = |\mathbb{N}|$. Let Σ_n be the alphabet $\{[1,]_1, \dots, [n,]_n\}$. Its size is typically either clear from the context or irrelevant; in both cases we omit the subscript. We write λ for the empty word. We write Σ^* for the set of finite words over Σ . We write Σ^ω for the set of infinite words $\{w \mid w : \mathbb{N} \rightarrow \Sigma\}$ over Σ . We write $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. We write $w(i)$ to refer to the symbol at position i in w . We write $w(i, \dots, j)$ for the substring of w beginning at position i and ending at position j . Let $v, w \in \Sigma^\infty$. Then v is a prefix of w , denoted $v \preceq w$, if $v = w$ or if there exists $v' \in \Sigma^\infty$ such that $vv' = w$. We write $|w|, |w|_\sigma \in \mathbb{N}_0 \cup \{\aleph_0\}$ respectively for the length of w and for the number of occurrences of symbol σ in w . Let \mathbb{E} be the set of all regular expressions over $\bigcup_{n \geq 1} \Sigma_n$. We write $L(e)$ (resp. $L(M)$) for the language of a regular expression e (resp. an automaton M). For $e_1, e_2 \in \mathbb{E}$, we write $e_1 \equiv e_2$ iff $L(e_1) = L(e_2)$.

2. Balanced regular languages

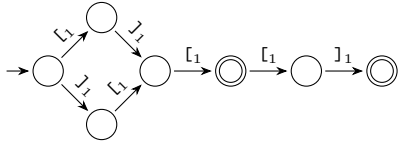
In this section, we formally define our notion of balancedness and characterize balanced regular languages in terms of finite automata and regular expressions.

Balancedness

A word $w \in \Sigma^*$ is *i-balanced* if $|w|_{[i]} = |w|_{]i}$ and if, for all prefixes v of w , $|v|_{[i]} \geq |v|_{]i}$. It is *balanced* if it is *i-balanced* for all i . We extend this terminology to languages, automata and expressions in the expected way: a language is (*i*-)balanced if all of its words are; an automaton or expression is (*i*-)balanced if its language is.

4 *L. Edixhoven and S.-S. Jongmans*

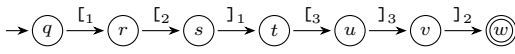
Language: $\{[_1]_1 [_1]_1 [_1]_1, [_1]_1 [_1]_1,]_1 [_1]_1 [_1]_1,]_1 [_1]_1\}$

Finite automaton:  no certificate

Regular expression: $e = ([_1]_1 +]_1 [_1]_1)_1 ([_1]_1 + \lambda) \quad \nabla(e, 1) = 1 \quad \nabla^{\min}(e, 1) = -1$

(a) Unbalanced.

Language: $\{[_1 [_2]_1 [_3]_3]_2\}$

Finite automaton: 

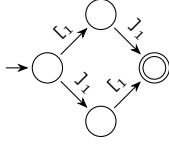
$\nabla(q, 1) = 0 \quad \nabla(r, 1) = 1 \quad \nabla(s, 1) = 1 \quad \nabla(t, 1) = 0$
 $\nabla(q, 2) = 0 \quad \nabla(r, 2) = 0 \quad \nabla(s, 2) = 1 \quad \nabla(t, 2) = 1 \quad \dots$
 $\nabla(q, 3) = 0 \quad \nabla(r, 3) = 0 \quad \nabla(s, 3) = 0 \quad \nabla(t, 3) = 0$

Regular expression: $e = [_1 [_2]_1 [_3]_3]_2$

$\nabla(e, 1) = \nabla(e, 2) = \nabla(e, 3) = 0$
 $\nabla^{\min}(e, 1) = \nabla^{\min}(e, 2) = \nabla^{\min}(e, 3) = 0$

(b) Balanced.

Language: $\{[_1]_1,]_1 [_1]_1\}$

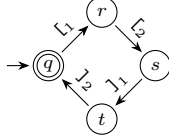
Finite automaton:  no certificate

Regular expression: $e = [_1]_1 +]_1 [_1]_1$

$\nabla(e, 1) = 0$
 $\nabla^{\min}(e, 1) = -1$

(c) Unbalanced.

Language: $\{[_1 [_2]_1]_2, [_1 [_2]_1]_2 [_1]_2, \dots\}$

Finite automaton: 

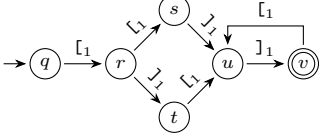
$\nabla(q, 1) = \nabla(t, 1) = 0$
 $\nabla(r, 1) = \nabla(s, 1) = 1$
 $\nabla(q, 2) = \nabla(r, 2) = 0$
 $\nabla(s, 2) = \nabla(t, 2) = 1$

Regular expression: $e = ([_1 [_2]_1]_2)^*$

$\nabla(e, 1) = \nabla(e, 2) = 0$
 $\nabla^{\min}(e, 1) = 0$
 $\nabla^{\min}(e, 2) = 0$

(d) Balanced.

Language: $\{[_1 [_1]_1]_1, [_1]_1 [_1]_1, [_1]_1 [_1]_1 [_1]_1, [_1]_1 [_1]_1 [_1]_1, \dots\}$

Finite automaton: 

$\nabla(q, 1) = \nabla(t, 1) = \nabla(v, 1) = 0$
 $\nabla(r, 1) = \nabla(u, 1) = 1$
 $\nabla(s, 1) = 2$

Regular expression: $e = [_1 ([_1]_1 +]_1 [_1]_1) (]_1 [_1]_1)^*]_1 \quad \nabla(e, 1) = 0 \quad \nabla^{\min}(e, 1) = 0$

(e) Balanced.

Fig. 1: Examples of (un)balancedness.

Example 1. Figure 1 shows examples of balanced and unbalanced languages. The finite automata and regular expressions in the figure can be ignored for now; we discuss them shortly.

The languages in Figures 1b, 1d, and 1e are balanced because, informally, in every word of those languages, every $]_i$ is preceded by a corresponding $[_i$ for every $i \in \{1, 2, 3\}$. In contrast, the language in Figure 1a is unbalanced because each of its words w violates requirement $|w|_{[_1} = |w|_{]_1}$, while the language in Figure 1c is unbalanced because word $]_1[_1$ violates requirement $|v|_{[_1} \geq |v|_{]_1}$ for prefix $v =]_1$.

Finite automata

For a (nondeterministic) finite automaton to be balanced, all words leading to a final state must be balanced: for every i , they must contain equally many opening and closing i -parentheses and the number of closing i -parentheses may never exceed the number of opening i -parentheses in any prefix. It follows that, for any state with a $]_i$ -transition to a final state, all words leading to this state must contain exactly one unmatched opening i -parenthesis—along with the previous condition on prefixes. Following this line of thought, a similar condition must hold for every state in the automaton^a: for every state and for every i , there must exist some $n \in \mathbb{N}_0$ such that $|w|_{[_i} - |w|_{]_i} = n$ for every word w leading to that state. We will call this n a state's i -balance (i.e., the number of unmatched opening i -parentheses), denoted $\nabla(q, i)$ for a state q . Additionally, the i -balances of the automaton's states must be consistent with its transitions: if $\nabla(p, 1) = 1$ and $p \xrightarrow{[_1} q$ then $\nabla(q, 1) = 2$ and $\nabla(q, i) = \nabla(p, i)$ for all $i \neq 1$.

Formally, we can prove the following theorem:

Theorem 2. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton. Then M is balanced iff there exists a certificate $\nabla : Q \times \mathbb{N} \mapsto \mathbb{Z}$ such that, for all i :

- $\nabla(q, i) \geq 0$ for all q ;
- $\nabla(q, i) = 0$ for $q = q_0$ and for all $q \in F$; and
- if $(p, \sigma, q) \in \delta$ then $\nabla(q, i) = \nabla(p, i) + \nabla(\sigma, i)$, where $\nabla(\sigma, i) = 1$ if $\sigma = [_i$, $\nabla(\sigma, i) = -1$ if $\sigma =]_i$ and $\nabla(\sigma, i) = 0$ otherwise.

Example 3. Figure 1 shows examples of balanced and unbalanced finite automata. The regular expressions in the figure can be ignored for now; we discuss them shortly.

For the automata in Figures 1b, 1d, and 1e, a certificate exists, so they are balanced. In contrast, for the automata in Figures 1a and 1c, a certificate does not exist because it is impossible to assign a non-negative number to the bottom state of the diamond.

^aTechnically such a condition only needs to hold for every state from which a final state can be reached. However, we may assume without loss of generality that a finite automaton contains no states from which no final state can be reached — unless its language is empty, in which case we may assume that it has exactly one state and no transitions.

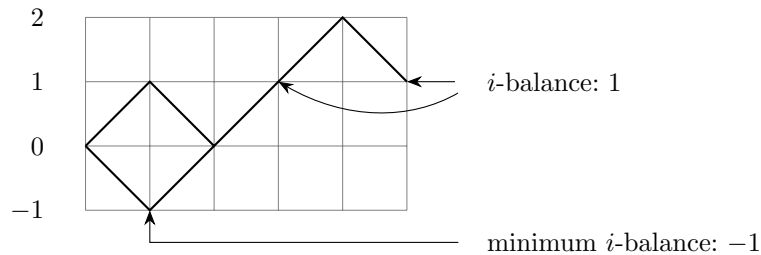


Fig. 2: The i -balance and minimum i -balance of $([i]_i +]_i [i]) [i] ([i]_i + \lambda)$. The i -balance points at the final height of the four words in the expression's language, which is the same in every case. The minimum i -balance points at the lowest height encountered by any word at any point.

Regular expressions

We can apply the same notion of i -balance to regular expressions: to every i -balanced regular expression e , we assign a value $\nabla(e, i)$, which we show to correspond to the number n of unmatched opening i -parentheses in every word in its language. In general, for an arbitrary regular expression (e.g., $\lambda + [i]$ and $[i]^*$), n does not necessarily exist. In such cases we leave $\nabla(e, i)$ undefined. Shortly, we show that $\nabla(e, i)$ is defined for all e for which n exists.

Besides keeping track of the number of unmatched opening i -parentheses, additionally, we need to differentiate between, for example, $[i]_i$ and $]_i [i]$. They have the same i -balance but the former is balanced while the latter is not. To do this we assign a second value which we call the *minimum i -balance*, denoted $\nabla^{\min}(e, i)$, which we show to correspond to the smallest i -balance among every prefix of every word in its language. Both are illustrated with a simple example in Figure 2. An expression is balanced if its i -balance and minimum i -balance equal 0 for every i .

Example 4. *Figure 1 shows examples of balanced and unbalanced regular expressions, including their balances and minimum balances.*

Formally, we define partial functions $\nabla, \nabla^{\min} : \mathbb{E} \times \mathbb{N} \mapsto \mathbb{Z}$ as in Figure 3. We show in Lemma 5 that ∇ and ∇^{\min} have the intended properties we described, and in Lemma 6 that they are defined when they should be.

As stated before, these functions are partial. In particular, $\nabla(e_1 + e_2, i)$ is defined only if $\nabla(e_1, i) = \nabla(e_2, i)$, while $\nabla(e^*, i)$ is defined only if $\nabla(e, i) = 0$. The definition of $\nabla^{\min}(e_1 \cdot e_2, i)$ relies on ∇ and may thus be undefined as well. The empty language \emptyset is a special case: we choose to leave $\nabla(\emptyset, i)$ and $\nabla^{\min}(\emptyset, i)$ undefined. Using standard algebraic rules, we can rewrite any regular expression representing a non-empty language into an equivalent expression that does not contain \emptyset . To avoid overcomplicating our definitions and proofs, we assume for any regular expression e that e does not contain \emptyset , unless $e = \emptyset$.

$$\nabla(\llbracket_i, j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \quad \begin{array}{l} \nabla(\lambda, i) = 0 \\ \nabla(e_1 \cdot e_2, i) = \nabla(e_1, i) + \nabla(e_2, i) \\ \nabla(e_1 + e_2, i) = \nabla(e_1, i) \quad \text{if } \nabla(e_1, i) = \nabla(e_2, i) \\ \nabla(e^*, i) = 0 \quad \text{if } \nabla(e, i) = 0 \end{array}$$

(a) i -balance.

$$\begin{array}{l} \nabla^{\min}(\llbracket_i, j) = 0 \\ \nabla^{\min}(\lrcorner_i, i) = -1 \\ \nabla^{\min}(\lrcorner_i, j) = 0 \text{ if } i \neq j \\ \nabla^{\min}(\lambda, i) = 0 \\ \nabla^{\min}(e_1 \cdot e_2, i) = \min(\nabla^{\min}(e_1, i), \nabla(e_1, i) + \nabla^{\min}(e_2, i)) \\ \nabla^{\min}(e_1 + e_2, i) = \min(\nabla^{\min}(e_1, i), \nabla^{\min}(e_2, i)) \\ \nabla^{\min}(e^*, i) = \nabla^{\min}(e, i) \end{array}$$

(b) Minimum i -balance.Fig. 3: The i -balance and minimum i -balance of regular expressions.

Let $e \in \mathbb{E}$. If $\nabla(e, i)$ and $\nabla^{\min}(e, i)$ are defined, then:

- (i) $|w|_{\llbracket_i} - |w|_{\lrcorner_i} = \nabla(e, i)$ for every $w \in L(e)$;
- (ii) $|v|_{\llbracket_i} - |v|_{\lrcorner_i} \geq \nabla^{\min}(e, i)$ for every prefix v of every $w \in L(e)$; and
- (iii) $|v|_{\llbracket_i} - |v|_{\lrcorner_i} = \nabla^{\min}(e, i)$ for some prefix v of some $w \in L(e)$.

Lemma 6. Let $e \in \mathbb{E}$. If $|v|_{\llbracket_i} - |v|_{\lrcorner_i} = |w|_{\llbracket_i} - |w|_{\lrcorner_i}$ for every $v, w \in L(e)$ and $L(e) \neq \emptyset$, then $\nabla(e, i)$ and $\nabla^{\min}(e, i)$ are defined.

The proofs are straightforward by structural induction on e . Applying them gives us the following characterization:

Theorem 7. Let $e \in \mathbb{E}$. e is balanced iff $e = \emptyset$ or $\nabla(e, i) = \nabla^{\min}(e, i) = 0$, for every i .

8 *L. Edixhoven and S.-S. Jongmans*

$$\begin{aligned}
 e &::= \emptyset \mid \lambda \mid [_1 \cdot] _1 \mid [_2 \cdot] _2 \mid \dots \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \mid \sqcup_{\theta}^1(e_1) \mid \sqcup_{\theta}^2(e_1, e_2) \mid \dots \\
 \theta &::= \emptyset \mid \lambda \mid 1 \mid 2 \mid \dots \mid \theta_1 + \theta_2 \mid \theta_1 \cdot \theta_1 \mid \theta^*
 \end{aligned}$$

Fig. 4: A grammar \mathbb{E}^{\sqcup} for expressing balanced regular languages.

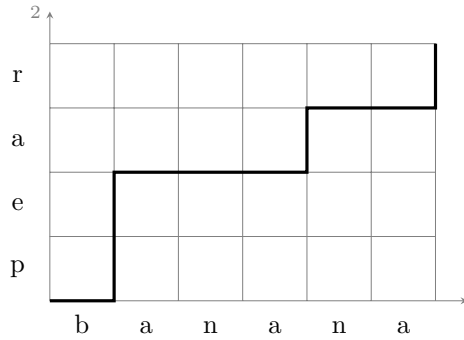


Fig. 5: The shuffle of ‘banana’ and ‘pear’ over a trajectory 1221112112: ‘bpeanaanar’.

3. Balanced-by-construction regular languages

The main contribution of this section is a grammar of balanced-by-construction expressions, \mathbb{E}^{\sqcup} in Figure 4. We show that it can express all and only all balanced regular languages. It uses regular expressions as a basis and differs in two ways:

- Parentheses can syntactically occur only in ordered pairs instead of separately, so the atoms are all balanced.
- We add a family of operators $\sqcup_{\theta}^n(e_1, \dots, e_n)$, called *shuffle on trajectories*, in order to interleave words of subexpressions.

The shuffle on trajectories operator is a powerful variation of the traditional shuffle operator, which adds a control trajectory (or a set thereof) to restrict the permitted orders of interleaving. This allows for fine-grained control over orderings when shuffling words or languages. The binary operator was defined—and its properties thoroughly studied—by Mateescu et al. [10]; we use the multiary variant, which was introduced slightly later [11].

When defined on words, the shuffle on trajectories takes n words and a *trajectory*, which is a word over the alphabet $\{1, \dots, n\}$. This trajectory specifies the exact order of interleaving of the shuffled words: in Figure 5 the trajectory 1221112112 specifies that the result should first take a symbol from the first word, then from the second, then again from the second and so on.

Formally, let $w_1, \dots, w_n \in \Sigma^*$ and let $t \in \{1, \dots, n\}^*$. Then:

$$\sqcup_t^n(w_1, \dots, w_n) = \begin{cases} \sigma \sqcup_{t'}^n(w_1, \dots, w'_i, \dots, w_n) & \text{if } t = it' \wedge w_i = \sigma w'_i, \\ \lambda & \text{if } t = w_1 = \dots = w_n = \lambda. \end{cases}$$

We note that $\sqcup_t^n(w_1, \dots, w_n)$ is only defined if the number of occurrences of i in t precisely matches the length of w_i , i.e., if $|t|_i = |w_i|$, for every i . If $|t|_i = |w_i|$, we say that t fits w_i .

Example 8.

- $\sqcup_{121332}([_1]_1, [_2]_2, [_3]_3) = _1[_2]_1[_3]_3_2$ since 121332 fits every word.
- $\sqcup_{121}^2([_1]_1, [_2]_2)$ is undefined since 121 does not fit $[_2]_2$.

The shuffle on trajectories operator naturally generalizes to languages and expressions: the shuffle of a number of languages on a set (i.e., language) of trajectories is defined as the set of all valid shuffles of words in the languages for which the trajectory fits all the words. The language of a shuffle of expressions is the shuffle of the corresponding languages. Formally:

$$\begin{aligned} \sqcup_T^n(L_1, \dots, L_n) &= \{\sqcup_t^n(w_1, \dots, w_n) \mid t \in T, w_1 \in L_1, \dots, w_n \in L_n\}. \\ L(\sqcup_\theta^n(e_1, \dots, e_n)) &= \sqcup_{L(\theta)}^n(L(e_1), \dots, L(e_n)). \end{aligned}$$

Example 9.

- $\sqcup_{12+21}^2([_1 + _2]_2,]_1) \equiv _1[_1] +]_1[_1$. Note that $[_2]_2$ in the first operand is never used since no trajectory fits it.
- $\sqcup_{12+22}^2([_1,]_1) \equiv _1[_1]_1$. Note that the trajectory 22 is never used since it does not fit a word in either operand.
- $\sqcup_{(12)^*}^2([_1]_1^*, [_2]_2^*) \equiv (_1[_2]_1)_2^*$.
- $\sqcup_{12+11}^2([_1, \lambda) \equiv \emptyset$ since neither trajectory fits words in both operands simultaneously.
- $\sqcup_{(12)^*}^2([_1]_1, [_2]_2[_2]^*) \equiv \emptyset$ since no trajectory fits words in both operands simultaneously (due to a parity issue).

We say that a set of trajectories T fits L_i if every $t \in T$ fits some $w_i \in L_i$ and that θ fits e_i if $L(\theta)$ fits $L(e_i)$. We note that the grammar \mathbb{E}^\sqcup allows any regular set of trajectories.

As the operator's arity is clear from its operands, we will omit it from now on.

In the remainder of this section, we show that the grammar \mathbb{E}^\sqcup can express only (soundness) and all (completeness) balanced regular languages.

Soundness

Showing that every expression in \mathbb{E}^\sqcup represents a balanced regular language is straightforward. The base cases all comply and both balanced and regular languages are closed under choice, concatenation and finite repetition. The shuffle on

$$\begin{array}{ll}
\langle \rangle_i^k = ([_i]_i)^k ([_i]_i)^* & \langle \rangle_i^\omega = ([_i]_i)^\omega \\
\langle \rangle_i^k = \langle \rangle_i^k [_i] & \langle \rangle_i^\omega = ([_i]_i)^\omega \\
\langle \rangle_i^k =]_i \langle \rangle_i^k & \langle \rangle_i^\omega = ([_i]_i)^\omega \\
\langle \rangle_i^k =]_i \langle \rangle_i^k [_i & \langle \rangle_i^\omega = ([_i]_i)^\omega \\
\langle \rangle_i^k = (\langle \rangle_i^k)^* & \langle \rangle_i^\omega = ([_i]_i)^\omega \\
\langle \rangle_i^k = (\langle \rangle_i^k)^* & \langle \rangle_i^\omega = ([_i]_i)^\omega
\end{array}$$

Fig. 6: Factors, with $i \in \mathbb{N}$ and $k \in \mathbb{N}_0$. The factors in the top row are balanced, those in the bottom row are not. We omit the superscript when it is not relevant. The ω -factors will be used in Section 4.

trajectories operator yields an interleaving of its operands: a simple inductive proof will show closure of balanced languages under the operation. Mateescu et al. show that regular languages are closed under binary shuffle on regular trajectory languages by constructing an equivalent finite automaton [10]; their construction can be generalized in a straightforward way to fit the multiary operator, which shows:

Theorem 10. $\{L(e) \mid e \in \mathbb{E}^\sqcup\} \subseteq \{L \mid L \text{ is a balanced and regular language}\}$.

Completeness

To show that \mathbb{E}^\sqcup can express any balanced regular language, we take an arbitrary balanced regular expression e and show that there exists an equivalent expression in \mathbb{E}^\sqcup . We cannot use a simple inductive proof, as not every subexpression of a balanced regular expression is necessarily balanced; $[_i$ and $]_i$ are the most obvious offenders. Instead, we show the more general result that, in fact, any regular expression whose every (minimum) i -balance is defined (but does not necessarily equal 0) can be written as the shuffle of a particular set of expressions, i.e., those in Figure 6, which we call *factors*. Some of these factors are balanced and some are not; we show that the number of unbalanced factors can be limited by the expression's i -balance and minimum i -balance. In other words: if e is balanced then the resulting shuffle expression only uses balanced factors and is thus in \mathbb{E}^\sqcup .

To simplify matters, we only consider regular expressions that do not contain $+$. After all, we can rewrite any regular expression into a disjunctive normal form $e_1 + \dots + e_n$ such that all e_i contain no $+$; since \mathbb{E}^\sqcup contains $+$, we can then ignore it. Additionally, we only consider regular expressions that do not contain \emptyset . Again, we can rewrite any regular expression representing a nonempty language into an equivalent one that does not contain \emptyset . Since \mathbb{E}^\sqcup contains \emptyset , all expressions representing the empty language are trivially covered and can thus be ignored.

For the remaining expressions we construct an equivalent expression $\sqcup_\theta(e_1, \dots, e_n)$, in such a way that the e_i are among the factors in Figure 6 and that the number of unbalanced factors depends on the original expression's (minimum) i -balances. We will use jigsaw pieces as a visual representation of the factors, where trailing opening parentheses and preceding closing parentheses are visualized with tabs and blanks, to make the construction and its proof more tangible. These are pictured in Figure 7.

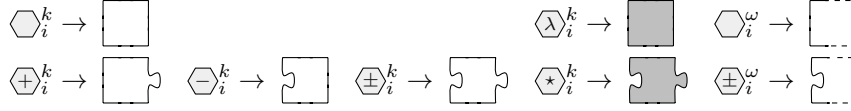


Fig. 7: Factors as jigsaw pieces. A trailing opening parenthesis is visualized with a tab on the right side. A preceding closing parenthesis is visualized with a blank on the left side. Possible emptiness (in the case of repetition) is visualized by shading the piece. Infinite repetition is visualized by not finishing the piece on the right side.

$$\begin{array}{lll}
 (\oplus_i^k, \ominus_i^\ell) \rightarrow \ominus_i^{k+\ell+1} & (\ominus_i^k, \oplus_i^\ell) \rightarrow \oplus_i^{k+\ell} & \\
 (\oplus_i^k, \oplus_i^\ell) \rightarrow \oplus_i^{k+\ell+1} & (\oplus_i^k, \ominus_i^\ell) \rightarrow \ominus_i^{k+\ell+1} & \\
 (\oplus_i^k, \star_i^\ell) \rightarrow \oplus_i^k & (\star_i^k, \ominus_i^\ell) \rightarrow \ominus_i^\ell & (\oplus_i^k, \oplus_i^\ell) \rightarrow \oplus_i^{k+\ell+1} \\
 (\oplus_i^k, \star_i^\ell) \rightarrow \oplus_i^k & (\star_i^\ell, \oplus_i^k) \rightarrow \oplus_i^k & (\star_i^k, \star_i^\ell) \rightarrow \star_i^{\min(k,\ell)} \\
 (\oplus_i^k, \oplus_i^\omega) \rightarrow \oplus_i^\omega & (\oplus_i^k, \oplus_i^\omega) \rightarrow \oplus_i^\omega & (\star_i^k, \oplus_i^\omega) \rightarrow \oplus_i^\omega
 \end{array}$$

Fig. 8: Merging common pairs of factors, with $i \in \mathbb{N}$ and $k, \ell \in \mathbb{N}_0$.

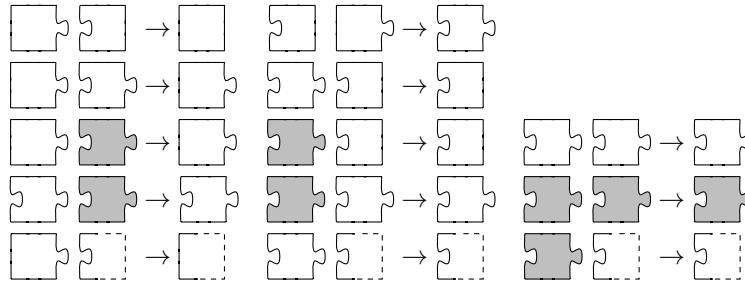


Fig. 9: Merging pairs of factors as jigsaw pieces.

Specifically, the jigsaw pieces help to understand a crucial step in our construction: we show that, under certain conditions, we can *merge* two operands of a shuffle, i.e., $\sqcup_T(L_1, \dots, L_{n-1}, L_n) = \sqcup_{T'}(L_1, \dots, L_{n-1}L_n)$ for some T' . The formal details can be found in Lemma 11, where we define T' using morphisms, similarly to previous work [10, 2]. In particular, we later use this lemma to merge the pairs of factors in Figure 8. In terms of jigsaw pieces, this merging operation may be thought of as fitting two pieces together; the aforementioned pairs of factors are shown using jigsaw pieces in Figure 9. In this visualization, two factors can be merged if their pieces fit nicely together; typically, this consists of fitting a tab with a blank, which corresponds to matching a trailing opening parenthesis with a preceding closing one. The resulting pieces have the expected shape and are shaded only if both original pieces were. One notable exception is the second pair of pieces in the top row of

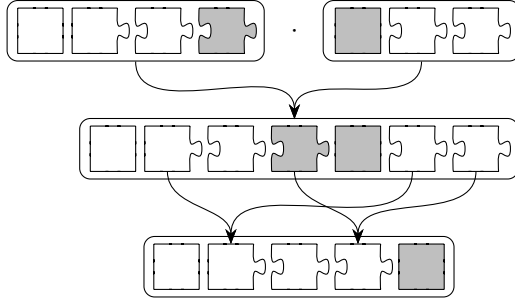


Fig. 10: The concatenation (and merging) of two shuffles of factors as groups of jigsaw pieces.

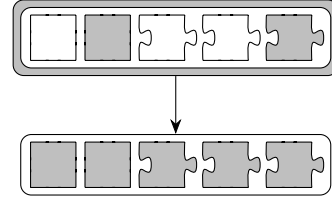


Fig. 11: The finite repetition of a shuffle of factors as a group of jigsaw pieces.

Figure 9: these two are put back-to-back. There are more pieces that can be nicely put back-to-back: for example, $\blacksquare \square \rightarrow \square$. However, since these other pairs are unnecessary for our later proof—unlike the ones in Figure 9—they are omitted.

Lemma 11 (Merge) *Let $L = \sqcup_T(L_1, \dots, L_n)$. If*

- (a) *T fits every L_i ,*
- (b) *for every $t \in T$, if $t(i) = n - 1$ and $t(j) = n$ then $i < j$, and*
- (c) *for all $v, w \in L_{n-1}L_n$, if $|v| = |w|$ then $v = w$,*

then $L = \sqcup_{T'}(L_1, \dots, L_{n-1}L_n)$ for some T' such that T' fits $L_1, \dots, L_{n-1}L_n$.

Proof. Let ψ be a homomorphism such that $\psi(n) = n - 1$ and $\psi(i) = i$ for all other i . We proceed to show that $L = \sqcup_{\psi(T)}(L_1, \dots, L_{n-1}L_n)$. Since T fits every L_i , $\psi(T)$ also fits $L_1, \dots, L_{n-1}L_n$. \square

Equipped with this merging lemma, we proceed with the construction. Recall that the result should be a single shuffle of factors, i.e., $\sqcup_\theta(e_1, \dots, e_n)$ for some factors e_1, \dots, e_n , with a minimal number of unbalanced factors. The construction is by structural induction. The base cases are straightforward: for λ , $[_i$ and $]_i$ we can use respectively $\sqcup_\lambda(\langle \diamond_i^0 \rangle)$, $\sqcup_1(\langle \oplus_i^0 \rangle)$ and $\sqcup_1(\langle \ominus_i^0 \rangle)$. In jigsaw pieces these are \blacksquare , \square and \square . Since we can ignore both \emptyset and $+$, we are left with just two inductive cases: \cdot (concatenation) and $*$ (finite repetition).

- For concatenation, we use that $\sqcup_{T_1}(L_1, \dots, L_n) \cdot \sqcup_{T_2}(L_{n+1}, \dots, L_{n+m}) = \sqcup_{T_3}(L_1, \dots, L_{n+m})$ for some T_3 . Having combined the two into a single shuffle, we then apply Lemma 11 as needed to merge suitable pairs of factors to minimize the number of unbalanced factors. This is illustrated using jigsaw pieces in Figure 10. To understand the figure, suppose that we start with two expressions: the first with balance 1 and minimum balance -2 —for the sake of simplicity, there is only one type of parentheses—and the second with balance 0 and minimum balance -2 . In the figure, these two

expressions are represented on the top row as two groups of jigsaw pieces; the number of blanks equals the minimum balance of the corresponding expression, while the number of tabs minus the number of blanks equals its balance. The first step is to combine the two groups into one. The other step is then to fit pairs of pieces from the first and the second group. Recall from Figure 3 that the balance of the resulting expression is 1 and its minimum balance is -2 ; indeed, the resulting group of pieces contains three tabs and two blanks. In other words, the resulting group of factors is minimal.

- For finite repetition, we show that, under our assumed conditions, $(\sqcup_T(L_1, \dots, L_n))^* = \sqcup_{T^*}(L_1^*, \dots, L_n^*)$. In jigsaw pieces, this simply means shading every non-shaded piece, as illustrated in Figure 11—note that $(e^*)^* \equiv e^*$. The $*$ -operator, visualized as an extra shaded container, does not change an expression's balance or minimum balance, and since the number of tabs and blanks does not change either, the resulting group of factors remains minimal.

This construction and its proof are formalized in Lemma 12.

Lemma 12 (Rewrite) *Let $\text{pos}_i(e_1, \dots, e_n)$, $\text{neg}_i(e_1, \dots, e_n)$, $\text{neut}_i(e_1, \dots, e_n)$ be the number of \oplus_i , \ominus_i and $[\oplus_i$ or $\ominus_i]$ among e_1, \dots, e_n .*

Let $e \in \mathbb{E}$ such that e contains no $+$ and that its (minimum) i -balance is defined for every i . Then there exist θ and factors e_1, \dots, e_n such that $e \equiv \sqcup_\theta(e_1, \dots, e_n)$ and, additionally,

- $\text{pos}_i(e_1, \dots, e_n) - \text{neg}_i(e_1, \dots, e_n) = \nabla(e, i)$ for every i ,
- $-\text{neg}_i(e_1, \dots, e_n) - \text{neut}_i(e_1, \dots, e_n) = \nabla^{\min}(e, i)$ for every i ,
- there are not both \oplus_i and \ominus_i among e_1, \dots, e_n for some i , and
- θ fits every e_i .

Proof. This is a proof by induction on the structure of e .

The base cases λ , $[_i$ and $]_i$ are covered by $\sqcup_\lambda^1(\ominus_i^0)$, $\sqcup_1^1(\oplus_i^0)$ and $\sqcup_1^1(\ominus_i^0)$. Since e contains no $+$, this leaves us with two inductive cases:

- Let $e = \hat{e}^*$. The induction hypothesis gives us some $\hat{e}_1, \dots, \hat{e}_n$ and $\hat{\theta}$ satisfying all conditions for \hat{e} . It should be clear that $L((\sqcup_{\hat{\theta}}(\hat{e}_1, \dots, \hat{e}_n))^*) \subseteq L((\sqcup_{\hat{\theta}^*}(\hat{e}_1^*, \dots, \hat{e}_n^*))^*) \subseteq L(\sqcup_{\hat{\theta}^*}(\hat{e}_1^*, \dots, \hat{e}_n^*))$. Since $\nabla(e, i)$ is defined for all i , $\nabla(\hat{e}, i) = 0$ for all i . It then follows from (a) and (c) that $\hat{e}_1, \dots, \hat{e}_n$ contain no \oplus_i or \ominus_i , so all \hat{e}_i^* are also factors.

To prove inclusion in the other direction, we show in two steps that $L(\sqcup_{\hat{\theta}^*}(\hat{e}_1^*, \dots, \hat{e}_n^*)) \subseteq L((\sqcup_{\hat{\theta}}(\hat{e}_1, \dots, \hat{e}_n))^*) \subseteq L((\sqcup_{\hat{\theta}^*}(\hat{e}_1, \dots, \hat{e}_n))^*)$.

The balances, minimum balances and factor counts are unchanged, so (a-c) are satisfied. Finally, since $\hat{\theta}$ fits every \hat{e}_i , $\hat{\theta}^*$ fits every \hat{e}_i^* , so (d) also holds.

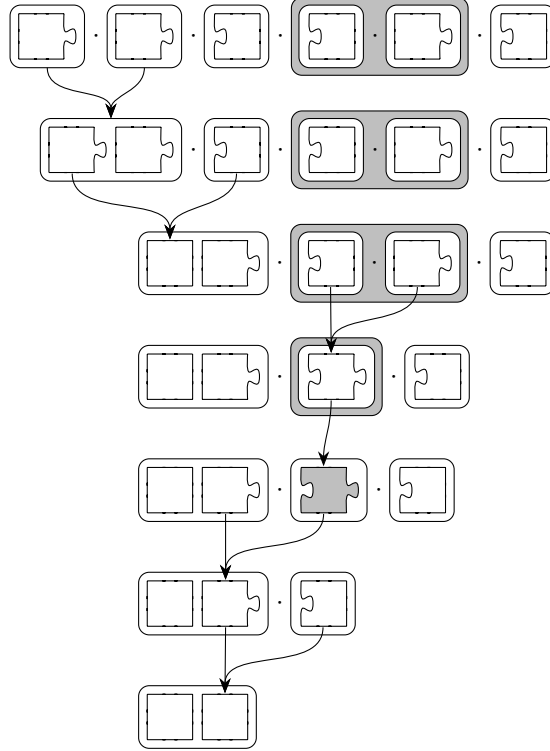


Fig. 12: Construction of Example 14 as groups of jigsaw pieces.

- Let $e = \hat{e}_1 \cdot \hat{e}_2$. The induction hypothesis gives us some $e_{1,1}, \dots, e_{1,n_1}$ and θ_1 satisfying all conditions for \hat{e}_1 , and similarly for \hat{e}_2 . Let φ be a homomorphism such that $\varphi(i) = i + n_1$. Then $e' = \sqcup_{\theta_1 \varphi(\theta_2)}(e_{1,1}, \dots, e_{1,n_1}, e_{2,1}, \dots, e_{2,n_2}) \equiv e$ and e' satisfies (d), but not necessarily (a–c). We resolve the latter by merging operands $e_{1,j}, e_{2,k}$ where applicable by Lemma 11. We merge pairs of factors from Figure 8, taking care to prioritize pairs containing both $\hat{\oplus}_i$ and $\hat{\ominus}_i$ over pairs containing only one of these, and pairs containing only one over pairs containing none. By Lemma 11, the resulting expression is equivalent to e' and satisfies (d). It also satisfies (a–c). \square

Since a balanced regular expression has an i -balance and minimum i -balance of 0 for every i (Theorem 7), the following theorem follows directly from Lemma 12.

Theorem 13. $\{L(e) \mid e \in \mathbb{E}^\omega\} \supseteq \{L \mid L \text{ is a balanced and regular language}\}$.

Example 14. Consider $e = [{}_1([{}_1]_1 +]_1[{}_1])(]_1[{}_1]^*)]_1$ (see also Figure 1e). We first rewrite e as $[{}_1[{}_1]_1(]_1[{}_1]^*)]_1 + [{}_1]_1[{}_1(]_1[{}_1]^*)]_1$. We proceed to show how to

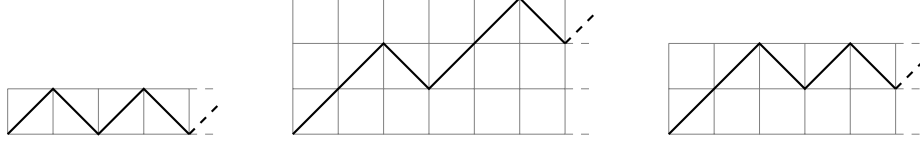


Fig. 13: From left to right: $w_1 = ([_i]_i)^\omega$, $w_2 = ([_i[_i]_i]_i)^\omega$ and $w_3 = [_i([_i]_i)]_i^\omega$.

construct an expression in \mathbb{E}^ω for the first part of the disjunction:

$$\begin{aligned}
 [_1[_1]_1]_1([_1[_1]_1]^*)]_1 &\equiv \underline{\omega_1(\langle \oplus_1^0 \rangle)} \omega_1(\langle \oplus_1^0 \rangle) \omega_1(\langle \ominus_1^0 \rangle) (\omega_1(\langle \ominus_1^0 \rangle) \omega_1(\langle \oplus_1^0 \rangle))^* \omega_1(\langle \ominus_1^0 \rangle) \\
 &\equiv \underline{\omega_{12}(\langle \oplus_1^0 \rangle, \langle \oplus_1^0 \rangle)} \omega_1(\langle \ominus_1^0 \rangle) (\omega_1(\langle \ominus_1^0 \rangle) \omega_1(\langle \oplus_1^0 \rangle))^* \omega_1(\langle \ominus_1^0 \rangle) \\
 &\equiv \omega_{121}(\langle \ominus_1^1 \rangle, \langle \oplus_1^0 \rangle) (\omega_1(\langle \ominus_1^0 \rangle) \omega_1(\langle \oplus_1^0 \rangle))^* \omega_1(\langle \ominus_1^0 \rangle) \\
 &\equiv \omega_{121}(\langle \ominus_1^1 \rangle, \langle \oplus_1^0 \rangle) (\omega_{11}(\langle \oplus_1^0 \rangle))^* \omega_1(\langle \ominus_1^0 \rangle) \\
 &\equiv \underline{\omega_{121}(\langle \ominus_1^1 \rangle, \langle \oplus_1^0 \rangle)} \omega_{(11)^*}(\langle \oplus_1^0 \rangle) \omega_1(\langle \ominus_1^0 \rangle) \\
 &\equiv \underline{\omega_{121}(22)^*}(\langle \ominus_1^1 \rangle, \langle \oplus_1^0 \rangle) \omega_1(\langle \ominus_1^0 \rangle) \\
 &\equiv \omega_{121}(22)^*2(\langle \ominus_1^1 \rangle, \langle \oplus_1^1 \rangle).
 \end{aligned}$$

Figure 12 illustrates this construction using groups of jigsaw pieces, line by line.

4. Balanced-by-construction ω -regular languages

ω -Languages are languages of infinite words. To define balancedness of infinite words and languages, consider the three words in Figure 13.

In all three cases, the number of opening parentheses equals the number of closing parentheses (since $2 \times \aleph_0 = \aleph_0$ and $\aleph_0 + 1 = \aleph_0$) and every (finite) prefix has at least as many opening as closing parentheses. However, successive prefixes of w_2 contain an ever-increasing number of unmatched opening parentheses. As stated in Section 1, our study of balanced languages stems from our interest in communication protocols. Channels in such protocols often require buffers of finite size. As such, we do not consider w_2 to be balanced as it can cause an unbounded number of messages to be in transit. At the same time, we consider w_3 to be fine: it will be perpetually at least one message ahead, but never more than two. To this end, we define a notion of boundedness and use it as an additional constraint on balancedness.

Boundedness

A (finite or infinite) word $w \in \Sigma^\infty$ is *i-bounded* by $n \in \mathbb{N}_0$ if $|v|_{[_i]} - |v|_{]_i]} \leq n$ for all finite prefixes v of w . A language is *i-bounded* by n if all of its words are. A word or language is *i-bounded* if it is *i-bounded* by n for some n . A word or language is *bounded* if it is *i-bounded* for all i . The *minimal i-bound* of a word or language is the smallest n for which it is *i-bounded*. We extend these definitions to expressions

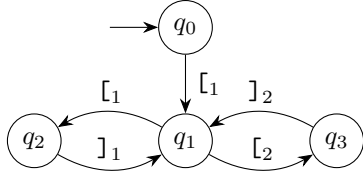


Fig. 14: A Muller automaton with acceptance condition $\{\{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\}$, accepting the (unbalanced) language $[_1([_1]_1 + [2]_2)]^\omega$.

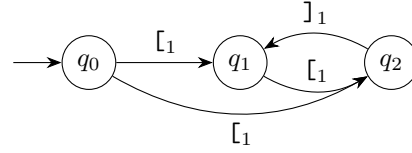


Fig. 15: A Muller automaton with acceptance condition $\{\{q_1, q_2\}\}$, accepting the (balanced) language $(\lambda + [1])([1]_1)^\omega$.

and automata in the expected way: they are (*i*-)bounded if their corresponding languages are.

We note that the boundedness of words does not necessarily imply the boundedness of a language: all of the words in $[_i^*([_i]_i)^\omega$ are bounded but the language itself is not.

Balancedness

A word $w \in \Sigma^\infty$ is *i*-balanced if $|w|_{[i]} = |w|_{]i}$, $|v|_{[i]} \geq |v|_{]i}$ for all (finite) prefixes v of w , and w is *i*-bounded. A language $L \subseteq \Sigma^\infty$ is *i*-balanced if all of its words are and if it is *i*-bounded. These are extended to balancedness and automata and expressions in the expected way: words and languages are balanced if they are *i*-balanced for all *i*, automata and expressions are (*i*-)balanced if their corresponding languages are. We note that all finite words are bounded by default, and that any regular language whose *i*-balance is defined is *i*-bounded as well. In other words: boundedness is only a restriction on infinite words and on languages containing infinite words.

4.1. Balanced ω -automata

ω -Automata are automata accepting ω -languages. As with finite automata, there are multiple classes of ω -automata. In this paper we will use Muller automata [13].

A Muller automaton differs from a finite automaton only in its acceptance condition: instead of a set of final states, a Muller automaton has a set of sets of states F , and it accepts exactly those runs in which the set of states that are visited infinitely often is member of F . Muller automata, both deterministic and non-deterministic, characterize the class of ω -regular languages.

Our characterization of balanced ω -automata follows the same approach as for balanced finite automata: we define a balance function and use this to specify balancedness conditions. We use two examples to illustrate the differences with finite automata.

- Consider the automaton in Figure 14. With acceptance condition $\{\{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\}$ it accepts the language $[_1([_1]_1 + [2]_2)]^\omega$.

The elements $\{q_1, q_2\}$ and $\{q_1, q_2, q_3\}$ both lead to the acceptance of balanced words, but $\{q_1, q_3\}$ yields unbalanced words: since q_2 is only visited finitely often, all resulting words have finite and unequal numbers of opening and closing 1-parentheses.

To classify this automaton as unbalanced, we thus require for every i and for every set of states in the acceptance condition either that there is some $[_i$ - or $]$ -transition between two states in the set (violated in Figure 14), in which case all corresponding words contain infinitely many i -parentheses, or that the i -balance of all states in the set equals 0 (also violated in Figure 14), in which case all corresponding words contain finitely many i -parentheses and equally many opening and closing ones.

- Consider the automaton in Figure 15. Its language, $(\lambda + [_1)([1]_1)^\omega$, is balanced, but no unique 1-balance can be assigned to states q_1 and q_2 : they can be either 1 and 2 if the first transition taken is that from q_0 to q_1 , or 0 and 1 if the first transition taken is that from q_0 to q_2 .

To remedy this, instead of assigning a single value as a state's i -balance, we now assign a range of values by giving an upper and lower bound on its i -balance. For the automaton in Figure 15, the lower and upper bound of the 1-balance of q_1 are respectively 0 and 1, and those of q_2 are respectively 1 and 2—those of q_0 are both 0.

Formally, and combining these changes, we can prove the following theorem:

Theorem 15. *Let $M = (Q, \Sigma, \delta, Q_0, F)$ be a Muller automaton. M is balanced iff there exist lower and upper bounds on the i -balance of every state in Q (respectively $\nabla^L(q, i)$ and $\nabla^U(q, i)$ for a state q) such that, for every i ,*

- $\nabla^L(q, i) = \nabla^U(q, i) = 0$ for every $q \in Q_0$;
- $\nabla^L(q, i) \geq 0$ for every $q \in Q$;
- $\nabla^L(q, i) \leq \nabla^L(p, i) + \nabla(\sigma, i) \leq \nabla^U(p, i) + \nabla(\sigma, i) \leq \nabla^U(q, i)$ for every $(p, \sigma, q) \in \delta$; and
- for every $\{f_1, \dots, f_\ell\} \in F$ (representing a nonempty language), either there exist some j, k such that $(\nabla^L(f_j, i), \nabla^U(f_j, i)) \neq (\nabla^L(f_k, i), \nabla^U(f_k, i))$, or $\nabla^L(f_j, i) = \nabla^U(f_j, i) = 0$ for every j .

4.2. Balanced ω -regular expressions

We use Ω for the set of all ω -regular expressions over $\bigcup_{n \geq 1} \Sigma_n$, defined as follows:

$$\frac{}{\emptyset \in \Omega} \quad \frac{e \in \mathbb{E} \quad \lambda \notin L(e)}{e^\omega \in \Omega} \quad \frac{e_1 \in \mathbb{E} \quad e_2 \in \Omega}{e_1 \cdot e_2 \in \Omega} \quad \frac{e_1, e_2 \in \Omega}{e_1 + e_2 \in \Omega} \quad (1)$$

As before, we assume without loss of generality that an ω -regular expression e does not contain \emptyset , unless $e = \emptyset$, to simplify our definitions and proofs.

Our characterization of balanced ω -regular expressions is a generalization of that of balanced regular expressions, adapted to deal with the complications noted with

18 *L. Ediahoven and S.-S. Jongmans*

$$\begin{array}{ccccc}
 \overline{\xi(\llbracket i, i \rrbracket)} & \overline{\xi(\llbracket i, i \rrbracket)} & \frac{\xi(e_1, i) \vee \xi(e_2, i)}{\xi(e_1 \cdot e_2, i)} & \frac{\xi(e_1, i) \quad \xi(e_2, i)}{\xi(e_1 + e_2, i)} & \frac{\xi(e, i)}{\xi(e^\omega, i)} \\
 & \frac{\xi(e, i)}{\xi^\omega(e^\omega, i)} & \frac{\xi^\omega(e_2, i)}{\xi^\omega(e_1 \cdot e_2, i)} & \frac{\xi^\omega(e_1, i) \quad \xi^\omega(e_2, i)}{\xi^\omega(e_1 + e_2, i)} &
 \end{array}$$

 Fig. 16: The i -occurrence of regular and ω -regular expressions.

$$\begin{array}{l}
 \nabla^\dagger(\llbracket i, j \rrbracket) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \\
 \nabla^\dagger(\llbracket i, j \rrbracket) = \begin{cases} -1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \\
 \nabla^\dagger(\lambda, i) = 0
 \end{array}
 \quad
 \begin{array}{l}
 \nabla^\dagger(e_1 \cdot e_2, i) = \begin{cases} \nabla^\dagger(e_2, i) & \text{if } \xi^\omega(e_2, i) \\ \nabla^\dagger(e_1, i) + \nabla^\dagger(e_2, i) & \text{otherwise} \end{cases} \\
 \nabla^L(e_1 + e_2, i) = \min(\nabla^L(e_1, i), \nabla^L(e_2, i)) \\
 \nabla^U(e_1 + e_2, i) = \max(\nabla^L(e_1, i), \nabla^L(e_2, i)) \\
 \nabla^\dagger(e^*, i) = 0 \quad \text{if } \nabla^\dagger(e, i) = 0 \\
 \nabla^\dagger(e^\omega, i) = 0 \quad \text{if } \nabla^\dagger(e, i) = 0
 \end{array}$$

 Fig. 17: The i -balance lower and upper bounds of regular and ω -regular expressions, where $\dagger \in \{L, U\}$.

ω -automata:

- We introduce two predicates for expressions: $\xi(e, i)$ and $\xi^\omega(e, i)$. Intuitively, $\xi(e, i)$ iff every word in $L(e)$ contains at least one i -parenthesis, while $\xi^\omega(e, i)$ iff every word in $L(e)$ contains infinitely many. The predicates are formally defined in Figure 16, while the corresponding properties are shown in Lemma 16.
- As with ω -automata, we swap single i -balances for pairs of lower and upper bounds $\nabla^L(e, i)$ and $\nabla^U(e, i)$. These bounds are formally defined in Figure 17. The changes to minimum i -balance are as expected: we add $\nabla^{\min}(e^\omega, i) = \nabla^{\min}(e, i)$ and we redefine $\nabla^{\min}(e_1 \cdot e_2, i)$ to use $\nabla^L(e_1, i)$ instead of $\nabla(e_1, i)$. The other equations remain as in Figure 3. The corresponding properties are shown in Lemmas 17 and 18. We note that, for any regular expression $e \in \mathbb{E}$, $\nabla^L(e, i) = \nabla^U(e, i) = \nabla(e, i)$.

Lemma 16. *Let $e \in \mathbb{E} \cup \Omega$. If $e \neq \emptyset$, then:*

- (i) $\xi(e, i)$ iff $|w|_{\llbracket i \rrbracket} + |w|_{\lceil i \rceil} > 0$ for every $w \in L(e)$;
- (ii) $\xi^\omega(e, i)$ iff $|w|_{\llbracket i \rrbracket} + |w|_{\lceil i \rceil} = \aleph_0$ for every $w \in L(e)$.

Lemma 17 (cf. Lemma 5) *Let $e \in \mathbb{E} \cup \Omega$. If $\nabla^L(e, i)$, $\nabla^U(e, i)$ and $\nabla^{\min}(e, i)$ are defined, then:*

- (i) For every $w \in L(e)$, $|w|_{\llbracket i \rrbracket}$ and $|w|_{\lceil i \rceil}$ are either both finite or both infinite;

- (ii) For every $w \in L(e)$, if $|w|_{\lceil_i}, |w|_{\rfloor_i}$ are finite, then $\nabla^L(e, i) \leq |w|_{\lceil_i} - |w|_{\rfloor_i} \leq \nabla^U(e, i)$;
- (iii) If $e \in \mathbb{E}$, then there exist $w_1, w_2 \in L(e)$ such that $|w_1|_{\lceil_i} - |w_1|_{\rfloor_i} = \nabla^L(e, i)$ and $|w_2|_{\lceil_i} - |w_2|_{\rfloor_i} = \nabla^U(e, i)$;
- (iv) If $\xi^\omega(e, i)$, then $\nabla^L(e, i) = \nabla^U(e, i) = 0$;
- (v) $|v|_{\lceil_i} - |v|_{\rfloor_i} \geq \nabla^{\min}(e, i)$ for every finite prefix v of every $w \in L(e)$;
- (vi) $|v|_{\lceil_i} - |v|_{\rfloor_i} = \nabla^{\min}(e, i)$ for some finite prefix v of some $w \in L(e)$;
- (vii) $L(e)$ is i -bounded.

Lemma 18 (cf. Lemma 6) *Let $e \in \mathbb{E} \cup \Omega$. If $e \neq \emptyset$, e is i -bounded, if there exists some n such that $|(v|_{\lceil_i} - |v|_{\rfloor_i}) - (|w|_{\lceil_i} - |w|_{\rfloor_i})| \leq n$ for all $v, w \in L(e)$ with finite i -parenthesis counts, then $\nabla^L(e, i)$, $\nabla^U(e, i)$ and $\nabla^{\min}(e, i)$ are defined.*

The proofs of these lemmas are straightforward by structural induction on e . Applying these lemmas gives us the following characterization:

Theorem 19. *Let $e \in \mathbb{E} \cup \Omega$. Then e is balanced iff $\nabla^L(e, i) = \nabla^U(e, i) = \nabla^{\min}(e, i) = 0$ for every i or if $e = \emptyset$.*

Example 20. *In this example, we further illustrate the role of lower and upper bounds on i -balances in the presence of ω .*

- Let $e_1 = \lceil_1 + \lceil_1 \lceil_1 \lceil_1$. We have:

$$\begin{aligned} \nabla^L(\lceil_1 + \lceil_1 \lceil_1 \lceil_1, 1) &= \min(\nabla^L(\lceil_1, 1), \nabla^L(\lceil_1 \lceil_1 \lceil_1, 1)) = \min(1, 3) = 1 \\ \nabla^U(\lceil_1 + \lceil_1 \lceil_1 \lceil_1, 1) &= \max(\nabla^U(\lceil_1, 1), \nabla^U(\lceil_1 \lceil_1 \lceil_1, 1)) = \max(1, 3) = 3 \end{aligned}$$

Thus, e_1 is unbalanced. Intuitively, the problem is that there are unmatched opening 1-parentheses.

- Let $e_2 = e_1 \cdot (\lceil_1 \lceil_1)^*$. Since $\neg \xi^\omega((\lceil_1 \lceil_1)^*, 1)$, we have:

$$\begin{aligned} \nabla^L(e_1 \cdot (\lceil_1 \lceil_1)^*, 1) &= \nabla^L(e_1, 1) + \nabla^L((\lceil_1 \lceil_1)^*, 1) = 1 + 0 = 1 \\ \nabla^U(e_1 \cdot (\lceil_1 \lceil_1)^*, 1) &= \nabla^U(e_1, 1) + \nabla^U((\lceil_1 \lceil_1)^*, 1) = 3 + 0 = 3 \end{aligned}$$

Thus, e_2 is unbalanced. Intuitively, the problem remains that there are unmatched opening 1-parentheses.

- Let $e_3 = e_1 \cdot (\lceil_1 \lceil_1)^\omega$. Since $\xi^\omega((\lceil_1 \lceil_1)^\omega, 1)$, we have:

$$\begin{aligned} \nabla^L(e_1 \cdot (\lceil_1 \lceil_1)^\omega, 1) &= \nabla^L((\lceil_1 \lceil_1)^\omega, 1) = 0 \\ \nabla^U(e_1 \cdot (\lceil_1 \lceil_1)^\omega, 1) &= \nabla^U((\lceil_1 \lceil_1)^\omega, 1) = 0 \end{aligned}$$

Thus, e_3 is balanced, even though its prefix e_1 is unbalanced. Intuitively, the solution is that even though, infinitely often, there are more opening parentheses than closing parentheses, every opening parenthesis is eventually matched and the number of unmatched opening parentheses never exceeds three. This precisely coincides with the notion of balancedness for infinite words that we adopted, as motivated at the beginning of this section.

$$\begin{aligned}
e &::= \emptyset \mid e + e \mid E \cdot e \mid E_+^\omega \mid \sqcup_{T_\omega} (C, \dots, C) && (\omega\text{-regular}) \\
E &::= \emptyset \mid \lambda \mid P \mid E + E \mid E \cdot E \mid E^* \mid \sqcup_T (E, \dots, E) && (\text{regular}) \\
E_+ &::= \emptyset \mid P \mid E_+ + E_+ \mid E \cdot E_+ \cdot E \mid \sqcup_{T_+} (E, \dots, E) && (\text{non-empty}) \\
P &::= [_ \cdot]_1 \mid [_ \cdot]_2 \mid \dots && (\text{parentheses}) \\
C &::= e \mid E && (\omega\text{-shuffle operand}) \\
T &::= \emptyset \mid \lambda \mid 1 \mid 2 \mid \dots \mid T + T \mid T \cdot T \mid T^* && (\text{trajectory}) \\
T_+ &::= \emptyset \mid 1 \mid 2 \mid \dots \mid T_+ + T_+ \mid T \cdot T_+ \cdot T && (\text{non-empty}) \\
T_\omega &::= \emptyset \mid T_\omega + T_\omega \mid T \cdot T_\omega \mid T_+^\omega && (\omega\text{-trajectory})
\end{aligned}$$

Fig. 18: A grammar Ω^\sqcup for expressing balanced regular languages.

4.3. *Balanced-by-construction ω -regular languages*

To construct balanced ω -regular expressions, we extend the grammar in Figure 4 with ω as in (1) to obtain the expression grammar Ω^\sqcup in Figure 18.

Since the inductive definition of the shuffle on trajectories operator does not support words of infinite length, we formally redefine it as follows. The definition is adapted from the definition given by Mateescu et al. for the binary case [10]. Let $w_1, \dots, w_n \in \Sigma^\infty$ and let $t \in \{1, \dots, n\}^\infty$. If t fits w_1, \dots, w_n , i.e., if $|t|_i = |w_i|$ for every i , then $\sqcup_t(w_1, \dots, w_n) = w(1)w(2)\dots w(|t|)$ if t has finite length and $w(1)w(2)\dots$ if t has infinite length, where $w(i) = w_j(k)$ for $j = t(i)$ and $k = |t(1, \dots, i)|_j$. The result is as expected. As before, this naturally extends to languages and expressions.

Soundness

Balanced languages are closed under shuffle; this follows immediately from its definition. Mateescu et al. show that ω -regular languages are closed under binary shuffle on ω -regular trajectory languages [10]. We extend their result to multiary shuffles by constructing a Muller automaton M for $\sqcup_T(L_1, \dots, L_n)$ out of a Muller automaton M_T for T and finite/Muller automata M_1, \dots, M_n for L_1, \dots, L_n , where L_1, \dots, L_n are all either regular or ω -regular.

The construction of M is analogous to the construction of a finite automaton for a shuffle of regular languages and differs only in the construction of the acceptance criterion F . To define it, let F_i be the acceptance criterion of M_i : if M_i is a finite automaton, then F_i is a set of states, and we may assume without loss of generality that no state in F_i has any outgoing transition (otherwise, we can construct an equivalent automaton which does have this property); if M_i is a Muller automaton, then F_i is a set of sets of states. We also assume, without loss of generality, that every i such that L_i is ω -regular occurs infinitely often in every trajectory in T .

We now define F as the cross product of all the F_i : F is the set of sets of states such that, if L_i is ω -regular, then the projection of these states on i is an element of F_i , and if L_i is regular, then the projection of these states on i is a single state in F_i . Formally, let $\varphi_i((q_t, q_1, \dots, q_n)) = q_i$ and $\varphi_i(S) = \{\varphi_i(q) \mid q \in S\}$ be the projection of a state or set of states on i . Then $F = \{S \mid S \subseteq Q \wedge \forall i \in [1, n] : (\varphi_i(S) \in F_i \vee (\varphi_i(S) \subseteq F_i \wedge |\varphi_i(S)| = 1))\}$, where Q is the set of states of M . The automaton for T forces that every Muller automaton for some L_i takes infinitely many steps. By our assumption that the final states of finite automata have no outgoing transitions, all finite automata only take a finite number of steps. It follows that our constructed Muller automaton accepts the language of $\sqcup_T(L_1, \dots, L_n)$, which then must be ω -regular. In other words:

Theorem 21. $\{L(e) \mid e \in \Omega^\sqcup\} \subseteq \{L \mid L \text{ is a balanced } \omega\text{-regular language}\}$.

Completeness

Our approach to showing that every balanced ω -regular expression has an equivalent expression in Ω^\sqcup mirrors that of Section 3: we first rewrite an expression into a disjunctive normal form and then recursively construct an expression in Ω^\sqcup for every term of the disjunction by merging pairs of factors.

Let $e \neq \emptyset$ be a balanced ω -regular expression. Without loss of generality, we may assume that $e = e_1 e_2^\omega + \dots + e_{2m-1} e_{2m}^\omega$, where every e_i is a regular expression containing no $+$ or \emptyset . Otherwise, we can rewrite it accordingly. We show how to construct an expression in Ω^\sqcup for $e_1 e_2^\omega$.

Since $\nabla^L(e, i) = \nabla^U(e, i) = \nabla^{\min}(e, i) = 0$ by Theorem 19, it follows that $\nabla^{\min}(e_1, i) = \nabla^L(e_2, i) = \nabla^U(e_2, i) = 0$. Then, by Lemma 12, we can write e_1 as a shuffle of factors $\langle \ominus_i, \langle \lambda_i, \oplus_i \rangle$ and e_2 as a shuffle of factors $\langle \ominus_i, \langle \lambda_i, \oplus_i, \star_i \rangle$. The idea is then to: (a) rewrite e_2^ω in terms of factors $\langle \ominus_i, \langle \lambda_i, \oplus_i^\omega, \star_i^\omega \rangle$ and then; (b) merge every \oplus_i in e_1 with a \oplus_i^ω in e_2^ω into $\langle \ominus_i^\omega$, using Lemma 11. We run into two complications:

- In step (a), e_2^ω may not necessarily be expressible as a single shuffle of factors. Consider $e_2 = [1]_1([2]_2)^* \equiv \sqcup_{11(22)^*}(\langle \ominus_1, \langle \lambda_2 \rangle)$: then e_2^ω contains both words with finite and infinite numbers of 2-parentheses. The latter requires a factor $\langle \ominus_2^\omega$, while the former requires its absence.

To remedy this, we write e_2^ω as a *disjunction* of shuffles of factors; one for every combination of finite and infinite versions of $\langle \ominus_i, \langle \lambda_i \rangle$. In this case:

$$\begin{aligned} e_2^\omega &\equiv \sqcup_{(11(22)^*)^\omega} (\langle \ominus_1, \langle \lambda_2 \rangle) \\ &\quad + \sqcup_{(11(22)^*)^\omega} (\langle \ominus_1^\omega, \langle \lambda_2 \rangle) \\ &\quad + \sqcup_{(11(22)^*)^\omega} (\langle \ominus_1, \langle \ominus_2^\omega \rangle) \\ &\quad + \sqcup_{(11(22)^*)^\omega} (\langle \ominus_1^\omega, \langle \ominus_2^\omega \rangle) \end{aligned}$$

The second term of the disjunction contains all words with finitely many 2-parentheses and the fourth term contains all those with infinitely many.

Note that the first and third terms yield empty languages, but this does not affect the resulting language. It is thus unnecessary to exclude these.

This is further detailed in Lemma 22.

- In step (b), the number of $\langle \oplus \rangle_i^\omega$ in a term of e_2^ω may not necessarily match the number of $\langle \oplus \rangle_i$ in e_1 : if $e_1 = [1]$ and $e_2 = [1]_1$, then e_1 contains one $\langle \oplus \rangle_1$ and e_2 contains one factor $\langle \ominus \rangle_1$. To solve this, we use two observations:
 - We can apply Lemma 11 to split a $\langle \ominus \rangle_i$ into $\langle \oplus \rangle_i$ and $\langle \ominus \rangle_i$, inverting the direction in which we have used the lemma so far.
 - Since $e_2^\omega \equiv (e_2 \cdot e_2)^\omega$, we can produce arbitrarily many copies of the factors in e_2 .

Combining the above, we can always split a $\langle \ominus \rangle_i$ into $\langle \oplus \rangle_i$ and $\langle \ominus \rangle_i$, then create copies of them and merge them back into one $\langle \ominus \rangle_i$ and one $\langle \oplus \rangle_i$. Since we can merge all other factors with their own copy, this effectively adds one $\langle \oplus \rangle_i$. Now that we have at least one, we can create more: we create a copy of every factor, then merge every factor with its own copy except for some number of $\langle \oplus \rangle_i$. This is further detailed in Lemma 23.

Lemma 22. *Let $e = \sqcup_\theta(e_1, \dots, e_n) \in \mathbb{E}^\sqcup$ be a shuffle of factors $\langle \ominus \rangle_i, \langle \lambda \rangle_i, \langle \oplus \rangle_i$ such that θ fits every e_j and contains no $+$. Then $e^\omega \equiv \hat{e}_1 + \dots + \hat{e}_m$, where $\hat{e}_k = \sqcup_{\theta_k}(e_{k,1}, \dots, e_{k,n})$ is a shuffle of factors $\langle \ominus \rangle_i, \langle \lambda \rangle_i, \langle \oplus \rangle_i^\omega, \langle \oplus \rangle_i^\omega$ for every k such that the number of $\langle \oplus \rangle_i$ in e is the same as the number of $\langle \oplus \rangle_i^\omega$ in \hat{e}_k for every i , and θ_k fits every $e_{k,j}$.*

Proof. Let $\varphi : \mathbb{E} \mapsto 2^{\mathbb{E} \cup \Omega}$ such that $\varphi(\langle \ominus \rangle_i^k) = \{\langle \ominus \rangle_i^k, \langle \oplus \rangle_i^\omega\}$, $\varphi(\langle \lambda \rangle_i^k) = \{\langle \lambda \rangle_i^k, \langle \oplus \rangle_i^\omega\}$ and $\varphi(\langle \oplus \rangle_i^k) = \{\langle \oplus \rangle_i^\omega\}$. We can then show that $e^\omega \equiv \hat{e}_1 + \dots + \hat{e}_m$, where $\{\hat{e}_1, \dots, \hat{e}_m\} = \{\sqcup_{\theta^\omega}(e'_1, \dots, e'_n) \mid e'_1 \in \varphi(e_1), \dots, e'_n \in \varphi(e_n)\}$.

Moreover, since φ maps $\langle \oplus \rangle_i$ to $\langle \oplus \rangle_i^\omega$, the number of factors $\langle \oplus \rangle_i^\omega$ in every \hat{e}_k matches the number of factors $\langle \oplus \rangle_i$ in e . However, if $\hat{e}_k = \sqcup_{\theta^\omega}(e'_1, \dots, e'_n)$, then θ^ω may not necessarily fit every e'_j : if e'_j is one of $\langle \ominus \rangle_i, \langle \lambda \rangle_i$, then there are $t \in L(\theta^\omega)$ with infinitely many j , while every word in $L(e'_j)$ is finite. Instead of θ^ω , we can use the trajectory $\theta^* \cdot \psi(\theta)^\omega$, where ψ is a homomorphism such that $\psi(j) = \lambda$ if e'_j is one of $\langle \ominus \rangle_i, \langle \lambda \rangle_i$ and $\psi(j) = j$ otherwise. This covers exactly the part of θ^ω that fits every e'_j . \square

Lemma 23. *Let $\sqcup_\theta(e_1, \dots, e_n) \equiv e \in \mathbb{E}$ be a shuffle of factors $\langle \ominus \rangle_i, \langle \lambda \rangle_i, \langle \oplus \rangle_i, \langle \star \rangle_i$ such that θ fits every e_j and contains no $+$, and $\xi(e, i)$. If there are ℓ factors $\langle \oplus \rangle_i, \langle \star \rangle_i$ among e_1, \dots, e_n , then for every $k \geq \ell$ (such that $k > 0$), there exists some shuffle of factors $\hat{e} = \sqcup_{\hat{\theta}}(\hat{e}_1, \dots, \hat{e}_m)$ such that $e^\omega \equiv \hat{e}^\omega$, \hat{e} contains k factors $\langle \oplus \rangle_i$ and no $\langle \star \rangle_i$ and $\hat{\theta}$ fits every \hat{e}_j .*

Proof. This proof consists of three steps. First, we need to make sure that we have at least one $\langle \oplus \rangle_i$. Second, we replace any remaining factors $\langle \star \rangle_i$ with $\langle \oplus \rangle_i$. Third, we create additional copies of $\langle \oplus \rangle_i$ as needed.

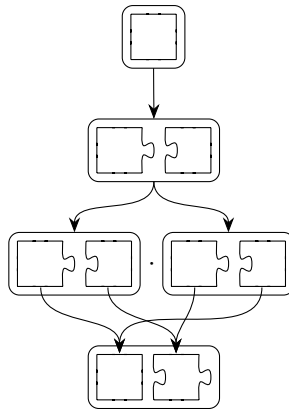
- (1) Suppose that there are no $\langle \pm \rangle_i$ among e_1, \dots, e_n . Then our first step consists of creating one. Since $\xi(e, i)$ and θ contains no $+$, there exists some $e_j \in \{\langle \circ \rangle_i, \langle \lambda \rangle_i, \langle * \rangle_i\}$ such that $|t|_j > 0$ for every $t \in L(\theta)$. Without loss of generality, we may assume that $j = n$.

If $e_n = \langle * \rangle_i^k$, since $|t|_n > 0$ for every t then $e \equiv \sqcup_{\theta}(e_1, \dots, \langle \pm \rangle_i^k)$ and we can proceed with step 2. Otherwise, if $e_n = \langle \lambda \rangle_i^k$, then $e \equiv \sqcup_{\theta}(e_1, \dots, \langle \circ \rangle_i^k)$ and if $e_n = \langle \circ \rangle_i^0$, then $e \equiv \sqcup_{\theta}(e_1, \dots, \langle \circ \rangle_i^1)$. Going forward, we may thus assume that $e_n = \langle \circ \rangle_i^k$ with $k \geq 1$. Since $|t|_n > 0$ for every $t \in L(\theta)$ and θ contains no $+$, it follows that $\theta = \theta_1 \cdot \theta_2$ such that both θ_1 and θ_2 only contain trajectories with odd numbers of n . We can then apply the proof of Lemma 11 to show that $e \equiv \sqcup_{\theta_3}(e_1, \dots, e_{n-1}, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2})$ for some θ_3, k_1, k_2 .

If e_1, \dots, e_{n-1} contain a $\langle \circ \rangle_i$, then without loss of generality we may assume that $e_{n-1} = \langle * \rangle_i^{k_3}$. We may assume that there exists some $t \in L(\theta)$ such that $|\theta|_{n-1} = 0$; otherwise we would have selected this factor as e_n earlier in this step and then proceeded with step 2. It follows that all trajectories in θ_1 and θ_2 , and therefore in θ_3 , contain even numbers of n . Then, in the same way that we split $\langle \circ \rangle_i^k$ into $\langle + \rangle_i^{k_1}$ and $\langle - \rangle_i^{k_2}$ before, we can show that $e \equiv \sqcup_{\theta_4}(e_1, \dots, e_{n-2}, \langle * \rangle_i^{k_4}, \langle * \rangle_i^{k_5}, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2})$ for some θ_4, k_4, k_5 . As seen in Figure 8, we can then merge $\langle * \rangle_i^{k_4}$ with $\langle - \rangle_i^{k_2}$ and $\langle * \rangle_i^{k_5}$ with $\langle + \rangle_i^{k_1}$ to obtain $e \equiv \sqcup_{\theta_5}(e_1, \dots, e_{n-2}, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2})$ for some θ_5 . This covers the case where $k = \ell > 0$ but there are no factors $\langle \pm \rangle_i$. We may thus assume without loss of generality that $e \equiv \sqcup_{\theta_6}(e_1, \dots, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2})$ for some θ_6 .

Since we still lack a $\langle \pm \rangle_i$, we use that $e^\omega \equiv (e \cdot e)^\omega$ to construct $e' = \sqcup_{\theta_6}(e_1, \dots, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2}) \cdot \sqcup_{\theta_6}(e_1, \dots, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2}) \equiv \sqcup_{\theta_7}(e_1, \dots, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2}, e_1, \dots, \langle + \rangle_i^{k_1}, \langle - \rangle_i^{k_2})$ for some θ_7 . We can then merge the first $\langle + \rangle_i^{k_1}$ with the second $\langle - \rangle_i^{k_2}$ into $\langle \circ \rangle_i^{k_1+k_2+1}$ and merge the second $\langle + \rangle_i^{k_1}$ with the first $\langle - \rangle_i^{k_2}$ into $\langle \pm \rangle_i^{k_1+k_2}$. We can merge every other factor with its own copy, which gives us $e' \equiv \sqcup_{\theta_8}(e'_1, \dots, \langle \circ \rangle_i^{k_1+k_2+1}, \langle \pm \rangle_i^{k_1+k_2})$ and $e_1^\omega \equiv e^\omega$.

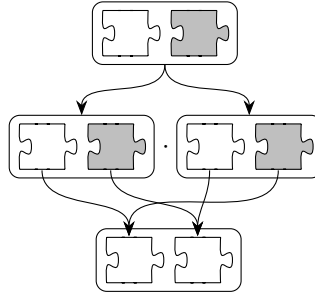
Not including corner cases, this step can be visualized as follows:



24 *L. Edithoven and S.-S. Jongmans*

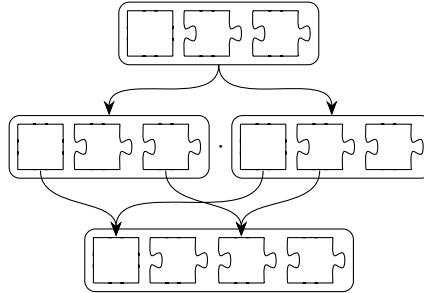
- (2) Now that we have at least one $\langle \pm \rangle_i$, we can reuse methods applied in the first step to replace any remaining $\langle \star \rangle_i$: create a copy of every factor using $e^\omega \equiv (e \cdot e)^\omega$, then merge the two copies of $\langle \star \rangle_i$ with the copies of some $\langle \pm \rangle_i$ as in Figure 8. By merging every other factor with its own copy, we effectively replace one $\langle \star \rangle_i$ with one $\langle \pm \rangle_i$. We repeat this step until there are no $\langle \star \rangle_i$ left.

This step can be visualized as follows:



- (3) Finally, by copying every factor and then merging every factor with its own copy except for a number of $\langle \pm \rangle_i$, we can create any additional number of $\langle \pm \rangle_i$, until we have some $\hat{e} = \sqcup_{\hat{\theta}}(\hat{e}_1, \dots, \hat{e}_m)$ with k $\langle \pm \rangle_i$. Since every rewriting step preserves equivalence of the ω -closures and the fitting of the trajectories, it follows that $\hat{e}^\omega \equiv e^\omega$ and that $\hat{\theta}$ fits every \hat{e}_j .

This step can be visualized as follows:



□

Summarizing, given $e_1 \cdot e_2^\omega$, by applying Lemmas 23 and 22 we can rewrite e_1 as a shuffle of factors $\langle \circ \rangle_i, \langle \lambda \rangle_i, \langle \pm \rangle_i$, and e_2^ω as a disjunction of shuffles of factors $\langle \circ \rangle_i, \langle \lambda \rangle_i, \langle \pm \rangle_i^\omega, \langle \pm \rangle_i^\omega$, such that the number of $\langle \pm \rangle_i^\omega$ in every term of the disjunction equals the number of $\langle \pm \rangle_i$ in e_1 . By applying the laws of distributivity, we can then rewrite $e_1 \cdot e_2^\omega$ as a disjunction of concatenations of shuffles. Since the numbers of $\langle \pm \rangle_i$ and $\langle \pm \rangle_i^\omega$ match in every term of this disjunction, we can apply Lemma 11 to merge every pair into $\langle \pm \rangle_i^\omega$. Since all factors are now balanced, every balanced ω -regular language has a corresponding expression in Ω^ω :

Theorem 24. $\{L(e) \mid e \in \Omega^\omega\} \supseteq \{L \mid L \text{ is a balanced } \omega\text{-regular language}\}.$

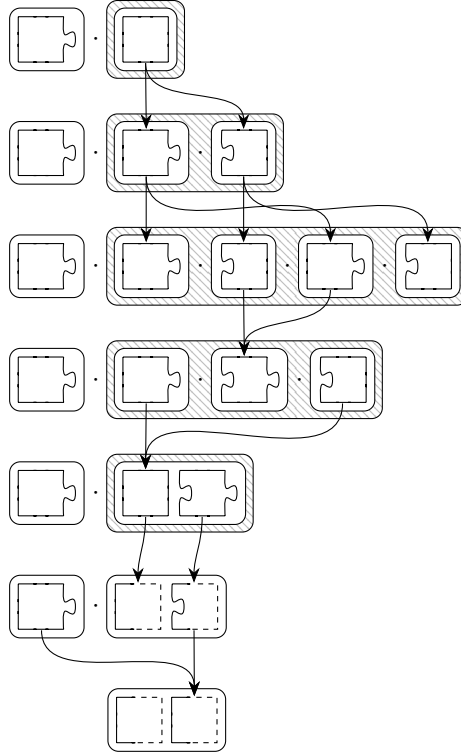


Fig. 19: Construction of Example 25 as groups of jigsaw pieces.

Example 25. We show how to construct an expression in Ω^ω for $e = [{}_{1}([{}_{1}]_1)]^\omega$.

$$\begin{aligned}
 [{}_{1}([{}_{1}]_1)]^\omega &\equiv \sqcup_1(\oplus_1^0)(\sqcup_{11}(\ominus_1^1))^\omega \\
 &\equiv \sqcup_1(\oplus_1^0)(\sqcup_1(\oplus_1^0) \sqcup_1(\ominus_1^0))^\omega \\
 &\equiv \sqcup_1(\oplus_1^0)(\sqcup_1(\oplus_1^0) \sqcup_{11}(\ominus_1^0) \sqcup_1(\oplus_1^0) \sqcup_1(\ominus_1^0))^\omega \\
 &\equiv \sqcup_1(\oplus_1^0)(\sqcup_1(\oplus_1^0) \sqcup_{11}(\oplus_1^0) \sqcup_1(\ominus_1^0))^\omega \\
 &\equiv \sqcup_1(\oplus_1^0)(\sqcup_{1221}(\ominus_1^1, \oplus_1^0))^\omega \\
 &\equiv \sqcup_1(\oplus_1^0) \sqcup_{(1221)^\omega}(\ominus_1^\omega, \oplus_1^\omega) \\
 &\equiv \sqcup_{1(2112)^\omega}(\ominus_1^\omega, \oplus_1^\omega).
 \end{aligned}$$

Figure 19 illustrates this construction using groups of jigsaw pieces, line by line. The ω -operator is visualized as an extra hatched container.

5. Discussion

We end this paper with a discussion on two topics for future work.

5.1. *Balanced context-free languages*

Having dealt with balanced regular and ω -regular languages, the natural next step is to consider balanced context-free languages. As these are languages of finite words, we can reuse the definition of balancedness as in Section 2.

In the same way we characterized balanced regular languages in terms of syntactic constraints on finite automata and regular expressions, it is possible to characterize balanced context-free languages in terms of syntactic constraints on pushdown automata and context-free grammars. In both cases, we add a certificate ∇ , just as for finite automata. For pushdown automata, this certificate does not only consider the state but also the content of the stack; for context-free grammars, the certificate assigns a value to the grammar's variables and terminals.

However, developing a balanced-by-construction grammar for context-free languages is not as straightforward as adding the shuffle on trajectories operator to a context-free grammar, such as the regular-like expressions described by Gruska [6]. The main complication is that context-free languages are not closed under the shuffle on trajectories operator, i.e., the shuffle of a set of context-free languages on a context-free language of trajectories is not necessarily context-free. In fact, Mateescu et al. show that $\sqcup_T(L_1, L_2)$ is guaranteed to be context-free only if at most one of T , L_1 and L_2 is context-free [10].

This holds even for seemingly simple sets of trajectories. For instance, the context-free language $[_1^n [_2]_1^n]_2$ can be straightforwardly expressed as $\sqcup_{1^m 2^1 m^2}([_1^n]_1^n, [_2]_2)$. However, when we use the same context-free set of trajectories to shuffle the context-free language $L_1 = a^n b^n c^*$ with the regular language $L_2 = d^*$, the result is not context-free. To see this, note that $\sqcup_{1^m 2^1 m^2}(a^n b^n c^*, d^*) \cap a^+ b^+ d^+ c^+ d^+ = a^n b^n d c^{2n} d$, which is not context-free; as context-free languages are known to be closed under intersection with regular languages, $\sqcup_{1^m 2^1 m^2}(a^n b^n c^*, d^*)$ cannot be context-free.

Developing a balanced-by-construction grammar for context-free languages thus is a non-trivial open problem. We note that there exists further research on grammars with trajectories [9, 14], which may inspire future efforts towards this goal.

5.2. *Binary shuffles*

Another non-trivial question is whether the grammar \mathbb{E}^\sqcup in Figure 4 is equally expressive if we restrict the shuffle on trajectories operator to its form with two operands: is it possible to rewrite any n -ary shuffle to (nested) binary ones?

As a simple first example, take $\sqcup_{123123}([_1]_1, [_2]_2, [_3]_3) \equiv [_1 [_2 [_3]_1]_2]_3$. This ternary shuffle can straightforwardly be rewritten with nested binary shuffles as $\sqcup_{122122}([_1]_1, \sqcup_{1212}([_2]_2, [_3]_3))$. The reason this example is easy, is because dif-

ferent types of parentheses can be neatly isolated into separate operands of the (nested) binary shuffles.

As a more complicated second example, take:

$$\sqcup_{1(2112)^*(3131)^*1}((\llbracket 1 \rrbracket_1)^*, (\llbracket 2 \rrbracket_2)^*, (\llbracket 3 \rrbracket_3)^*) \equiv \llbracket 1(\llbracket 2 \rrbracket_1 \llbracket 1 \rrbracket_2)^*(\llbracket 3 \rrbracket_1 \rrbracket_3 \llbracket 1 \rrbracket_1)^* \rrbracket_1$$

The first difficulty is that different types of parentheses cannot be neatly isolated as in the previous example. Moreover, a second difficulty is that the two loops in the trajectory expression mix the loops of the operands; as a result, neither one of those loops can be neatly isolated. To overcome these difficulties, a “trick” that we can use is to decompose the expression based on the number of times that the second loop (in the trajectory expression) is repeated: 0 times or at least once. In the former case, the expression simplifies to $\llbracket 1(\llbracket 2 \rrbracket_1 \llbracket 1 \rrbracket_2)^* \rrbracket_1$; this one is straightforward to rewrite. In the latter case, the expression simplifies to $\llbracket 1(\llbracket 2 \rrbracket_1 \llbracket 1 \rrbracket_2)^* \llbracket 3 \rrbracket_1 \rrbracket_3 \llbracket 1(\llbracket 3 \rrbracket_1 \rrbracket_3 \llbracket 1 \rrbracket_1)^* \rrbracket_1$, which can be broken down as the concatenation of $\llbracket 1(\llbracket 2 \rrbracket_1 \llbracket 1 \rrbracket_2)^* \llbracket 3 \rrbracket_1 \rrbracket_3$ and $\llbracket 1(\llbracket 3 \rrbracket_1 \rrbracket_3 \llbracket 1 \rrbracket_1)^* \rrbracket_1$, both of which are straightforward to write with binary shuffles by extracting single pairs of parentheses at a time.

We believe it is possible to prove that this “trick” can be generalized to deal with any number of concatenated loops, i.e., any expression $v_0 w_1^* v_1 w_2^* \dots w_n^* v_n$ where all v_i and w_i are words. Consequently, the generalized “trick” can deal with any balanced regular expression without nested loops.

However, the generalized “trick” does not seem to work for nested loops. As a counterexample, consider $\llbracket 1(\llbracket 2 \rrbracket_1(\llbracket 3 \rrbracket_3 \llbracket 2 \rrbracket_2 \llbracket 3 \rrbracket_3)^* \llbracket 1 \rrbracket_2)^* \rrbracket_1$. A straightforward attempt would be to isolate the inner loop (and the parentheses that occur in it): $\sqcup_{1(21(222222)^*12)^*1}((\llbracket 1 \rrbracket_1)^*, (\llbracket 2(\llbracket 3 \rrbracket_3 \llbracket 2 \rrbracket_2 \llbracket 3 \rrbracket_3)^* \rrbracket_2)^*)$. However, the shuffle operator has no way to distinguish, e.g., $\llbracket 2 \rrbracket_2 \llbracket 2 \rrbracket_2 \llbracket 2 \rrbracket_2 \llbracket 2 \rrbracket_2$ (four iterations of the outer loop and none of the inner) and $\llbracket 2 \rrbracket_3 \llbracket 3 \rrbracket_2 \llbracket 2 \rrbracket_3 \rrbracket_3 \rrbracket_2$ (one iteration of both), as both have length 8. As a result, it also accepts the word $\llbracket 1 \llbracket 2 \rrbracket_1 \rrbracket_2 \llbracket 2 \rrbracket_2 \llbracket 2 \rrbracket_2 \llbracket 2 \rrbracket_1 \rrbracket_2 \rrbracket_1$, which is not in the original language. We have as of yet found no way to avoid this ambiguity and to express this and expressions with nested loops in general using only binary shuffle on trajectories operators.

Acknowledgments

We thank Hendrik Jan Hoogeboom for his thoughts in an early stage of this research and in particular for drawing our attention to the work of Mateescu et al. [10]. We thank Benjamin Lion and Hans-Dieter Hiep for the various discussions on binary shuffles (Section 5.2).

The second author was funded by the Netherlands Organisation of Scientific Research (NWO): 016.Veni.192.103.

References

- [1] N. Chomsky and M. Schützenberger, The algebraic theory of context-free languages, *Computer Programming and Formal Systems, Studies in Logic and the Foundations of Mathematics* **26** (Elsevier, 1959), pp. 118 – 161.

- [2] M. Domaratzki, Trajectory-based embedding relations, *Fundam. Informaticae* **59**(4) (2004) 349–363.
- [3] P. Duchon, On the enumeration and generation of generalized Dyck words, *Discret. Math.* **225**(1-3) (2000) 121–135.
- [4] L. Edixhoven and S. Jongmans, Balanced-by-construction regular and ω -regular languages, *DLT, Lecture Notes in Computer Science* **12811**, (Springer, 2021), pp. 130–142.
- [5] L. Edixhoven and S.-S. Jongmans, Balanced-by-construction regular and ω -regular languages (technical report), Tech. Rep. OUNL-CS-2021-1, Open University of the Netherlands (2021).
- [6] J. Gruska, A characterization of context-free languages, *J. Comput. Syst. Sci.* **5**(4) (1971) 353–364.
- [7] J. Labelle and Y. Yeh, Generalized Dyck paths, *Discret. Math.* **82**(1) (1990) 1–6.
- [8] J. Liebehenschel, Lexicographical generation of a generalized Dyck language, *SIAM J. Comput.* **32**(4) (2003) 880–903.
- [9] C. Martín-Vide, A. Mateescu, G. Rozenberg and A. Salomaa, Contexts on trajectories, *Int. J. Comput. Math.* **73**(1) (1999) 15–36.
- [10] A. Mateescu, G. Rozenberg and A. Salomaa, Shuffle on trajectories: Syntactic constraints, *Theor. Comput. Sci.* **197**(1-2) (1998) 1–56.
- [11] A. Mateescu, K. Salomaa and S. Yu, On fairness of many-dimensional trajectories, *J. Autom. Lang. Comb.* **5**(2) (2000) 145–157.
- [12] M. Moortgat, A note on multidimensional Dyck languages, *Categories and Types in Logic, Language, and Physics, Lecture Notes in Computer Science* **8222**, (Springer, 2014), pp. 279–296.
- [13] D. E. Muller, Infinite sequences and finite machines, *SWCT*, (IEEE Computer Society, 1963), pp. 3–16.
- [14] A. Okhotin and K. Salomaa, Contextual grammars with uniform sets of trajectories, *Fundam. Informaticae* **64**(1-4) (2005) 341–351.
- [15] H. Prodinger, On a generalization of the Dyck-language over a two letter alphabet, *Discret. Math.* **28**(3) (1979) 269–276.