Measuring the Expressive Power of Practical Regular Expressions by Classical Stacking Automata Models*

Taisei Nogami[a,*], Tachio Terauchi[a,*]

[a]Waseda University, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan

Abstract

A *rewb* is a regular expression extended with a feature called backreference. It is broadly known that backreference is a practical extension of regular expressions, and is supported by most modern regular expression engines, such as those in the standard libraries of Java, Python, and more. Meanwhile, *indexed languages* are the languages generated by indexed grammars, a formal grammar class proposed by A.V.Aho. We show that these two models' expressive powers are related in the following way: every language described by a rewb is an indexed language. As the smallest formal grammar class previously known to contain rewbs is the class of context sensitive languages, our result strictly improves the known upper-bound. Moreover, we prove the following four claims: (1) there exists a rewb whose language does not belong to the class of stack languages, which is a proper subclass of indexed languages, (2) the language described by a rewb without a captured reference is in the class of nonerasing stack languages, which is a proper subclass of stack languages, (3) there exists a rewb that describes a stack language but not a nonerasing stack language, and (4) a rewb extended with another practical extension called lookaheads can describe a non-indexed language. Finally, we show that the hierarchy investigated in a prior study, which separates the expressive power of rewbs by the notion of nested levels, is within the class of nonerasing stack languages.

Keywords: Regular expressions, Backreferences, Lookaheads, Expressive power

1. Introduction

A *rewb* is a regular expression empowered with a certain extension, called *backreference*, that allows preceding substrings to be used later. It is closer to practical regular expressions than the pure ones, and supported by the standard libraries of most modern programming languages. A typical example of a rewb follows:

Example 1. Let Σ be the alphabet $\{a, b\}$. The language $L(\alpha)$ described by the rewb $\alpha = ({}_1(a+b)^*)_1 \backslash 1$ is $\{ww \mid w \in \Sigma^*\}$. Intuitively, α first *captures* a preceding string $w \in L((a+b)^*)$ by $({}_1\,)_1$, and second *references* that w by following $\backslash 1$. Therefore, α matches ww. Because this $L(\alpha)$ is a textbook example of a non-context-free language (and therefore non-regular), the expressive power of rewbs exceeds that of the pure ones.

In 1968, A.V.Aho discovered indexed languages with characterizations by two equivalent models: indexed grammars and (1N[1]) nested stack automata (Nested-SA) [1, 2]. The class of indexed languages is a proper superclass of context free languages (CFL), and a proper subclass of context sensitive languages (CSL) [1].

Berglund and van der Merwe [4], and Câmpeanu et al. [6] have shown that the class of rewbs is incomparable with the class of CFLs and is a proper subclass of CSLs. As the first main contribution of this paper, we prove that the language described by a rewb is an indexed language. Since the class of CSLs was the previously known best upper-bound of rewbs, our result gives a novel and strictly tighter upper-bound.

*This work was supported by JSPS KAKENHI Grant Numbers JP20K20625 and JP23K24826.

*Corresponding authors.

[1]One-way nondeterministic. "One-way" means that the input cursor will not move back to left. The antonym is "two-way."

Meanwhile, there is a class of the languages called stack languages [13, 12]. This class corresponds to the model (1N) stack automata (SA), a restriction of Nested-SA. Hence, it trivially follows that the class of stack languages is a subclass of indexed languages. Actually, this containment is known to be proper [2]. Furthermore, a model called nonerasing stack automata (NESA) has been studied in papers such as [13, 16, 20], and its language class is known to be a proper subclass of stack languages [20].

In this paper, we show that every rewb without a captured reference (that is, one in which no reference $\backslash i$ appears as a subexpression of an expression of the form $({}_i\alpha)_i$) describes a nonerasing stack language. Given our result, the following question is natural: does every rewb describe a (nonerasing) stack language? We show that the answer is no. Namely, we show a rewb that describes a non-stack language. Simultaneously, we also show a rewb that describes a stack language but not a nonerasing stack language.[2]

Additionally, we investigate the relationship between stacking automata models and rewb extended with another popular practical extension called *lookaheads*.[3] Namely, does every rewb with lookaheads describe an indexed language? We again answer the question negatively.

Finally, Larsen [17] has proposed a notion called *nested levels* of a rewb and showed that they give rise to a concrete increasing hierarchy of expressive powers of rewbs by exhibiting, for each nested level $i \in \mathbb{N}$, a language L_i that is expressible by a rewb at level i but not at any levels below i. We show that this hierarchy is within the class of nonerasing stack languages, that is, there exists an NESA A_i recognizing L_i for every nested level i. Below, we summarize the main contributions of the paper.

(a) Every rewb describes an indexed language. (Section 4, Corollary 24)
(b) Every rewb without a captured reference describes a nonerasing stack language. (Section 4, Corollary 25)
(c) There exists a rewb that describes a non-stack language. (Section 5, Theorem 26 and Corollary 31)
(d) There exists a rewb that describes a stack language but not a nonerasing stack language. (Section 5, Corollary 33)
(e) A rewb with lookaheads can describe a non-indexed language. (Section 6, Corollary 39)
(f) The hierarchy given by Larsen [17] is within the class of nonerasing stack languages. (Section 7, Theorem 40)

Note that by (b) and (c), it follows that there is a rewb language that no rewb without a captured reference can describe (Section 5, Corollaries 27 and 32). See also Figure 4 for a summary of the results.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 defines preliminary notions used in the paper such as the syntax and semantics of rewb, SA, NESA, and Nested-SA. Sections 4, 5, 6, and 7 formally state and prove the paper's main contributions listed above. Section 8 concludes the paper with a discussion on future work.

This paper is an extended version of the conference paper [19] and contains the following additional contributions:

1. We present a more natural example of a rewb that describes a non-stack language. The non-stack rewb $\alpha_0 = ((_1\backslash 4\, a)_1\, (_2\backslash 3)_2\, (_3\backslash 2\, a)_3\, (_4\backslash 1\backslash 3)_4)^*$ given in the conference version deeply relied on mutual backreferences and was somewhat artificial. We present a new non-stack rewb that does not depend on such a complex structure. Furthermore, we show that a slight modification of the rewb yields the *first* rewb whose language belongs to the stack languages but not to the nonerasing stack languages.
2. We have shown that the language class of regular expressions extended with backreferences is contained in the class of indexed languages, but does the containment still hold if they are further extended with lookaheads? We settle this question negatively, that is, the languages described by rewbs extended with lookaheads are not always indexed languages.

2. Related work

First, we discuss related work on rewbs. There are several variants of the syntax and semantics of rewbs since they first appeared in the seminal work by Aho [3]. A recent study by Berglund and van der Merwe [4] summarizes the variants and the relations between them. In sum, there are two variants of the syntax, whether or not a

[2] Note that this language witnesses the separation between the classes of stack languages and nonerasing stack languages, but this separation has been already shown by Ogden [20].

[3] See Example 36 for an example of lookaheads.

same label may appear as the index of more than one capture ("may repeat labels", "no label repetitions"), and two variants of the semantics, whether an unbound reference is interpreted as the empty string or an undefined factor (ε-semantics, $\emptyset$-semantics). As shown in [4], there is no difference in the expressive powers between these two semantics under the "may repeat labels" syntax (therefore, there are three classes with different expressive powers, namely "no label repetitions" with $\emptyset$-semantics, "no label repetitions" with ε-semantics, and "may repeat labels"). In this paper, we focus on the "may repeat labels" formalization, which has the highest expressive power of the three and is often studied in formal language theory. We adopt the ε-semantics as the semantics of rewbs. Note that the pioneering formalization of rewbs given by Aho [3] has the equivalent expressive power as this class. The rewbs with "may repeat labels" with ε-semantics was recently proposed by Schmid with the notions of a ref-word and a dereferencing function [21]. Simultaneously, he proposed a class of automata called *memory automata* (MFA), and showed that its expressive power is equivalent to that of rewbs. Freydenberger and Schmid extended MFA to *MFA with trap-state* [10]. Berglund and van der Merwe [4] showed that the class of Schmid's rewbs is a proper subclass of CSLs, and is incomparable with the class of CFLs. Note that there is a pumping lemma for the formalization given by Câmpeanu et al. [6] but it is known not to work for Schmid's rewbs. As mentioned above, Larsen introduced the notion of nested levels and showed that increase in the levels increases the expressive powers of rewbs [17]. As for lookaheads, Morihata [18] and Berglund et al. [5] showed that regular expressions with lookaheads are still regular, but Chida and Terauchi [8] showed that rewbs with lookaheads are strictly more powerful than rewbs.

Next, we discuss related work on the three automata models used throughout the paper, namely SA, NESA, and Nested-SA. Ginsburg et al. introduced SA as a mathematical model that is more powerful than pushdown automaton (PDA), and NESA as a restricted version of SA [13]. Hopcroft and Ullman discovered a type of Turing machine corresponding to the class of two-way NESA [16]. Ogden proposed a pumping lemma for stack languages and non-erasing stack languages [20]. Aho proposed Nested-SA with a proof of the fact that (1N) Nested-SA and indexed grammars given by himself in [1] are equivalent in their expressive powers, and recognized PDA and SA as special cases of Nested-SA [2]. Aho also showed that the class of indexed languages is a proper superclass of CFLs, and a proper subclass of CSLs [1]. Hayashi proposed a pumping lemma for indexed languages [14].

3. Preliminaries

3.1. Syntax and semantics of rewbs

In this section, we formalize the syntax and the semantics of rewbs following the formalization given in [10]. We begin with the syntax. Let $\Sigma_\varepsilon = \Sigma \uplus \{\varepsilon\}$ and $[k] = \{1, 2, \ldots, k\}$, where the symbol $\uplus$ denotes a disjoint union.

Definition 2. For each natural number $k \geq 1$, the set of *k-rewbs* over Σ, also written k-rewb by abuse of notation, and the mapping $\mathrm{var} : k\text{-rewb} \to \mathcal{P}([k])$ are defined as follows, where $a \in \Sigma_\varepsilon$ and $i \in [k]$:

$$(\alpha, \mathrm{var}(\alpha)) ::= (a, \emptyset) \mid (\backslash i, \{i\}) \mid (\alpha_0\alpha_1, \mathrm{var}(\alpha_0) \cup \mathrm{var}(\alpha_1)) \mid (\alpha_0 + \alpha_1, \mathrm{var}(\alpha_0) \cup \mathrm{var}(\alpha_1))$$
$$\mid (\alpha_0^*, \mathrm{var}(\alpha_0)) \mid ((_j\alpha_0)_j, \mathrm{var}(\alpha_0) \uplus \{j\}) \text{ where } j \in [k] \backslash \mathrm{var}(\alpha_0).$$

Let 0-rewb denote the set of all regular expressions over Σ. Then, the set of all rewbs is $\bigcup_{k\geq 0} k\text{-rewb}$.

Example 3. For example, ε, a, $\backslash 1$, $a^*\backslash 1$, $(_1a^*)_1$, $((_1a^*)_1)^*$, $(_2a^*)_2\backslash 2$, $(_1a^*)_1(_2b^*)_2(\backslash 1 + \backslash 2)$, $(_2(_1(a+b)^*)_1\backslash 1)_2\,\backslash 2\,(_2\backslash 1)_2{}^*$, $((_1\backslash 4\,a)_1\,(_2\backslash 3)_2\,(_3\backslash 2\,a)_3\,(_4\backslash 1\backslash 3)_4)^*$ are rewbs. On the other hand, $(_1(_1a^*)_1)_1$, $(_1a^*\,\backslash 1)_1$, $(_1(_2(_1a^*)_1)_2)_1$ are not rewbs.

Note that this syntax allows multiple occurrences of captures with the same label, that is, we adopt the "may repeat labels" convention. Next, we define the semantics.

Definition 4. Let $B_k = \{[_i,]_i \mid i \in [k]\}$. The mapping $\mathcal{R}_k : k\text{-rewb} \to \mathcal{P}((\Sigma \uplus B_k \uplus [k])^*)$ is defined as follows, where $a \in \Sigma_\varepsilon$ and $i \in [k]$:

$$\mathcal{R}_k(a) = \{a\},\ \mathcal{R}_k(\backslash i) = \{i\},\ \mathcal{R}_k(\alpha_0\alpha_1) = \mathcal{R}_k(\alpha_0)\mathcal{R}_k(\alpha_1),$$
$$\mathcal{R}_k(\alpha_0 + \alpha_1) = \mathcal{R}_k(\alpha_0) \cup \mathcal{R}_k(\alpha_1),\ \mathcal{R}_k(\alpha^*) = \mathcal{R}_k(\alpha)^*,\ \mathcal{R}_k((_i\alpha)_i) = \{[_i\}\,\mathcal{R}_k(\alpha)\,\{]_i\}.$$

We let $\Sigma_k^{[*]}$ denote $\bigcup_{\alpha\in k\text{-rewb}} \mathcal{R}_k(\alpha)$.

Example 5. $\mathcal{R}_k((_1(a+b)^*)_1 \backslash 1) = \{[_1\}\{a,b\}^*\{]_1\}\{1\} = \{[_1\, w\,]_1\, 1 \mid w \in \{a,b\}^*\}$.

That is, we first regard a rewb α over Σ as a regular expression over $\Sigma \uplus B_k \uplus [k]$, deducing the language $\mathcal{R}_k(\alpha)$. The second step, described next, is to apply the *dereferencing (partial) function* $\mathcal{D}_k : (\Sigma \uplus B_k \uplus [k])^* \rightharpoonup \Sigma^*$ to each of its element.

We give an intuitive description of $\mathcal{D}_k$. First, $\mathcal{D}_k$ scans its input string from the beginning toward the end, seeking $i \in [k]$. If such i is found, $\mathcal{D}_k$ replaces this i with the substring obtained by removing the brackets in v that comes from the preceding $[_i\, v\,]_i$ if $[_i$ exists (if this $[_i$ has no corresponding $]_i$, $\mathcal{D}_k$ becomes undefined). Otherwise, $\mathcal{D}_k$ replaces this i with ε. The dereferencing function $\mathcal{D}_k$ repeats this procedure until all elements of $[k]$ appearing in the string are exhausted, then removes all remaining brackets. We let $v_{[r]}$ denote the string which $\mathcal{D}_k$ scans at the r^{th} number $n_r \in [k]$ at the r^{th} loop.

1. $[_1 a\, [_2 b]_2\, 2\,]_1\, 1$. In this example, $\mathcal{D}_k$ encounters $n_1 = 2$ first, and this 2 corresponds the preceding $[_2 b]_2$, therefore this 2 is replaced with $v_{[1]} = b$. As a result, the input string becomes $[_1 a\, [_2 b]_2\, b\,]_1\, 1$. $\mathcal{D}_k$ repeats this process again. Now, $\mathcal{D}_k$ locates $n_2 = 1$ corresponding the preceding $[_1 a\, [_2 b]_2\, b\,]_1$, so this 1 is replaced with $v_{[2]} = a[_2 b]_2 b$ but with the brackets erased. Therefore, we gain $[_1 a\, [_2 b]_2\, b\,]_1\, abb$. Finally, $\mathcal{D}_k$ removes all remaining brackets and produces $abbabb$. Here is the diagram: $[_1 a\, \underline{[_2 b]_2}\, 2\,]_1\, 1 \to \underline{[_1 a\, [_2 b]_2\, b\,]_1}\, 1 \to [_1 a\, [_2 b]_2\, b\,]_1\, abb \to abbabb$.
2. $[_1 a]_1\, 1\, [_1 bb]_1\, 1$. In this example, $n_1 = n_2 = 1$, $v_{[1]} = a$, $v_{[2]} = bb$, and

$$\underline{[_1 a]_1}\, 1\, [_1 bb]_1\, 1 \to [_1 a]_1\, a\, \underline{[_1 bb]_1}\, 1 \to [_1 a]_1\, a\, [_1 bb]_1\, bb \to aabbbb.$$

3. $abc\, 1\, 2$. In this example, $n_1 = n_2 = 1$, $v_{[1]} = v_{[2]} = \varepsilon$, and $abc\, 1\, 2 \to abc\, 2 \to abc$.

Note that an unbound reference is replaced with the empty string ε, that is, we adopt the ε-semantics. However, as mentioned in Section 2, this semantics' expressive power is equivalent to that of the $\emptyset$-semantics under the "may repeat labels" convention (see [4] for the proof). We define the language $L(\alpha)$ denoted by a k-rewb α to be $\mathcal{D}_k(\mathcal{R}_k(\alpha)) = \{\mathcal{D}_k(v) \mid v \in \mathcal{R}_k(\alpha)\}$ (Lemmas 6 and 8 ensure that $L(\alpha)$ is well-defined), and define the language class of rewbs as the union of the language classes of k-rewbs for all k.

Let $g : (\Sigma \uplus B_k)^* \to \Sigma^*$ denote the free monoid homomorphism where $g(x)$ is x for each $x \in \Sigma$, and ε for each $x \in B_k$. Every $v \in (\Sigma \uplus B_k \uplus [k])^*$ can be written uniquely in the form $v = v_0 n_1 v_1 \cdots n_m v_m$, where $m \geq 0$ (denoted by $\operatorname{cnt} v$), and $v_r \in (\Sigma \uplus B_k)^*$ and $n_r \in [k]$ for each $r \in \{0, \ldots, m\}$. Here, let $y_0 \triangleq v_0$ and for each $r \in \{1, \ldots, m\}$, $y_r \triangleq v_0 n_1 v_1 \cdots n_r v_r$. A string $v = v_0 n_1 v_1 \cdots n_m v_m$ over $\Sigma \uplus B_k \uplus [k]$ is said to be *matching* if

$$\forall r \in \{1, \ldots, m\}\,.\, \forall x_1, x_2.\, y_{r-1} = x_1 [_{n_r} x_2 \Longrightarrow (\exists x_2', x_3.\, x_2 = x_2']_{n_r} x_3 \ \wedge\ x_2' \notin [_{n_r},]_{n_r})$$

holds. Intuitively, a string v being matching means that for all $n_r \in [k]$ in v, if there exists a left bracket $[_{n_r}$ in the string immediately before n_r, then there is a right bracket $]_{n_r}$ in between this $[_{n_r}$ and n_r. The following three lemmas follow.

Lemma 6. *Given a matching string v, $\mathcal{D}_k(v) = g(v_0)\, g(v_{[1]})\, g(v_1) \cdots g(v_{[m]})\, g(v_m)$.*

Lemma 7. *A prefix of a matching string is matching. That is, if we decompose a string v into $v = xy$, x is matching. Moreover, $x_{[r]} = v_{[r]}$ holds for each $r = 1, \ldots, \operatorname{cnt} x\, (\leq \operatorname{cnt} v)$.*

Lemma 8. *Every $v \in \Sigma_k^{[*]}$ is matching.*

In the rest of this subsection, we formally define the dereferencing function $\mathcal{D}_k$ and the notation $v_{[r]}$, and prove the three lemmas above by observing some basic features of $\mathcal{D}_k$. The reader may skip and go to the next subsection.

Definition 9. Suppose that $\bot \notin \Sigma \uplus B_k \uplus [k]$, and we define a Turing machine with one tape $\mathcal{D}_k$ as follows:

$\mathcal{D}_k$ = "On input string $v \in (\Sigma \uplus B_k \uplus [k])^*$,

Step 1. Move the head to the right until it reads an $i \in [k]$, then go to Step 2. If such i does not exist, go to Step 6.

Step 2. The assertion $P_2 \triangleq$ 'The symbol the head points out now is the leftmost natural number $i \in [k]$ on the tape' holds. Move the head to the left until it reads a $[_i$, then go to Step 3. If such $[_i$ does not exist, go to Step 5.

Step 3. Let P_3 be 'There is a right bracket $]_i$ between the current head position and this i'. If P_3 holds, go to Step 4. Otherwise, go to Step 7.

Step 4. Move the head to the right one by one, seeking a bracket $]_i$. Note that no $j \in [k]$ can appear in this scan since P_2 and P_3 holds. Now, if the symbol written on the tape cell that the head points to is $a \in \Sigma$, then insert a into the position immediately preceding this i, and go right; if it is $b \in B_k \setminus \{]_i\}$, simply go right; if it is $]_i$, go to Step 5.

Step 5. Go back to Step 1 and remove this i.

Step 6. $P_6 \triangleq$ 'No $j \in [k]$ is written on the tape' holds. Again scan the tape from the beginning, and remove all brackets.

Step 7. Erase all symbols written on the tape, and write the symbol $\bot$."

The order of execution of $\mathcal{D}_k$ is $(12(345 + 5))^*1(6 + 237)$. Observe that $\mathcal{D}_k$ halts for any input because Step 5 cannot be taken arbitrarily many times. Thus, we think of $\mathcal{D}_k$ as a computable function $(\Sigma \uplus B_k \uplus [k])^* \to \Sigma^* \uplus \{\bot\}$.

Lemma 10. *Suppose that the loop unit (namely Steps 1,2,3,4,5 or Steps 1,2,5) is executed exactly m' times when an input string $v = v_0 n_1 v_1 \cdots n_m v_m$ ($m = \operatorname{cnt} v$) is given to $\mathcal{D}_k$. Then, for each $r \in \{0, 1, \ldots, m'\}$, let $v^{(r)}$ be the string written on the tape immediately after the r^{th} loop, and let $v_{[r]}$ be the string over $\Sigma \uplus B_k$ defined as follows:*

$$v_{[r]} \triangleq \begin{cases} \textit{the string bracketed in } [_i \textit{ and }]_i \textit{ scanned at Step 4}, & (\textit{if Steps 1,2,3,4,5 are executed}) \\ \varepsilon. & (\textit{if Steps 1,2,5 are executed}) \end{cases}$$

Under the assumptions above, for each $r \in \{0, 1, \ldots, m'\}$ the following equality holds:

$$v^{(r)} = v_0\, g(v_{[1]})\, v_1 \cdots g(v_{[r]})\, v_r n_{r+1} v_{r+1} \cdots n_m v_m.$$

Proof. When $r = 0$, the left side is $v^{(0)}$ and the right side is $v_0 n_1 v_1 \cdots n_m v_m = v$, as required. Recall that immediately before the $(r + 1)^{\text{st}}$ loop, the string written on the tape is $v^{(r)}$. It continues as follows:

- At Step 1, the head of $\mathcal{D}_k$ is placed at immediately after $g(v_{[r]})$ in

  $$v^{(r)} = v_0\, g(v_{[1]})\, v_1 \cdots g(v_{[r]})\, v_r \underline{\underline{n_{r+1}}} v_{r+1} \cdots n_m v_m,$$

  and moves to n_{r+1}. Henceforth, we write simply s_r for $v_0\, g(v_{[1]})\, v_1 \cdots g(v_{[r]})\, v_r$, which appears before n_{r+1}.

- At Step 2, there are two cases:
  - When $\mathcal{D}_k$ follows Steps 3,4,5, since it moves from Step 2 to Step 3 and P_3 holds, s_r is of the form $x_0 [_{n_{r+1}} v_{[r+1]}]_{n_{r+1}} x_1$. At Step 4 and Step 5, the strings written on the tape after each step are

    $$\text{Step 4: } s_r\, g(v_{[r+1]})\, \underline{\underline{n_{r+1}}} v_{r+1} \cdots n_m v_m, \text{ and Step 5: } s_r\, g(v_{[r+1]})\, v_{r+1} \cdots n_m v_m.$$

    By definition, this string after Step 5 is $v^{(r+1)}$.
  - When $\mathcal{D}_k$ follows Step 5, since $v_{[r+1]} = \varepsilon$, the string immediately after the execution of Step 5, namely $v^{(r+1)}$, is $s_r v_{r+1} \cdots n_m v_m = s_r\, g(v_{[r+1]})\, v_{r+1} \cdots n_m v_m$.

  In both cases, the equation holds for $r + 1$.

This completes the proof. □

Lemma 11. *For a matching string v, the loop of $\mathcal{D}_k$ runs exactly $m = \operatorname{cnt} v$ times. Hence, $\mathcal{D}_k(v) \in \Sigma^*$.*

Proof. Let m' be the number of loop iterations. Because obviously $m' \leq m$, we suppose $m' < m$ for contradiction. If so, since it is the case that the execution moves from Steps 1,2,3 to Step 7 at the $(m' + 1)^{\text{st}}$ loop, P_3 does not hold for $v^{(m')}$. By Lemma 10,

$$v^{(m')} = \underbrace{v_0\, g(v_{[1]})\, v_1 \cdots g(v_{[m']})\, v_{m'}}_{s_{m'}} \underline{\underline{n_{m'+1}}} v_{m'+1} \cdots n_m v_m$$

follows. Since i in the $(m' + 1)^{\text{st}}$ loop is this $n_{m'+1}$ and $s_{m'} \ni [_{n_{m'+1}}$, there is v_j among $v_0, \ldots, v_{m'}$ such that $v_j \ni [_{n_{m'+1}}$. This v_j can be written in the form $u_0 [_{n_{m'+1}} u_1$. Because v is matching and $y_{m'} \supseteq v_j \ni [_{n_{m'+1}}$, one of $u_1, v_{j+1}, \ldots, v_{m'}$ contains $]_{n_{m'+1}}$. This contradicts the fact that P_3 does not hold at Step 3. □

Proof of Lemma 6. In Lemma 11, the string $v^{(m)}$, which is written on the tape immediately after the m^{th} loop, becomes $g(v^{(m)})$ at Step 6, and therefore $\mathcal{D}_k$ halts, as required. □

Proof of Lemma 7. Immediate from Lemma 6. □

Finally, we prove that every $v \in \Sigma_k^{[*]}$ (see Definition 4) is matching, concluding $L(\alpha) \subseteq \Sigma^*$ with Lemma 11.

Lemma 12. *The following facts hold where $\alpha \in k$-rewb and $i \in [k]$:*

(a) $\mathcal{R}_k(\alpha) \subseteq \left(\Sigma \uplus \left\{ [_j,]_j \,\middle|\, j \in \mathrm{var}(\alpha) \right\} \uplus \mathrm{var}(\alpha)\right)^*$,
(b) $\forall v_1, v_2.\, v_1[_i v_2 \in \mathcal{R}_k(\alpha) \Longrightarrow \exists v_2', v_3.\, v_2 = v_2']_i v_3 \ \wedge\ v_2' \notin [_i,]_i, i.$

Proof. Immediate from the definitions of $\mathcal{R}_k(\alpha)$ and $\mathrm{var}(\alpha)$. □

Proof of Lemma 8. There exists $\alpha \in k$-rewb such that $v \in \mathcal{R}_k(\alpha)$. Hereafter, we let $m = \mathrm{cnt}\, v$ and $v = v_0 n_1 v_1 \cdots n_m v_m$. For each $r \in \{1, \ldots, m\}$, if y_{r-1} can be written in the form $x_1[_{n_r} x_2$, we obtain $v = x_1[_{n_r} x_2 n_r v_r \cdots n_m v_m$ and by Lemma 12 (b), $x_2 n_r v_r \cdots n_m v_m$ can be written in the form $x_2']_{n_r} x_3$ with $x_2' \notin [_{n_r},]_{n_r}, n_r$. Here $x_2']_{n_r} \notin n_r$ holds. Therefore, $x_2']_{n_r}$ is a prefix of x_2 and of course $x_2' \notin [_{n_r},]_{n_r}$. Hence, v is matching. □

3.2. Classical automata models

Next, we recall the notions of SA, NESA, and Nested-SA. In this paper, we unify their definitions based on [2, 12] to clarify the different capabilities of these models. First, we review NFA. Here is the definition in the textbook by Hopcroft et al. [15]:

Definition 13 ([15], p.57). A *nondeterministic finite automaton* N is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ a finite set of input symbols (also called alphabet), $q_0 \in Q$ a start state, $F \subseteq Q$ a set of final states, and $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ a transition function.

As well known, the transition function δ can be *extended* to $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ where $\hat{\delta}(q, w)$ represents the set of all states reachable from q via w. Let $q \xrightarrow[N]{a} q'$ denote $q' \in \delta(q, a)$, and $q \overset{w}{\underset{N}{\Rightarrow}} q'$ denote $q' \in \hat{\delta}(q, w)$. With this notation, the language of an NFA N can be written as follows: $L(N) = \left\{ w \in \Sigma^* \,\middle|\, \exists q_f \in F.\, q_0 \overset{w}{\underset{N}{\Rightarrow}} q_f \right\}$.

A pushdown automaton (PDA) is an NFA equipped with a *stack* such that the PDA may write and read its *stack top* with a transition. A *stack automaton* (SA) is "an extended PDA", which can reference not only the top but inner content of the stack. That is, while the *stack pointer* of a PDA is fixed to the top, an SA allows its pointer to move left and right and read a stack symbol pointed to by the pointer. However, the only place on the stack that can be rewritten is the top, as in PDA. Formally, a (1N) SA A is a 10-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \dashv, \#, \$, F)$ satisfying the following conditions: the components Q, Σ, q_0 and F are the same as those of NFA. $\Gamma\ (\neq \emptyset)$ is a finite set of stack symbols, and $Z_0 \in \Gamma$ is an initial stack symbol. The symbol $\dashv \notin \Sigma$ is the endmarker of the input and the symbol $\# \notin \Sigma \cup \Gamma$ (resp. $\$ \notin \Sigma \cup \Gamma$) is always and only written at the leftmost (bottom) (resp. the right most (top)) of the stack.[4] The transition function δ has the following two modes, where $\mathrm{L}, \mathrm{S}, \mathrm{R} \notin (\Sigma \cup \Gamma) \uplus \{\dashv, \#, \$\}$, $\Delta_i \triangleq \{\mathrm{S}, \mathrm{R}\}$, $\Delta_s \triangleq \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}$ and $\Sigma' \triangleq \Sigma \uplus \{\dashv\}$:

(i) (pushdown mode) $Q \times \Sigma' \times \Gamma\$ \to \mathcal{P}(Q \times \Delta_i \times \Gamma^*\$)$,
(ii) (stack reading mode) (a) $Q \times \Sigma' \times \Gamma\$ \to \mathcal{P}(Q \times \Delta_i \times \{\mathrm{L}, \mathrm{S}\})$, (b) $Q \times \Sigma' \times \Gamma \to \mathcal{P}(Q \times \Delta_i \times \Delta_s)$, (c) $Q \times \Sigma' \times \{\#\} \to \mathcal{P}(Q \times \Delta_i \times \{\mathrm{S}, \mathrm{R}\})$.

[4] These special symbols $\#, \$$ representing "bottom" and "top" of the stack respectively do not appear in [12] and are introduced anew in this paper to define NESA and Nested-SA, which will be defined later, in the style of [2]. In fact, SA defined in [12] is not capable of directly discerning whether the stack pointer is at the top or not. Although it is not difficult to see that directly adding the ability does not increase the expressive power of SA, the ability is directly in NESA as seen in [16, 20]. Therefore, to make it easy to see that NESA is a restriction of SA, we define SA to also directly have the ability.

Intuitively, δ works as follows (Definition 14 provides the formal semantics). (i) The statement $(q', d, w\$) \in \delta(q, a, Z\$)$ says that whenever the current state is q, the input symbol is a, and the pointer references the top symbol Z, the machine can move to the state q', move the input cursor along d, and replace Z with the string w. (ii) The statement (b) $(q', d, e) \in \delta(q, a, Z)$ says that whenever the current state is q, the input symbol is a, and the pointer references the symbol Z, the machine can move to the state q', move the input cursor along d, and move the pointer along e. The statements (a) and (c) are similar to (b) except that the direction in which the pointer can move is restricted lest the pointer go out of the stack. In particular, an SA that cannot erase a symbol once written on the stack is called a *nonerasing stack automaton* (NESA). That is, a (1N) nonerasing stack automaton is an SA whose transition function δ satisfies the condition that, in (i) (pushdown mode), $(q', d, y\$) \in \delta(q, a, Z\$)$ implies $y \in Z\Gamma^*$. To formally describe how SA works, we define a tuple called *instantaneous description* (ID), which consists of a state, an input string, and a string representation of the stack, and define the binary relation $\vdash_A$ over the set of these tuples. Let $\overline{\mathrm{L}} = -1$, $\overline{\mathrm{S}} = 0$, and $\overline{\mathrm{R}} = 1$.

Definition 14. Let A be an SA $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \dashv, \#, \$, F)$. An element of the set $I = Q \times \Sigma^* \{\dashv\} \times \{\#\} (\Gamma \uplus \{\restriction\})^* \{\$\}$ is called *instantaneous description*, where the stack symbol $\restriction \notin \Gamma$ stands for the position of stack pointer. Moreover, let $\vdash_A$ (or $\vdash$ when A is clear) be the smallest binary relation over I satisfying the following conditions for all $q, q' \in Q$, $k \geq 0$, $a_0, \ldots, a_k \in \Sigma \uplus \{\dashv\}$, $Z, Z_1, \ldots, Z_n \in \Gamma$, $\gamma, y \in \Gamma^*$, $d \in \Delta_i$ and $e \in \Delta_s$ with $k = 0 \to d \neq \mathrm{R}$:

(i) $(q, a_0 \cdots a_k, \#\gamma Z \restriction \$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#\gamma y \restriction \$)$ if $(q', d, y\$) \in \delta(q, a_0, Z\$)$.

(ii) (a) $(q, a_0 \cdots a_k, \#\gamma Z \restriction \$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#\gamma \restriction Z\$)$ if $(q', d, \mathrm{L}) \in \delta(q, a_0, Z\$)$, and
$(q, a_0 \cdots a_k, \#\gamma Z \restriction \$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#\gamma Z \restriction \$)$ if $(q', d, \mathrm{S}) \in \delta(q, a_0, Z\$)$.

(b) if $(q', d, e) \in \delta(q, a_0, Z)$, $Z = Z_j$ and $1 \leq j < n$ with $j = 1 \to e \neq \mathrm{L}$ and $j = n \to e \neq \mathrm{R}$, then $(q, a_0 \cdots a_k, \#Z_1 \cdots Z_j \restriction \cdots Z_n\$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#Z_1 \cdots Z_{j+\overline{e}} \restriction \cdots Z_n\$)$.

(c) $(q, a_0 \cdots a_k, \# \restriction Z\gamma\,\$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#Z \restriction \gamma\,\$)$ if $(q', d, \mathrm{R}) \in \delta(q, a_0, \#)$, and
$(q, a_0 \cdots a_k, \# \restriction Z\gamma\,\$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \# \restriction Z\gamma\,\$)$ if $(q', d, \mathrm{S}) \in \delta(q, a_0, \#)$.

Note that $\mathrm{L} \notin \Delta_i$, which means the input cursor will not move back to left. We say that A accepts $w \in \Sigma^*$ if there exist $\gamma_1, \gamma_2 \in \Gamma^*$, and $q_f \in F$ such that $(q_0, w\dashv, \#Z_0 \restriction \$) \vdash_A^* (q_f, \dashv, \#\gamma_1 \restriction \gamma_2\,\$)$. Let $L(A)$ denote the set of all strings accepted by A.

Notation 15. We introduce some arrow notations to draw SA diagrams. We denote the rule of (i) $(q', d, y\$) \in \delta(q, a, Z\$)$ as $q \xrightarrow{a, d/Z\$ \to y\$} q'$; (ii) $(q', d, e) \in \delta(q, a, \square)$ as $q \xrightarrow{a, d/\square, e} q'$ for $\square = Z\$, Z$ or #. If $d = \mathrm{R}$, we simply write $a, d/\cdots$ as $a/\cdots$. For convenience, we also introduce intuitive syntax sugars to represent multiple transitions at once. For instance, an ε-transition $\varepsilon/\cdots$ consists of putting $c, \mathrm{S}/\cdots$ for all $c \in \Sigma$, and we may omit the upper part in front of / and write the part $\cdots$ only. In (i), $\$ \to Z\$$ is a shorthand for putting $Z'\$ \to Z'Z\$$ for all $Z' \in \Gamma$. In (ii), for a finite subset $U \subseteq \Gamma$, the notation $\cdots/U, e$ (resp. $\cdots/U\$, e$) means putting $\cdots/u, e$ (resp. $\cdots/u\$, e$) for all $u \in U$. In particular, $U = \Gamma$, we omit U and merely type $\cdots/e$. The negation $\neg Z$ is a shorthand for $\Gamma \setminus \{Z\}$.

We next define *nested stack automaton* (Nested-SA) which is SA extended with the capability to create and remove substacks. For instance, suppose that the stack is $\#a_1a_2 \restriction a_3\$$ and we are to create a new substack containing b_1b_2:

$$\underline{\#a_1 \not\subset b_1b_2 \restriction \$}\, a_2a_3\$. \tag{1}$$

Note that the new substack $\not\subset b_1b_2\,\$$ is embedded below the symbol a_2 indicated by the stack pointer, and the pointer moves to the top of the created substack. The creation of the inner substack narrows the range within which the stack pointer can move as indicated by the underlined part $\#a_1 \not\subset b_1b_2 \restriction \$$. While the bottom of the entire stack is always fixed by the leftmost symbol #, the top of the embedded substack is regarded as the top of the entire stack. The inner substacks are allowed to be embedded endlessly and everywhere, whereas the writing in the pushdown mode is still restricted to the top of the stack:

$$\underline{\#a_1 \not\subset b_1b_2 \restriction \$}\, a_2a_3\$ \xrightarrow{\mathrm{L}} \underline{\#a_1 \not\subset b_1 \restriction b_2\,\$}\, a_2a_3\$ \xrightarrow{\text{create}} \underline{\#a_1 \not\subset \not\subset c_1c_2 \restriction \$}\, b_1b_2\$\, a_2a_3\$, \tag{2}$$

$$\underline{\#a_1 \not\subset \restriction b_1b_2\,\$}\, a_2a_3\$ \xrightarrow{\mathrm{L}} \underline{\#a_1 \restriction \not\subset b_1b_2\,\$}\, a_2a_3\$ \xrightarrow{\text{create}} \underline{\#\not\subset c_1c_2 \restriction \$}\, a_1 \not\subset b_1b_2\,\$\, a_2a_3\$. \tag{3}$$

We must empty the inner substack and then remove itself in advance whenever we want to reference the right side of the inner substack such as a_2, a_3. For example, let us empty the inner substack by popping twice from (1) and then removing it:

$$\underline{\#a_1 ¢\, b_1 b_2 \restriction \$}\, a_2 a_3 \$ \xrightarrow{\text{pop}} \underline{\#a_1 ¢\, b_1 \restriction \$}\, a_2 a_3 \$ \xrightarrow{\text{pop}} \underline{\#a_1 ¢ \restriction \$}\, a_2 a_3 \$ \xrightarrow{\text{destruct}} \underline{\#a_1 a_2 \restriction a_3 \$}. \tag{4}$$

Notice that the stack pointer moves to the right after removing the inner substack. We now define Nested-SA formally. A (1N) nested stack automaton A is a 10-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, ¢, \$, F)$ satisfying the following conditions: the components $Q, \Sigma, \Gamma, q_0, Z_0, \#, \$$ and F are the same as those of SA. The stack symbol $¢ \notin \Sigma \cup \Gamma$ represents the bottom of a substack.[5] The transition function δ has the following four modes, where $\Sigma' \triangleq \Sigma \uplus \{\dashv\}$ and $\Gamma' \triangleq \Gamma \uplus \{¢\}$:

(i) (pushdown mode) $Q \times \Sigma' \times \Gamma\$ \to \mathcal{P}(Q \times \Delta_i \times \Gamma^*\$)$.
(ii) (stack reading mode) (a) $Q \times \Sigma' \times \Gamma'\$ \to \mathcal{P}(Q \times \Delta_i \times \{\mathrm{L}, \mathrm{S}\})$, (b) $Q \times \Sigma' \times \Gamma' \to \mathcal{P}(Q \times \Delta_i \times \Delta_s)$, (c) $Q \times \Sigma' \times \{\#\} \to \mathcal{P}(Q \times \Delta_i \times \{\mathrm{S}, \mathrm{R}\})$.
(iii) (stack creation mode) $Q \times \Sigma' \times (\Gamma' \uplus \Gamma'\$) \to \mathcal{P}(Q \times \Delta_i \times \{¢\}\, \Gamma^*\$)$.
(iv) (stack destruction mode) $Q \times \Sigma' \times \{¢\}\, \$ \to \mathcal{P}(Q \times \Delta_i)$.

Moreover, we define how Nested-SA works with ID and $\vdash$ in the same manner as SA. Given a Nested-SA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \dashv, \#, ¢, \$, F)$, we define ID, $\vdash_A$, and $L(A)$ in the same way as Definition 14 (however, we let I be $Q \times \Sigma^* \{\dashv\} \times \{\#\}\, (\Gamma \uplus \{¢, \$, \restriction\})^*\, \{\$\}$). Here, we only give the rules corresponding to (iii) and (iv) in the definition of δ (the others are essentially the same as those of SA):

(iii) if $(q', d, ¢y\$) \in \delta(q, a_0, Z)$ and $Z = Z_j$, $1 \le j < n$, then
$(q, a_0 \cdots a_k, \#Z_1 \cdots Z_j \restriction \cdots Z_n\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#Z_1 \cdots ¢y \restriction \$Z_j \cdots Z_n\$)$,
and $(q, a_0 \cdots a_k, \#\gamma Z \restriction \$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#\gamma\, ¢y \restriction \$Z\$)$ if $(q', d, ¢y\$) \in \delta(q, a_0, Z\$)$.
(iv) $(q, a_0 \cdots a_k, \#\gamma_1 ¢ \restriction \$Z\gamma_2\$) \vdash_A (q', a_{\overline{d}} \cdots a_k, \#\gamma_1 Z \restriction \gamma_2\$)$ if $(q', d) \in \delta(q, a_0, ¢\$)$.

4. Every rewb describes an indexed language

4.1. Main theorem

As described above, to obtain the language $L(\alpha)$ described by a k-rewb α, we derive the regular language $\mathcal{R}_k(\alpha)$ over the alphabet $\Sigma \uplus B_k \uplus [k]$ first, then apply the dereferencing function $\mathcal{D}_k$ to every element of $\mathcal{R}_k(\alpha)$. Using this observation, we construct a Nested-SA A_α recognizing the language $L(\alpha)$ as follows.

The Nested-SA A_α is based on an NFA N recognizing the language $\mathcal{R}_k(\alpha)$, in the sense that each transition in A_α comes from a corresponding transition of N. The NFA N has the alphabet $\Sigma \uplus B_k \uplus [k]$, and so handles three types of characters. For each transition $q \xrightarrow[N]{a} q'$ with $a \in \Sigma$, i.e., moving from q to q' by an input symbol a, A_α also has the same transition except pushing a to the stack, denoted by $q \xrightarrow{a/\$ \to a\$} q'$. For each transition $q \xrightarrow[N]{b} q'$ with $b \in B_k$, i.e., moving by a bracket b, A_α has the transition pushing b without consuming input symbols, denoted by $q \xrightarrow{\varepsilon/\$ \to b\$} q'$.[6] For each transition $q \xrightarrow[N]{i} q'$ with $i \in [k]$, A_α has a large "transition" that consists of several transitions. In this "transition," A_α first seeks the left bracket $[_i$ of the bracketed string $[_i\, v]_i$ within the stack, and checks if the input from the cursor position matches v character by character while consuming the input, and finally moves to q' if all characters of v matched.

A difficult yet interesting point is that Nested-SA cannot check v against the stack and push v onto the stack at the same time, that is, after checking a character c of v, if A_α wants to push c to the stack, A_α must leave from v, climb up the stack toward the top, and write c. However, after the push, A_α becomes lost by not knowing where to go back to. How about marking the place where A_α should return in advance? Unfortunately, that does not work; Nested-SA can insert such marks anywhere by creating substacks, but due to the restriction of Nested-SA, it cannot go above

[5] Note that the bottom of the entire stack is always represented by # and not ¢, as mentioned above.
[6] See also Notation 15 for the ε-transition notation.

the position of the mark, much less climb up to the top. Therefore, Nested-SA cannot directly push the result of a dereferencing onto the stack.

We cope with this problem as follows. We allow $j \in [k]$ to appear in v, and for each appearance of j in the checking of v, A_α pauses the checking and puts a substack containing the current state as a marker at the stack pointer position. Then, A_α searches down the stack for the corresponding bracketed string $[_j v']_j$, and begins checking v' if it is found. By repeating this process, A_α eventually reaches a string $v'' \in (\Sigma \uplus B_k)^*$ containing no characters of $[k]$. Once done with the check of v'', A_α climbs up toward the stack top, finds a marker p denoting the state to return to, and resumes from p after deleting the substack containing the marker. By repeating this, if A_α returns to the position where it initially found j, it has successfully consumed the substring of the input string corresponding to the dereferencing of j. The following lemma is immediate.

Lemma 16. *Let $k \geq 1$ and $\alpha \in k$-rewb. There exists an NFA $(Q, \Sigma \uplus B_k \uplus [k], \delta, q_0, F)$ over $\Sigma \uplus B_k \uplus [k]$ recognizing $\mathcal{R}_k(\alpha)$ all of whose states can reach some final state, that is, $\forall q \in Q.\, \exists w \in (\Sigma \uplus B_k \uplus [k])^*.\, \exists q_f \in F.\, q \overset{w}{\underset{N}{\Rightarrow}} q_f$.*

Corollary 17. *Let N be the NFA in Lemma 16. For all $q \in Q$ and for all $w \in (\Sigma \uplus B_k \uplus [k])^*$, if $q_0 \overset{w}{\underset{N}{\Rightarrow}} q$ then w is matching.*

Proof. There are a string $w' \in (\Sigma \uplus B_k \uplus [k])^*$ and a final state $q_f \in F$ such that $q_0 \overset{w}{\underset{N}{\Rightarrow}} q \overset{w'}{\underset{N}{\Rightarrow}} q_f$. Hence, $ww' \in L(N) = \mathcal{R}_k(\alpha)$ follows. By Lemma 8 the string ww' is matching, therefore by Lemma 7 its prefix w is matching. □

We show the main theorem (the proof sketch is coming later):

Theorem 18. *For every rewb α, there exists a Nested-SA that recognizes $L(\alpha)$.*

The claim obviously holds when α is a pure regular expression (i.e., $\alpha \in 0$-rewb). Suppose that $\alpha \in k$-rewb with $k \geq 1$. By Lemma 16, there is an NFA $N = (Q_N, \Sigma \uplus B_k \uplus [k], \delta_N, q_0, F)$ that recognizes $\mathcal{R}_k(\alpha)$ and satisfies Corollary 17. We construct a Nested-SA $A_\alpha = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, \mathbin{¢}, \$, F)$ as follows. Let $Q \triangleq Q_N \uplus \{c_i, e_i, r_i \mid i \in [k]\} \uplus \left\{W_q \,\middle|\, q \in Q_N\right\} \uplus \left\{E_{p,i}, L_{p,i} \,\middle|\, p \in Q_N \uplus \{e_i \mid i \in [k]\}, i \in [k]\right\}$, $\Gamma \triangleq \Sigma \uplus B_k \uplus [k] \uplus Q \uplus \{Z_0\}$, and let δ be the smallest relation that, for all $a \in \Sigma$, $b \in B_k$, $c \in \Sigma \uplus \{\dashv\}$, $i, j \in [k]$, $q, q' \in Q_N$, $Z \in \Gamma$ and $p \in Q_N \uplus \{e_i \mid i \in [k]\}$, satisfies the following conditions:

(1) $\delta_N(q, a) \ni q' \Longrightarrow \delta(q, a, Z\$) \ni (q', \mathrm{R}, Za\$)$
(2) $\delta_N(q, b) \ni q' \Longrightarrow \delta(q, c, Z\$) \ni (q', \mathrm{S}, Zb\$)$
(3) $\delta_N(q, i) \ni q' \Longrightarrow \delta(q, c, Z\$) \ni (W_{q'}, \mathrm{S}, Zi\$)$
(4) $\delta(W_q, c, i\$) = \{(c_i, \mathrm{S}, \mathbin{¢}q\$)\}$
(5) $\delta(c_i, c, p\$) = \{(c_i, \mathrm{S}, \mathrm{L})\}$
(6) $\delta(c_i, c, Z) = \{(c_i, \mathrm{S}, \mathrm{L})\}$ where $Z \neq [_i, Z_0$
(7) $\delta(c_i, c, Z_0) = \{(r_i, \mathrm{S}, \mathrm{R})\}$
(8) $\delta(c_i, c, [_i) = \{(e_i, \mathrm{S}, \mathrm{R})\}$
(9) $\delta(e_i, a, a) = \{(e_i, \mathrm{R}, \mathrm{R})\}$
(10) $\delta(e_i, c, [_j) = \{(e_i, \mathrm{S}, \mathrm{R})\}$ where $i \neq j$
(11) $\delta(e_i, c,]_j) = \begin{cases} \{(r_i, \mathrm{S}, \mathrm{R})\} & (i = j) \\ \{(e_i, \mathrm{S}, \mathrm{R})\} & (i \neq j) \end{cases}$
(12) $\delta(e_i, c, j) = \left\{(c_j, \mathrm{S}, \mathbin{¢}e_i\$)\right\}$ where $i \neq j$
(13) $\delta(r_i, c, Z) = \{(r_i, \mathrm{S}, \mathrm{R})\}$
(14) $\delta(r_i, c, p\$) = \left\{(E_{p,i}, \mathrm{S}, \$)\right\}$
(15) $\delta(E_{p,i}, c, \mathbin{¢}\$) = \left\{(L_{p,i}, \mathrm{S})\right\}$
(16) $\delta(L_{e_j,i}, c, i) = \left\{(e_j, \mathrm{S}, \mathrm{R})\right\}$
(17) $\delta(L_{q,i}, c, i\$) = \{(q, \mathrm{S}, \mathrm{S})\}$

Rule (1) translates $q \overset{a}{\underset{N}{\to}} q'$ into $q \xrightarrow{a/\$ \to a\$} q'$, (2) translates $q \overset{b}{\underset{N}{\to}} q'$ into $q \xrightarrow{\varepsilon/\$ \to b\$} q'$, and rules (3)–(17) translate $q \overset{i}{\underset{N}{\to}} q'$ into a large "transition" to consume the string that corresponds to the dereferencing of i. The details of the "transition" are as follows. By looking at the underlying N with rule (3), A_α finds a state q' that it should go back to after going throughout the "transition," and goes to the state $W_{q'}$ by pushing i to the stack. At $W_{q'}$, by rule (4), A_α inserts $\mathbin{¢}q'\$$ just below i, and goes to the state c_i. The state c_i represents the *call mode* in which A_α looks for the left-nearest $[_i$ by rules (5) and (6) and proceeds to the state e_i (*execution mode*) by (8) if it finds $[_i$. Otherwise (i.e., the case when A_α arrives at the bottom of the stack), it proceeds to the state r_i (*return mode*) by rule (7). At e_i, A_α consumes input symbols by checking them against the symbols on the stack (rules (9)–(12)). In particular, rule (9)

handles the case when the symbols match. Rules (10) and (11) handle the cases when brackets are read from the stack. The first case of (11) handles the case when the right bracket $]_i$ is read, and the rules handle the other brackets (i.e., $[_j$ or $]_j$ with $i \neq j$) by simply skipping them (note that $[_j = [_i$ cannot happen since we started from the left-nearest $[_i$). Reading $j \in [k]$, by rule (12), A_α inserts $\mathbb{e}e_i\$$ just below j and goes to c_j to locate the corresponding $[_j$ (here, $j \neq i$ holds by the definition of the syntax). At r_i, A_α proceeds to return to the state p that passed the control to c_i (rules (13)–(17)). Since this p was pushed at the stack top, A_α first climbs up to the stack top by rule (13), transits to the state $E_{p,i}$ popping p by (14), then goes to $L_{p,i}$ removing the embedded substack by (15), and finally goes back to p by (16) and (17). A subtle point in the last step is that where the stack pointer should be placed depends on whether p is a state e_j (for some $j \in [k]$) or in Q_N. In the former case, after (15) removes the embedded substack $\mathbb{e}e_j\$$ that was created just below the call to i, the stack pointer points to i. However, the stack pointer should shift one more to the right, lest A_α begins to repeat the call reading i again by (12). Therefore, (16) correctly handles the case by doing the shift. In the latter case, as stipulated by (17), the stack pointer should point to the stack top symbol i since p is the state stored at (3).

Proof sketch of Theorem 18. For proving $L(\alpha) \subseteq L(A_\alpha)$, we take $w \in L(\alpha)$ and $v \in \mathcal{R}_k(\alpha)$ such that $w = \mathcal{D}_k(v)$. Decomposing v into $v_0 n_1 v_1 \cdots n_m v_m$ (where $m = \mathrm{cnt}\, v$), we obtain a transition sequence in the underlying NFA N, denoted by $q_0 \overset{v_0}{\underset{N}{\Longrightarrow}} q_{(0)} \overset{n_1 v_1}{\underset{N}{\Longrightarrow}} q_{(1)} \overset{n_2 v_2}{\underset{N}{\Longrightarrow}} \cdots \overset{n_m v_m}{\underset{N}{\Longrightarrow}} q_{(m)} \in F$. We prove by induction on $r = 0, \ldots, m$ that A_α can reach $q_{(r)}$ while consuming $z_r = g(v_0)\, g(v_{[1]})\, g(v_1) \cdots g(v_{[r]})\, g(v_r)$ from the input and pushing $y_r = v_0 n_1 v_1 \cdots n_r v_r$ to the stack. Conversely, we suppose a calculation in A_α, denoted by $C_{(1)} = (q_0, w\dashv, \#Z_0 \restriction \$) \vdash \cdots \vdash C_{(r)} \vdash \cdots \vdash C_{(m)} = (p_m, \dashv, \#\beta_m\$)$, where $p_m \in F$ and $C_{(r)} = (p_r, w_r\dashv, \#\beta_r\$)$ for each $r \in \{1, \ldots, m\}$. By induction on $r = 1, \ldots, m$, we extract an underlying transition $q_0 \overset{\gamma_r}{\underset{N}{\Longrightarrow}} p_r$ step by step while maintaining the invariants $\gamma_r \in (\Sigma \uplus B_k \uplus [k])^*$ and $w = \mathcal{D}_k(\gamma_r)\, w_r$, as long as $p_r \in Q_N$. □

Hereafter, we shall refine this proof sketch. We first state two lemmas used to write our full proof of Theorem 18. Let $\vdash_{(n)}$ denote the subrelation of $\vdash$ derived from the rule numbered (n). The following lemma is immediate from the definition of $\vdash_{(n)}$.

Lemma 19. *For all $q, q' \in Q_N$, $w, w' \in \Sigma^*$ and $\gamma, \gamma' \in \Gamma^*$,*

(i) (a) *for each $a \in \Sigma$, $(q, aw\dashv, \#Z_0\gamma \restriction \$) \vdash_{(1)} (q', w\dashv, \#Z_0\gamma a \restriction \$)$ if $q \xrightarrow[N]{a} q'$,*

(b) *$\exists a \in \Sigma.\, q \xrightarrow[N]{a} q' \wedge w = aw' \wedge \beta = Z_0\gamma a \restriction$ if $(q, w\dashv, \#Z_0\gamma \restriction \$) \vdash_{(1)} (q', w'\dashv, \#\beta\$)$,*

(ii) (a) *for each $b \in B_k$, $(q, w\dashv, \#Z_0\gamma \restriction \$) \vdash_{(2)} (q', w\dashv, \#Z_0\gamma b \restriction \$)$ if $q \xrightarrow[N]{b} q'$,*

(b) *$\exists b \in B_k.\, q \xrightarrow[N]{b} q' \wedge w = w' \wedge \beta = Z_0\gamma b \restriction$ if $(q, w\dashv, \#Z_0\gamma \restriction \$) \vdash_{(2)} (q', w'\dashv, \#\beta\$)$.*

In particular, letting $\vdash_{(1),(2)} = \vdash_{(1)} \uplus \vdash_{(2)}$, we obtain the following statement by repeating (i)(a) and (ii)(a) zero or more times: For all $v \in (\Sigma \uplus B_k)^*$, $(q, g(v)\, w\dashv, \#Z_0\gamma \restriction \$) \vdash^*_{(1),(2)} (q', w\dashv, \#Z_0\gamma v \restriction \$)$ if $q \overset{v}{\underset{N}{\Longrightarrow}} q'$.

Lemma 20. *Suppose that $q \xrightarrow[N]{i} q'$, and γi is matching. Let $m = \mathrm{cnt}\,(\gamma i)$. For all $p \in Q_N$, $w, w' \in \Sigma^*$ and $\beta \in (\Gamma \uplus \{\mathbb{e}, \$, \restriction\})^*$, the following (a) and (b) are equivalent:*

(a) *$p = q'$, $w = g((\gamma i)_{[m]})\, w'$, and $\beta = Z_0\gamma i \restriction$.*

(b) *$(q, w\dashv, \#Z_0\gamma \restriction \$) \vdash_{(3)} (W_{q'}, w\dashv, \#Z_0\gamma i \restriction \$) \vdash \cdots \vdash (p, w'\dashv, \#\beta\$)$, where no ID with a state in Q_N appears in the calculation $\cdots$.*

For the proof of Lemma 20, we recall the notations $v_{[r]}$ and $v^{(r)}$ informally (see Section 3 for the formal definitions of $v_{[r]}$ and $v^{(r)}$). Let k be a positive integer and $v = v_0 n_1 v_1 \ldots n_m v_m$ ($m = \mathrm{cnt}\, v$) a matching string over $\Sigma \uplus B_k \uplus [k]$. For each $r = 1, 2, \ldots$, the notation $v_{[r]}$ denotes the string which $\mathcal{D}_k$ scans at the r^{th} number n_r and $v^{(r)}$ the string immediately after the r^{th} replacement (also we let $v^{(0)} = v$). For example, in the case of $v = [_1 a\, [_2 b]_2\, 2\,]_1\, 1$, $\mathcal{D}_k$ processes v as $v^{(0)} = [_1 a\, \underline{[_2 b]_2}\, 2\,]_1\, 1 \to v^{(1)} = \underline{[_1 a\, [_2 b]_2\, b\,]_1}\, 1 \to v^{(2)} = [_1 a\, [_2 b]_2\, b\,]_1\, abb \to abbabb$, and therefore

$v_{[1]} = b$ and $v_{[2]} = a[_2 b]_2 b$. We can easily prove the following relationship between two notations $v_{[r]}$ and $v^{(r)}$ (see Lemma 10 for the formal proof): For each $r \in \{0, 1, \ldots, m\}$,

$$v^{(r)} = v_0\, g(v_{[1]})\, v_1 \cdots g(v_{[r]})\, v_r n_{r+1} v_{r+1} \cdots n_m v_m.$$

We continue preparing some more notations for the proof. Let $\Sigma^*_\perp \triangleq \Sigma^* \uplus \{\perp\}$ and for every $w \in \Sigma^*_\perp$ and $s \in \Sigma^*$, let w/s denote the string w but with the suffix s erased if w ends with s, and otherwise $\perp$. To use this notation, we expand the set of all IDs $I = Q \times \Sigma^*\{\dashv\} \times \{\#\}\,(\Gamma \uplus \{\not\subset, \$, \restriction\})^*\,\{\$\}$ to $I_\perp \triangleq Q \times \Sigma^*\{\dashv\} \uplus \{\perp\} \times \{\#\}\,(\Gamma \uplus \{\not\subset, \$, \restriction\})^*\,\{\$\}$, and we let $C(u)$ denote an ID $C = (\cdot, u, \cdots)$ and $\vdash'_{(n)}$ denote the following binary relation over $I_\perp$ (we regard $a_0 \cdots a_{-1}$ as the empty string ε):

$$\vdash'_{(n)} \triangleq \left\{ (C(u), C'(u/(a_0 \cdots a_{\overline{d}-1}))) \;\middle|\; C(a_0 \cdots a_k) \vdash_{(n)} C'(a_{\overline{d}} \cdots a_k), u \in \Sigma^*\{\dashv\} \uplus \{\perp\} \right\}.$$

We define $\vdash' \triangleq \bigcup_n \vdash'_{(n)}$. Then, $\vdash \subseteq \vdash'$ is immediate and we show that the converse partially holds, in the sense that:

Lemma 21. *For every string $w \in \Sigma^*$ and $w' \in \Sigma^*$, $C(w\dashv) \vdash' C'(w'\dashv)$ implies $C(w\dashv) \vdash C'(w'\dashv)$.*

Proof. By the definition of $\vdash'$, there is $(C(a_0 \cdots a_k), C'(a_{\overline{d}} \cdots a_k)) \in \vdash$ such that $w'\dashv = w\dashv/(a_0 \cdots a_{\overline{d}-1})$. By the definition of $\vdash$, $C(w\dashv) = C(a_0 \cdots a_{\overline{d}-1} w'\dashv) \vdash C'(w'\dashv)$ holds. □

Definition 22. Given $C, C' \in I_\perp$, we write $C \models_{(n)} C'$ if $\forall j, C''.C \vdash'_{(j)} C'' \iff j = n \wedge C'' = C'$. We often omit the subscript (n) and simply write $C \models C'$. Note that $C \models C'$ implies not only $C \vdash' C'$ but also determinism: $\forall C'' \in I_\perp. C \vdash' C'' \Longrightarrow C' = C''$.

Lemma 23. *Suppose that $\gamma \in (\Sigma \uplus B_k \uplus [k])^*$, $i \in [k]$, $w \in \Sigma^*$, $\beta \in (\Gamma \uplus \{\not\subset, \$\})^*$ and $p \in Q_N \uplus \{e_i \mid i \in [k]\}$. Let $m = \mathrm{cnt}\,(\gamma i)\,(\geq 1)$. If γi is matching,*

$$(c_i, w\dashv, \#Z_0\gamma\not\subset p\restriction \$i\beta\$) \models \cdots \models (r_i, w\dashv/g((\gamma i)_{[m]}), \#Z_0\gamma\not\subset p\restriction \$i\beta\$)$$

holds, where no ID with a state in Q_N appears in the calculation $\cdots$.

Proof. In this proof, we sometimes write the stack representation $\# \cdots Z\restriction \cdots \$$ as $\# \cdots \restriction Z \cdots \$$ with the head-reversed arrow $\restriction$. First, if $\gamma \not\ni [_i$, it holds that

$$\begin{aligned}(c_i, w\dashv, \#Z_0\gamma\not\subset p\restriction \$i\beta\$) &\models^* (c_i, w\dashv, \#Z_0\restriction \gamma\not\subset p\$i\beta\$)\\ &\models\ (r_i, w\dashv, \#Z_0\restriction \gamma\not\subset p\$i\beta\$) \models^* (r_i, w\dashv, \#Z_0\gamma\not\subset p\restriction \$i\beta\$),\end{aligned}$$

and by $(\gamma i)_{[m]} = \varepsilon$, we have $w = w/g((\gamma i)_{[m]})$, as required. Henceforth, we assume that $\gamma \ni [_i$ and the decomposition $\gamma = \gamma_0[_i\gamma_1$ ($\gamma_1 \not\ni [_i$). Moreover, we can further decompose $\gamma_1 = \gamma_2]_i\gamma_3$ ($\gamma_2 \not\ni [_i,]_i$, $\gamma_3 \not\ni [_i$) because γi is matching. We prove by induction on m.

Case $m = 1$: By $\mathrm{cnt}\,\gamma = 0$, $\gamma_2 \in (\Sigma \uplus B_k)^*$ follows. Letting $w' \triangleq w/g(\gamma_2)$, we have

$$\begin{aligned}&(c_i, w\dashv, \#Z_0\gamma_0[_i\gamma_1\not\subset p\restriction \$i\beta\$) \models^* (c_i, w\dashv, \#Z_0\gamma_0[_i\restriction \gamma_1\not\subset p\$i\beta\$) \models (e_i, w\dashv, \#Z_0\gamma_0[_i\restriction \gamma_1\not\subset p\$i\beta\$)\\ &\models^* (e_i, w'\dashv, \#Z_0\gamma_0[_i\gamma_2]_i\restriction \gamma_3\not\subset p\$i\beta\$) \models (r_i, w'\dashv, \#Z_0\gamma_0[_i\gamma_2]_i\restriction \gamma_3\not\subset p\$i\beta\$)\\ &\models^* (r_i, w'\dashv, \#Z_0\gamma_0[_i\gamma_2]_i\gamma_3\not\subset p\restriction \$i\beta\$).\end{aligned}$$

Therefore, the claim holds since no ID with a state in Q_N appears in this calculation and $\gamma_2 = (\gamma i)_{[m]}$ follows from $(\gamma i)^{(0)} = \gamma i = \gamma_0[_i\gamma_2]_i\gamma_3 i$, $\gamma_3 \not\ni [_i$.

Case $\{1, \ldots, m\} \Longrightarrow m + 1$: Let $m_0 \triangleq \mathrm{cnt}\,\gamma_0$ and $l \triangleq \mathrm{cnt}\,\gamma_2\,(\geq 0)$. Now, $m_0 + l \leq m = \mathrm{cnt}\,\gamma$ holds and we write $\gamma_2 = \lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l$. We also define $\eta_r \triangleq \gamma_0[_i\lambda_0 n_{m_0+1} \cdots \lambda_{r-1} n_{m_0+r}$ for each $r \in \{1, \ldots, l\}$. By η_r being a prefix of γi and Lemma 7, η_r is matching and $(\eta_r)_{[m_0+r]} = (\gamma i)_{[m_0+r]}$, $r \in \{1, \ldots, l\}$ holds. In particular, it follows that $n_{m_0+r} \neq i$ for every r (if there is r such that $n_{m_0+r} = i$, $\gamma_2 \supseteq \lambda_0 n_{m_0+1} \cdots \lambda_{r-1} \ni]_i$ holds but this contradicts $\gamma_2 \not\ni]_i$). Thus, letting $u_0 \triangleq w\dashv$, $u'_r \triangleq u_{r-1}/g(\lambda_{r-1})$, $u_r \triangleq u'_r/g((\eta_r)_{[m_0+r]})$ and $u' = u_l/g(\lambda_l)$, we have

$$(c_i, w\dashv, \#Z_0\gamma\not\subset p\restriction \$i\beta\$)$$

$$
\begin{aligned}
&\models^* (c_i, w\dashv, \#Z_0\gamma_0[_i \restriction \lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\models\ (e_i, u_0, \#Z_0\gamma_0[_i \uparrow \lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\models^* (e_i, u_1', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1} \restriction\ \lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\models\ (c_{n_{m_0+1}}, u_1', \#Z_0\gamma_0[_i\lambda_0 ¢ r_i \restriction \$n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\models^* (r_{n_{m_0+1}}, u_1, \#Z_0\gamma_0[_i\lambda_0 ¢ r_i \restriction \$n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by } \eta_1 \text{ being matching and induction hypothesis)}\\
&\models\ (E_{e_i,n_{m_0+1}}, u_1, \#Z_0\gamma_0[_i\lambda_0 ¢ \restriction \$n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\models\ (L_{e_i,n_{m_0+1}}, u_1, \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1} \restriction \lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\models\ (e_i, u_1, \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1} \uparrow \lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
\models^* \cdots \models^* &(e_i, u_l, \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l} \uparrow \lambda_l]_i\gamma_3 ¢ p\$i\beta\$)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by similar calculation and induction hypothesis)}\\
&\models^* (e_i, u', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i \restriction \gamma_3 ¢ p\$i\beta\$)\\
&\models\ (r_i, u', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i \uparrow \gamma_3 ¢ p\$i\beta\$)\\
&\models^* (r_i, u', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p \restriction \$i\beta\$),
\end{aligned}
$$

and

$$
\begin{aligned}
u' &= w\dashv/g(\lambda_0)\, g((\eta_1)_{[m_0+1]})\, g(\lambda_1)\cdots g((\eta_l)_{[m_0+l]})\, g(\lambda_l)\\
&= w\dashv/g(\lambda_0)\, g((\gamma i)_{[m_0+1]})\, g(\lambda_1)\cdots g((\gamma i)_{[m_0+l]})\, g(\lambda_l),
\end{aligned}
$$

where no ID with a state in Q_N appears in this calculation. Here, we write

$$
\gamma = \gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 = v_0 n_1 v_1 \cdots n_m v_m
$$

and decompose its substrings as

$$
v_{m_0} = \chi_0[_i\lambda_0, \quad v_{m_0+l} = \lambda_l]_i\chi_1, \text{ and } \gamma_3 = \chi_1 n_{m_0+l+1} v_{m_0+l+1}\cdots n_m v_m.
$$

Then, by Lemma 10 (or the relationship between two notations $v_{[r]}$ and $v^{(r)}$ mentioned above), we can write $(\gamma i)^{(m)}$ as

$$
v_0\cdots \underbrace{\chi_0[_i\lambda_0}_{v_{m_0}} \overbrace{g((\gamma i)_{[m_0+1]})\, v_{m_0+1}\cdots g((\gamma i)_{[m_0+l]})}^{=(\gamma i)_{[m+1]}} \underbrace{\lambda_l]_i\chi_1}_{v_{m_0+l}} \overbrace{g((\gamma i)_{[m_0+l+1]})v_{m_0+l+1}\cdots g((\gamma i)_{[m]})\, v_m}^{\gamma_3' \triangleq}\, i.
$$

That is, it holds that $\gamma_3' \triangleq \chi_1\, g((\gamma i)_{[m_0+l+1]})v_{m_0+l+1}\cdots g((\gamma i)_{[m]})\, v_m \not\equiv [_i$ by $\gamma_3 \not\equiv [_i$, and we obtain $(\gamma i)_{[m+1]} = \lambda_0\, g((\gamma i)_{[m_0+1]})\, \lambda_1 \cdots\, g((\gamma i)_{[m_0+l]})\, \lambda_l$, as shown above. Therefore, the claim holds for $m+1$ since $u' = w\dashv/g((\gamma i)_{[m+1]})$. □

Proof of Lemma 20. For an arbitrary $w \in \Sigma^*$, by Lemma 23,

$$
\begin{aligned}
&(q, w\dashv, \#Z_0\gamma \restriction \$) \vdash_{(3)} (W_{q'}, w\dashv, \#Z_0\gamma i \restriction \$) \models_{(4)} (c_i, w\dashv, \#Z_0\gamma ¢ q' \restriction \$i\$)\\
&\models^* (r_i, w\dashv/g((\gamma i)_{[m]}), \#Z_0\gamma ¢ q' \restriction \$i\$) \models_{(14)} (E_{q',i}, w\dashv/g((\gamma i)_{[m]}), \#Z_0\gamma ¢ \restriction \$i\$)\\
&\models_{(15)} (L_{q',i}, w\dashv/g((\gamma i)_{[m]}), \#Z_0\gamma i \restriction \$) \models_{(17)} (q', w\dashv/g((\gamma i)_{[m]}), \#Z_0\gamma i \restriction \$) \qquad (*)
\end{aligned}
$$

holds. Assuming (a), we can replace $\models$ in equation $(*)$ with $\vdash$ by Lemma 21 because $w/g((\gamma i)_{[m]}) = w' \in \Sigma^*$ holds, and therefore, (b) follows. Supposing (b) conversely, we have $(q, w\dashv, \#Z_0\gamma \restriction \$) \vdash_{(3)} (W_{q'}, w\dashv, \#Z_0\gamma i \restriction \$) \vdash' \cdots \vdash' (p, w'\dashv, \#\beta\$)$, where no ID with a state in Q_N appears in either this calculation or $(*)$ except in their leftmost and rightmost IDs. Therefore, their two calculations coincide by the determinism of $\models$. In particular, we obtain $p = q'$, $w'\dashv = w\dashv/g((\gamma i)_{[m]})$ and $\beta = Z_0\gamma i \restriction$ by the equality of their rightmost IDs, and thus, (a) follows because $w' \in \Sigma^*$. □

Proof of Theorem 18. Taking any $w \in L(\alpha)$, we have $v = v_0 n_1 v_1 \cdots n_m v_m \in \mathcal{R}_k(\alpha) = L(N)$ (here, $m = \mathrm{cnt}\, v$) such that $w = \mathcal{D}_k(v)$. Now, suppose that $q_0 \overset{v_0}{\underset{N}{\Longrightarrow}} q_{(0)} \overset{n_1 v_1}{\underset{N}{\Longrightarrow}} q_{(1)} \overset{n_2 v_2}{\underset{N}{\Longrightarrow}} \cdots \overset{n_m v_m}{\underset{N}{\Longrightarrow}} q_{(m)} \in F$. Letting $y_r \triangleq v_0 n_1 v_1 \cdots n_r v_r$ and $z_r \triangleq g(v_0)\, g(v_{[1]})\, g(v_1) \cdots g(v_{[r]})\, g(v_r)$ for each $r \in \{0, 1, \ldots, m\}$, we show by induction on r that for any $w' \in \Sigma^*$, $(q_0, z_r w' \dashv, \#Z_0 \restriction \$) \vdash^* (q_{(r)}, w' \dashv, \#Z_0 y_r \restriction \$)$ holds.

Case $r = 0$: It holds that $q_0 \overset{v_0}{\underset{N}{\Longrightarrow}} q_{(0)}$ and $y_0 = v_0 \in (\Sigma \uplus B_k)^*$. Letting $\gamma = \varepsilon$ in Lemma 19, we obtain $(q_0, g(v_0) w' \dashv, \#Z_0 \restriction \$) \vdash^* (q_{(0)}, w' \dashv, \#Z_0 y_0 \restriction \$)$, as required.

Case $r \Rightarrow r+1$: By the induction hypothesis, it holds that $(q_0, z_{r+1} w' \dashv, \#Z_0 \restriction \$) \vdash^* (q_{(r)}, g(v_{[r+1]})\, g(v_{r+1})\, w' \dashv, \#Z_0 y_r \restriction \$)$. By Lemma 8, v is matching. Hence, v's prefix $y_r n_{r+1}$ is also matching by Lemma 7. Let $q_{(r)} \overset{n_{r+1} v_{r+1}}{\underset{N}{\Longrightarrow}} q_{(r+1)}$ be $q_{(r)} \overset{n_{r+1}}{\underset{N}{\Longrightarrow}} q' \overset{v_{r+1}}{\underset{N}{\Longrightarrow}} q_{(r+1)}$. Because $\mathrm{cnt}\,(y_r n_{r+1}) = r+1$, the following calculation holds by Lemma 20 (a) $\Longrightarrow$ (b):

$$
\begin{aligned}
&(q_{(r)}, g((y_r n_{r+1})_{[r+1]})\, g(v_{r+1})\, w' \dashv, \#Z_0 y_r \restriction \$) \\
&\vdash_{(3)} (W_{q'}, g((y_r n_{r+1})_{[r+1]})\, g(v_{r+1})\, w' \dashv, \#Z_0 y_r n_{r+1} \restriction \$) \\
&\vdash^* (q', g(v_{r+1})\, w' \dashv, \#Z_0 y_r n_{r+1} \restriction \$) \vdash^* (q_{(r+1)}, w' \dashv, \#Z_0 y_r n_{r+1} v_{r+1} \restriction \$).
\end{aligned}
$$

By Lemma 7, it holds that $(y_r n_{r+1})_{[r+1]} = v_{[r+1]}$ and $y_r n_{r+1} v_{r+1} = y_{r+1}$, as required. In particular, letting $r = m$ and $w' = \varepsilon$ in the claim, we obtain $(q_0, w \dashv, \#Z_0 \restriction \$) \vdash^* (q_{(m)}, \dashv, \#Z_0 y_m \restriction \$)$ because $w = \mathcal{D}_k(v) = z_m$ by Lemma 6. Therefore, $w \in L(A_\alpha)$ holds since $q_{(m)} \in F$.

Conversely, take any $w \in L(A_\alpha)$. There exist m and an ID $C_{(r)} = (p_r, w_r \dashv, \#\beta_r \$)$ for each $r \in \{1, \ldots, m\}$ such that $C_{(1)} = (q_0, w \dashv, \#Z_0 \restriction \$) \vdash \cdots \vdash C_{(r)} \vdash \cdots \vdash C_{(m)} = (p_m, \dashv, \#\beta_m \$)$, where $p_m \in F$. We show by induction on r that for each $r = 1, \ldots, m$, the following claim holds: if $p_r \in Q_N$, there is γ_r such that $C_{(r)} = (p_r, w_r \dashv, \#Z_0 \gamma_r \restriction \$)$, (a) $\gamma_r \in (\Sigma \uplus B_k \uplus [k])^*$, (b) $w = \mathcal{D}_k(\gamma_r)\, w_r$ and (c) $q_0 \overset{\gamma_r}{\underset{N}{\Longrightarrow}} p_r$.

Case $r = 1$: It holds that $p_1 = q_0 \in Q_N$ and $C_{(1)} = (q_0, w \dashv, \#Z_0 \restriction \$)$. Letting $p_1 = q_0$, $w_1 = w$ and $\gamma_1 = \varepsilon$, we have (a) $\gamma_1 = \varepsilon \in (\Sigma \uplus B_k \uplus [k])^*$, (b) $w = \mathcal{D}_k(\gamma_1)\, w_1$ and (c) $q_0 \overset{\gamma_1}{\underset{N}{\Longrightarrow}} p_1 = q_0$, as required.

Case $\{1, \ldots, r\} \Rightarrow r+1$: Suppose that $p_{r+1} \in Q_N$. We can define j as the maximum of the set $\left\{ 1 \le j \le r \,\middle|\, p_j \in Q_N \right\}$ since $p_1 \in Q_N$. Rules that can be applied to $C_{(j)}$ are limited to (1), (2) and (3) because $p_j \in Q_N$.

In the case of (1), $j = r$ holds because $p_{j+1} \in Q_N$. By $p_r \in Q_N$ and the induction hypothesis, we have $C_{(r)} = (p_r, w_r \dashv, \#Z_0 \gamma_r \restriction \$)$, and p_r, w_r and γ_r satisfy conditions (a), (b) and (c). Hence, by Lemma 19 (i)(b), there is $a \in \Sigma$ such that $p_r \overset{a}{\underset{N}{\longrightarrow}} p_{r+1}$, $w_r = a w_{r+1}$ and $\beta_{r+1} = Z_0 \gamma_r a \restriction$. Now, we let $\gamma_{r+1} = \gamma_r a$. Since $\beta_{r+1} = Z_0 \gamma_{r+1} \restriction$, (a) $\gamma_{r+1} = \gamma_r a \in (\Sigma \uplus B_k \uplus [k])^*$ because $\gamma_r \in (\Sigma \uplus B_k \uplus [k])^*$, (b) $\mathcal{D}_k(\gamma_{r+1}) w_{r+1} = \mathcal{D}_k(\gamma_r) a w_{r+1} = \mathcal{D}_k(\gamma_r) w_r = w$, and (c) $q_0 \overset{\gamma_{r+1}}{\underset{N}{\Longrightarrow}} p_{r+1}$ because $q_0 \overset{\gamma_r}{\underset{N}{\Longrightarrow}} p_r \overset{a}{\underset{N}{\longrightarrow}} p_{r+1}$. The case of (2) follows similarly as the case of (1) with Lemma 19 (ii)(b). In the case of (3), there is $q' \in Q_N$ such that $p_j \overset{i}{\underset{N}{\longrightarrow}} q'$. By $p_j \in Q_N$ and the induction hypothesis, we have $C_{(j)} = (p_j, w_j \dashv, \#Z_0 \gamma_j \restriction \$)$, and p_j, w_j and γ_j satisfy the conditions (a), (b) and (c). Because $q_0 \overset{\gamma_j}{\underset{N}{\Longrightarrow}} p_j \overset{i}{\underset{N}{\longrightarrow}} q'$, $\gamma_j i$ is matching by Corollary 17. Besides, it holds that $p_{r+1} \in Q_N$ and $(p_j, w_j \dashv, \#Z_0 \gamma_j \restriction \$) \vdash_{(3)} (W_{q'}, w_j \dashv, \#Z_0 \gamma_j i \restriction \$) \vdash \cdots \vdash C_{(r+1)}$, where no ID with a state in Q_N appears in the calculation $\cdots$. Hence, by Lemma 20 (b) $\Longrightarrow$ (a), we have $p_{r+1} = q'$, $w_j = g((\gamma_j i)_{[m]}) w_{r+1}$, and $\beta_{r+1} = Z_0 \gamma_j i \restriction$. Now, we let $\gamma_{r+1} = \gamma_j i$. Since $\beta_{r+1} = Z_0 \gamma_{r+1} \restriction$, (a) $\gamma_{r+1} = \gamma_r i \in (\Sigma \uplus B_k \uplus [k])^*$ because $\gamma_r \in (\Sigma \uplus B_k \uplus [k])^*$, (b) $\mathcal{D}_k(\gamma_{r+1}) w_{r+1} = \mathcal{D}_k(\gamma_j)\, g((\gamma_j i)_{[m]})\, w_{r+1} = \mathcal{D}_k(\gamma_j) w_j = w$, and (c) $q_0 \overset{\gamma_{r+1}}{\underset{N}{\Longrightarrow}} p_{r+1}$ because $q_0 \overset{\gamma_j}{\underset{N}{\Longrightarrow}} p_j \overset{i}{\underset{N}{\longrightarrow}} p_{r+1}$.

Therefore, the claim holds for the case of $r+1$. In particular, letting $r = m$ in the claim, we have $C_{(m)} = (p_m, w_m \dashv, \#Z_0 \gamma_m \restriction \$)$, and p_m, w_m and γ_m satisfy (a), (b) and (c) (note that $p_m \in F \subseteq Q_N$). Because $w_m = \varepsilon$, it holds that $w = \mathcal{D}_k(\gamma_m)$ and that $q_0 \overset{\gamma_m}{\underset{N}{\Longrightarrow}} p_m \in F$, or $\gamma_m \in L(N) = \mathcal{R}_k(\alpha)$. Therefore, we have $w \in \mathcal{D}_k(\mathcal{R}_k(\alpha)) = L(\alpha)$. □

Corollary 24. *Every rewb describes an indexed language, but not vice versa.*

Proof. The first half follows by Theorem 18 since 1N Nested-SA and indexed grammars are equivalent [2]. The second half also follows since the class of CSLs is a subclass of indexed languages [1], and the language class of rewbs and that of CFLs are incomparable [4]. □

4.2. The case without a captured reference

In the case of a rewb α without a captured reference (that is, one in which no reference $\backslash i$ appears as a subexpression of an expression of the form $({}_j \dots)_j$), we can transform A_α into an NESA A''_α recognizing $L(\alpha)$, i.e., one that neither uses substacks nor pops its stack. First, we transform A_α to a Nested-SA without substacks (i.e., SA) A'_α. Inspecting how substacks are used in A_α, we can drop rules (12) and (16) in A'_α because there is no captured reference in α. We also remove the uses of substacks from rules (3) and (4), which correspond to calling, and rules (14), (15) and (17), which correspond to returning. Namely, while A_α, upon a call, stores the substack $\not\in q'\$$ that consists of just the state q' where the control should return, A'_α simply pushes q' to the stack top. That is, we remove (4), (15) and (17), and change (3) and (14) to the following (3') and (14'), respectively:

$$\text{(3')}\ \delta_N(q,i) \ni q' \Longrightarrow \delta(q,c,Z\$) \ni (c_i, \mathrm{S}, Ziq'\$), \qquad \text{(14')}\ \delta(r_i,c,q\$) = \{(q,\mathrm{S},\$)\}.$$

Furthermore, we transform A'_α to an SA without stack popping (i.e., NESA) A''_α. Observe that A'_α pops only when returning via (14') and popping a state that was pushed in a preceding call. Thus, A''_α, rather than popping q', leaves it on the stack, and has the modes c_i, e_i and r_i skip all state symbols on the stack except the ones at the top. Here, we only need to modify e_i since A_α already skips them at c_i and r_i (rules (6) and (13)). In short, we add the new rule (9*) and change (14') to (14"), as follows:

$$\text{(9*)}\ \delta(e_i,c,q) = \{(e_i,\mathrm{S},\mathrm{R})\}, \qquad \text{(14")}\ \delta(r_i,c,q\$) = \{(q,\mathrm{S},q\$)\}.$$

This NESA A''_α whose transition function consists of the rules (1),(2),(3'),(5)–(9),(9*),(10), (11), (13) and (14") recognizes $L(\alpha)$. Therefore,

Corollary 25. *Every rewb without a captured reference describes a nonerasing stack language, but not vice versa.*

For the latter part of Corollary 25, we can take the language $T \triangleq \{a^n b^n \mid n \in \mathbb{N}\}$ that can be described by an NESA but not by any rewb [4]. Here, we show that T is described by the NESA A given in Figure 1.

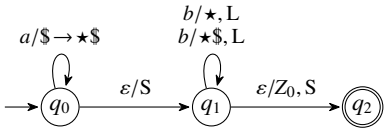


Figure 1: An NESA that recognizes $T = \{a^n b^n \mid n \in \mathbb{N}\}$ (see Notation 15 for the notations)

The set of stack symbols Γ is $\{Z_0, \star\}$, where $\star$ is a distinguished character. Intuitively, A first pushes $\star$ while consuming the input a to count the occurrences of a and leaves q_0 for q_1 nondeterministically. At q_1, A moves its stack pointer leftward while consuming the input b, and leaves there for the accepting state q_2 if and only if the stack pointer reaches the bottom of the stack. Finally, A halts at q_2. Here is the proof of $L(A) = T$:

Proof. The inclusion $T \subseteq L(A)$ follows from the following calculation for any $n \in \mathbb{N}$:

$$\begin{aligned}(q_0, a^n b^n \dashv, \#Z_0 \rceil \$) &\vdash^n (q_0, b^n \dashv, \#Z_0 \star^n \rceil \$)\\ &\vdash (q_1, b^n \dashv, \#Z_0 \star^n \rceil \$) \vdash^n (q_1, \dashv, \#Z_0 \rceil \star^n \$) \vdash (q_2, \dashv, \#Z_0 \rceil \star^n \$).\end{aligned}$$

Conversely, take any $w \in L(A)$. By the construction of A, we can assume that $w = a^n b^m$ (where $n, m \in \mathbb{N}$) and its accepting calculation is in the following form for some $l \in \mathbb{N}$:

$$\begin{aligned}(q_0, a^n b^m \dashv, \#Z_0 \rceil \$) &\vdash^l (q_0, a^{n-l} b^m \dashv, \#Z_0 \star^l \rceil \$)\\ &\vdash (q_1, a^{n-l} b^m \dashv, \#Z_0 \star^l \rceil \$) \vdash^* (q_1, \dashv, \#Z_0 \rceil \star^l \$) \vdash (q_2, \dashv, \#Z_0 \rceil \star^l \$).\end{aligned}$$

In this calculation, the steps $\vdash^*$ indicate $n = l = m$. □

Notice that there exists a rewb *with* a captured reference that describes a nonerasing stack language. The rewb $({}_1a)_1({}_2\backslash 1)_2\backslash 2$ is a simple counterexample. In addition, as shown later in Section 7, NESA can recognize nontrivial language (hierarchy) with a captured reference such as Larsen's hierarchy [17].

5. A rewb that describes a non-stack language

We just showed that every rewb describes an indexed language and in particular every rewb without a captured reference describes a nonerasing stack language. So, a natural question is whether every rewb describes a (nonerasing) stack language. We show that the answer is *no*. That is, there is a rewb that describes a non-stack language.

Ogden has proposed a pumping lemma for stack languages and shown that the language $\left\{ a^{n^3} \;\middle|\; n \in \mathbb{N} \right\}$ is a non-stack language as an application (see [20], Theorem 2). A key point in the proof is that the exponential n^3 of a is a cubic polynomial, and we can show that for every cubic polynomial $f : \mathbb{N} \to \mathbb{N}$, the language $\left\{ a^{f(n)} \;\middle|\; n \in \mathbb{N} \right\}$ is also a non-stack language by the same proof. Thus, a rewb that describes a language in this form is a counterexample. We borrow the technique in [10] (Example 1) which shows that the rewb $\alpha = ((_1\backslash 2)_1 (_2\backslash 1 a)_2)^*$ describes $L(\alpha) = \left\{ a^{n^2} \;\middle|\; n \in \mathbb{N} \right\}$. This follows since $\mathcal{D}_k(([_1 2]_1\, [_2 1\, a]_2)^n) = a^{n^2}$ holds by recording the iteration count of the Kleene star, n, in the capture $(_2\,)_2$ as a^n, and extending the length by $2n + 1$, as shown below:

$$\begin{aligned}
&\mathcal{D}_k(([_1 2]_1\, [_2 1\, a]_2)^{n+1}) = \mathcal{D}_k(([_1 2]_1\, [_2 1\, a]_2)^n [_1 2]_1\, [_2 1\, a]_2) = \mathcal{D}_k(\cdots \underline{[_2 a^n]_2}\, [_1 2]_1\, [_2 1\, a]_2) \\
&= \mathcal{D}_k(\cdots [_2 a^n]_2\, \underline{[_1 a^n]_1}\, [_2 1\, a]_2) = \mathcal{D}_k(\underline{\cdots [_2 a^n]_2}\, [_1 a^n]_1\, [_2 a^{n+1}]_2) = \underline{\underline{a^{n^2}}}\, a^{2n+1} = a^{(n+1)^2}.
\end{aligned}$$

Theorem 26. *There exists a rewb that describes a non-stack language.*

Proof. The rewb $\alpha_0 \triangleq ((_1\backslash 4\, a)_1\, (_2\backslash 3)_2\, (_3\backslash 2\, a)_3\, (_4\backslash 1\backslash 3)_4)^*$ describes $L_0 \triangleq \left\{ a^{n(n+7)(2n+1)/6} \;\middle|\; n \in \mathbb{N} \right\}$ and extends the length by a quadratic in n instead. Let $k = 4$. It is easy to see that $\mathcal{R}_k(\alpha_0) = \{([_1 4\, a]_1\, [_2 3]_2\, [_3 2\, a]_3\, [_4 1\, 3]_4)^n \mid n \in \mathbb{N}\}$. Let b_n denote $([_1 4\, a]_1\, [_2 3]_2\, [_3 2\, a]_3\, [_4 1\, 3]_4)^n$ (therefore $L(\alpha_0) = \{\mathcal{D}_k(b_n) \mid n \in \mathbb{N}\}$). We show by induction that $\mathcal{D}_k(b_n) = \mathcal{D}_k(c_1 \cdots c_n)$ where $c_n \triangleq [_1 a^{n(n+1)/2}]_1\, [_2 a^{n-1}]_2\, [_3 a^n]_3\, [_4 a^{n(n+3)/2}]_4$ for every $n \geq 1$. First, when $n = 1$,

$$\begin{aligned}
\mathcal{D}_k(b_1) &= \mathcal{D}_k([_1 4\, a]_1\, [_2 3]_2\, [_3 2\, a]_3\, [_4 1\, 3]_4) = \mathcal{D}_k([_1 a]_1\, [_2 3]_2\, [_3 2\, a]_3\, [_4 1\, 3]_4) \\
&= \mathcal{D}_k([_1 a]_1\, \underline{[_2]_2}\, [_3 2\, a]_3\, [_4 1\, 3]_4) = \mathcal{D}_k(\underline{[_1 a]_1}\, [_2]_2\, [_3 a]_3\, [_4 1\, 3]_4) \\
&= \mathcal{D}_k([_1 a]_1\, [_2]_2\, \underline{[_3 a]_3}\, [_4 a\, 3]_4) = \mathcal{D}_k([_1 a]_1\, [_2]_2\, [_3 a]_3\, [_4 a\, a]_4) = \mathcal{D}_k(c_1).
\end{aligned}$$

Next, in the case of $n + 1$,

$$\begin{aligned}
\mathcal{D}_k(b_{n+1}) &= \mathcal{D}_k(b_n\, [_1 4\, a]_1\, [_2 3]_2\, [_3 2\, a]_3\, [_4 1\, 3]_4) \\
&= \mathcal{D}_k(c_1 \cdots c_n [_1 4\, a]_1\, [_2 3]_2\, [_3 2\, a]_3\, [_4 1\, 3]_4) \\
&= \mathcal{D}_k(c_1 \cdots c_n [_1 a^{n(n+3)/2}\, a]_1\, [_2 a^n]_2\, [_3 a^n\, a]_3\, [_4 a^{n(n+3)/2}\, a\, a^n\, a]_4) \\
&= \mathcal{D}_k(c_1 \cdots c_n [_1 a^{(n+1)(n+2)/2}]_1\, [_2 a^n]_2\, [_3 a^{n+1}]_3\, [_4 a^{(n+1)(n+4)/2}]_4) \\
&= \mathcal{D}_k(c_1 \cdots c_n c_{n+1}).
\end{aligned}$$

$$\therefore |\mathcal{D}_k(b_n)| = \sum_{i=1}^{n} |g(c_i)| = \sum_{i=1}^{n} (i^2 + 4i - 1) = \frac{n(n+7)(2n+1)}{6}. \quad \text{(also for } n = 0\text{)}$$

Therefore, we have $L(\alpha_0) = L_0$.

□

From this theorem and Corollary 25, this rewb α_0 needs a captured reference, in the sense that:

Corollary 27. *There exists a rewb that describes a language that no rewb without a captured reference can describe.*

The rewb α_0 consists of mutual references by a Kleene star and the fact that it describes L_0 deeply depends on the formalization of rewbs that we have adopted.[7] However, we can show that, even with only more mundane uses of

[7] For instance, if we adopt the semantics of rewbs given in the ECMAScript 2023 specification [9], α_0 would describe a different language because that semantics processes a Kleene star expression α^* by iterating the matching of the input string against α and "resets" the strings captured within α to ϵ before each iteration.

backreferences that do not involve mutual references, rewbs can go beyond the class of stack languages. In addition, we illustrate that backreferences can be used to easily cross over the gaps between the nonerasing stack, stack, and nested stack (i.e., indexed) languages. Berglund and van der Merwe formalized mutual references on a rewb with the name *circular* [4]:

Definition 28. Let α be a rewb. We define the relationship $\to_\alpha$ as $j \to_\alpha i$ means the rewb α has a subexpression in the form $({}_i \cdots \backslash j \cdots)_i$ and $\to_\alpha^+$ as the transitive closure. The rewb α is called *circular* if there is a number i with $i \to_\alpha^+ i$.

Definition 29. Let α_1, α_2 and α_3 be the following non-circular rewbs, and L_1, L_2 and L_3 denote their languages:

$$\begin{aligned}
&\alpha_1 \triangleq ({}_1a^*)_1\, c \quad (\backslash 1\#)^*, && L_1 \triangleq L(\alpha_1) = \{\, wc(w\#)^n \mid w \in \{a\}^*, n \in \mathbb{N} \,\},\\
&\alpha_2 \triangleq ({}_1a^*)_1\, c\, ({}_2(\backslash 1\#)^*)_2\, c\, \backslash 2, && L_2 \triangleq L(\alpha_2) = \{\, wc(w\#)^n c(w\#)^n \mid w \in \{a\}^*, n \in \mathbb{N} \,\},\\
&\alpha_3 \triangleq ({}_1a^*)_1\, c\, ({}_2(\backslash 1\#)^*)_2\, c\, \backslash 2\, c\, \backslash 2, && L_3 \triangleq L(\alpha_3) = \{\, wc(w\#)^n c(w\#)^n c(w\#)^n \mid w \in \{a\}^*, n \in \mathbb{N} \,\}.
\end{aligned}$$

Lemma 30. *L_1 is nonerasing stack, L_2 is stack but not nonerasing stack, and L_3 is non-stack.*

The proof is coming later. Observe that the lemma portrays a surprising power of backreferences: a slight modification of the expressions by using a simple backreference to copy a substring crosses the borders between the nonerasing stack, stack, and nested stack (i.e., indexed) languages. We obtain the following three corollaries. The first and second corollaries are stronger versions of Theorem 26 and Corollary 27, and the third corollary claims the existence of a rewb language that is located between the stack and nonerasing stack languages:

Corollary 31. *There exists a non-circular rewb that describes a non-stack language.*

Corollary 32. *There exists a non-circular rewb that describes a language that no rewb without a captured reference can describe.*

Corollary 33. *There exists a non-circular rewb that describes a stack language but not a nonerasing stack language.*

In the rest of the section, we shall prove Lemma 30. The first claim follows from Corollary 25 because α_1 is free of captured references. For the other claims, we need Ogden's pumping lemmas for the stack languages and the nonerasing stack languages [20]:[8]

Theorem 34. *For an arbitrary (one-way) SA A, there exists some integer k satisfying the following conditions: for an arbitrary word $\xi_0 \in L(A)$ with the length no less than k, there exist strings μ, ν and ρ_i, σ_i, τ_i $(i \geq 0)$ that satisfy the following conditions:*

(i) $\xi_0 = \mu\rho_0\sigma_0\tau_0\nu$,

(ii) *for each $j > 0$, it holds that $\xi_j = \mu\,\rho_0 \cdots \rho_j\,\sigma_j\tau_j \cdots \tau_0\,\nu \in L(A)$,*

(iii) *there are integers $0 < m < n$, $p > 0$ such that ρ_i, τ_i, σ_i are of the following forms:*[9]

$$\begin{aligned}
\rho_0 &= \theta_0 && \delta_1 && \theta_1 && \cdots && \delta_{m-1} && \theta_{m-1}\\
\rho_{i+1} &= \alpha_0\beta_1^i\gamma_1 && \delta_1 && \gamma_2\beta_2^i\alpha_1\beta_3^i\gamma_3 && \cdots && \delta_{m-1} && \gamma_{2m-2}\beta_{2m-2}^i\alpha_{m-1},\\
\tau_0 &= \theta_m && \delta_m && \theta_{m+1} && \cdots && \delta_{n-2} && \theta_{n-1}\\
\tau_{i+1} &= \alpha_m\beta_{2m-1}^i\gamma_{2m-1} && \delta_m && \gamma_{2m}\beta_{2m}^i\alpha_{m+1}\beta_{2m+1}^i\gamma_{2m+1} && \cdots && \delta_{n-2} && \gamma_{2n-4}\beta_{2n-4}^i\alpha_{n-1},\\
\sigma_0 &= \chi_0 && && \chi_1 && \cdots && && \chi_{p-1}\\
\sigma_{i+1} &= \chi_0 && \psi_1^{i+1} && \chi_1 && \cdots && \psi_{p-1}^{i+1} && \chi_{p-1},
\end{aligned}$$

where $\alpha_j, \beta_j, \gamma_j, \delta_j, \theta_j, \psi_j, \chi_j \in \Sigma^*$,

(iv) $|\rho_0|$ *(resp.* $|\tau_0|$*)* $= 0 \iff |\rho_i|$ *(resp.* $|\tau_i|$*)* $= 0$ $(i > 0)$,

[8] Our excerpts (Theorems 34 and 35) work as limited versions of the original pumping lemmas, which are more general statements.

[9] The ellipsis in ρ_{i+1} continues as follows: $\delta_2\ \gamma_4\beta_4^i\alpha_2\beta_5^i\gamma_5\ \delta_3 \cdots \delta_{m-2}\ \gamma_{2m-4}\beta_{2m-4}^i\alpha_{m-2}\beta_{2m-3}^i\gamma_{2m-3}$. As well for τ_{i+1}.

(v) $|\psi_j| > 0$ *and* $|\delta_j| > 0$,
(vi) *At least one of the following conditions holds:*
  (a) $|\rho_0| > 0$,
  (b) $|\tau_0| > 0$,
  (c) $p \geq 2$ *and at least two* $|\chi_j| > 0$,
(vii) *The following inequality holds:*

$$\sum_j |\beta_j| + \sum_j |\psi_j| < k.$$

We obtain the pumping lemma for the nonerasing stack languages by trimming τ_j and ν from this theorem:

Theorem 35. *For an arbitrary (one-way) NESA A, there exists some integer k satisfying the following conditions: for an arbitrary word $\xi_0 \in L(A)$ with the length no less than k, there exist strings μ and ρ_i, σ_i $(i \geq 0)$ that satisfy the following conditions:*

(i) $\xi_0 = \mu\rho_0\sigma_0$,
(ii) *for each $j > 0$, it holds that $\xi_j = \mu\,\rho_0 \cdots \rho_j\,\sigma_j \in L(A)$,*
(iii) *there are integers $m, p > 0$ such that ρ_i, σ_i are of the following forms:*

$$\begin{array}{rccccccc}
\rho_0 = & \theta_0 & \delta_1 & \theta_1 & \cdots & \delta_{m-1} & \theta_{m-1} \\
\rho_{i+1} = & \alpha_0\beta_1^i\gamma_1 & \delta_1 & \gamma_2\beta_2^i\alpha_1\beta_3^i\gamma_3 & \cdots & \delta_{m-1} & \gamma_{2m-2}\beta_{2m-2}^i\alpha_{m-1},
\end{array}$$

$$\begin{array}{rccccccc}
\sigma_0 = & \chi_0 & & \chi_1 & \cdots & & \chi_{p-1} \\
\sigma_{i+1} = & \chi_0 & \psi_1^{i+1} & \chi_1 & \cdots & \psi_{p-1}^{i+1} & \chi_{p-1},
\end{array}$$

where $\alpha_j, \beta_j, \gamma_j, \delta_j, \theta_j, \psi_j, \chi_j \in \Sigma^*$,
(iv) $|\rho_0|$ *(resp.* $|\tau_0|$*)* $= 0 \iff |\rho_i|$ *(resp.* $|\tau_i|$*)* $= 0$ $(i > 0)$,
(v) $|\psi_j| > 0$ *and* $|\delta_j| > 0$,
(vi) *At least one of the following conditions holds:*
  (a) $|\rho_0| > 0$,
  (b) $p \geq 2$ *and at least two* $|\chi_j| > 0$,
(vii) *The following inequality holds:*

$$\sum_j |\beta_j| + \sum_j |\psi_j| < k.$$

Let us explain the proof idea for L_2 and L_3. Assume that L_2 is a nonerasing stack language and fix an integer k satisfying the conditions of Theorem 35. Let $\xi_0 \triangleq wc(w\#)^k c(w\#)^k \in L_2$ where w is a string of length no less than k, for example $w = a^k$. The first key point is considering the position of the second c of ξ_0. A word in L_2 has the property that the prefix up to the second c determines the entire string. Therefore, the second c of ξ_0 is not in μ or ρ_0; if so, $\xi_1 = \xi_0$ would hold because both ξ_1 and ξ_0 would have a common prefix up to the second c, but ξ_1 must be strictly longer than ξ_0 because of the conditions of Theorem 35. Thus, it must be the case that $c \in \sigma_0$. In essence, in the case of L_2, the substring $(w\#)^n$ has all information of the word it accepts, hence there is a prefix property in the sense that any two words that have this substring in common must be equal. Moreover, because ξ_j's have a common prefix $\mu_0\rho_0$, the second c must not be in this prefix. For proving the claim that L_3 is not a stack language, we also use a suffix property of L_3 and derive that both the second and third c must be in σ_0. The second key point is that the informative substring $(w\#)^n$ will be included in σ_0. From this, we can show that σ_1 must be longer than σ_0 by at least k. However, this contradicts the last condition of Theorem 35. A similar argument will also be used in the proof for L_3.

Let s, t be strings. We write $s \preceq t$ if there is a partition $s = s_1 \cdots s_r$ and strings $u_0, \ldots, u_r$ such that $t = u_0 s_1 u_1 \cdots s_r u_r$.

Proof of Lemma 30. We only prove that L_2 is a stack but not nonerasing stack language (L_3 being non-stack is quite similar). Assume to the contrary that L_2 is a nonerasing stack language and fix k given by Theorem 35. Let $\xi_0 = wc(w\#)^k c(w\#)^k \in L_2$ where $w = a^k$. As $|\xi_0| \geq k$, there exist μ, ρ_i and σ_i $(i \geq 0)$ that satisfy the conditions of Theorem 35. By conditions (iii), (iv) and (v), the inequality $(*)$ $|\xi_0| < |\xi_1|$ easily follows.

Henceforth, we write the first and second c of $\xi_0 = \mu\rho_0\sigma_0 = wc(w\#)^k c(w\#)^k$ as c_1, c_2 respectively. Let us prove $c_2 \in \sigma_0$. Assume $c_2 \in \mu$. Now, μ is of the form $wc(w\#)^k c \cdots$. However, $\xi_1 = \mu\rho_0\rho_1\sigma_1 = wc(w\#)^k c \cdots \in L_2$ and so $\xi_1 = \xi_0$, contrary to $(*)$. Similarly, $c_2 \in \rho_0$ also does not hold. Thus, σ_0 is of the form $\eta_0 c(w\#)^k$. Because $\sigma_0 \preceq \sigma_1$, $c \in \sigma_1$. Now, we can derive $|\sigma_1| - |\sigma_0| \geq k$ as follows.

If $c_1 \in \mu\rho_0$, both ξ_0 and ξ_1 have the prefix wc in common. Because $\xi_1 \in L_2$, σ_1 can be written as $\sigma_1 = \eta_1 c(w\#)^m$ where $\eta_0 \preceq \eta_1$ and $c \notin \eta_0, \eta_1$. Inequality $(*)$ requires $m > k$. Then, $|\sigma_1| - |\sigma_0| = |\eta_1| - |\eta_0| + |(w\#)^m| - \left|(w\#)^k\right| \geq k$. Otherwise, $c_1 \in \sigma_0$ and we can write $\mu\rho_0 = u_0$, $\sigma_0 = v_0 c(w\#)^k c(w\#)^k$ and $w = u_0 v_0$. Because $\sigma_0 \preceq \sigma_1$ and $\xi_1 \in L_2$, σ_1 is of the form $v_1 c(x\#)^m c(x\#)^m$, $v_0 \preceq v_1$. Because $\xi_1 = \mu\rho_0\rho_1\sigma_1 = u_0 \cdots v_1 c(x\#)^m c(x\#)^m = xc(x\#)^m c(x\#)^m$, $|x| \geq |w|$ follows. On the other hand, since $\sigma_0 \preceq \sigma_1$, $(w\#)^k \preceq (x\#)^m$ holds and $m \geq k$ by counting #. Inequality $(*)$ ensures that either $|x| > |w|$ or $m > k$ holds. In either case, $|\sigma_1| - |\sigma_0| \geq k$.[10]

However, this contradicts the last condition of the pumping lemma. Therefore, L_2 must not be nonerasing stack. The fact that L_2 is stack follows by the SA shown in Figure 2. □

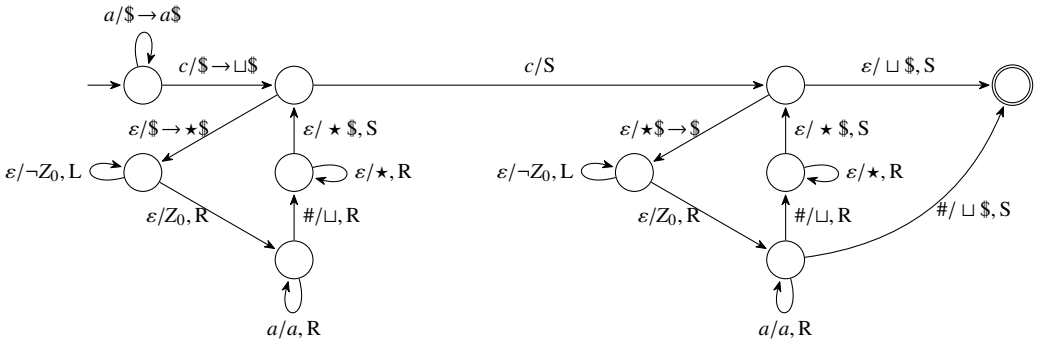


Figure 2: A stack automaton that recognizes L_2 (see Notation 15 for the notations)

6. Backreferences and lookaheads go beyond indexed languages

Chida and Terauchi formalized *rewbl*, a regular expression extended with backreferences and lookaheads, and showed that its expressive power strictly exceeds that of rewbs [8]. There are two kinds of lookaheads: positive lookaheads `?=` and negative lookaheads `?!`. In particular, a rewbl_p (resp. rewbl_n) is a rewb with positive (resp. negative) lookaheads. We write rewbl for a rewb with both positive and negative lookaheads. In what follows, for brevity, we use the *character set* notation, which regards a nonempty finite set of characters $C = \{a_1, \ldots, a_n\} \subseteq \Sigma$ as a shorthand for the expression $a_1 + \cdots + a_n$.

Example 36. Let Σ denote the alphabet $\{a, b\}$. The rewbl_p $\alpha = (\texttt{?=}\,ab)\,\Sigma\,\Sigma\,\Sigma$ describes the language $L(\alpha) = \{aba, abb\}$ (i.e., the set of all three length words starting with ab). This is because the subexpression $(\texttt{?=}\,ab)$ first "filters" *a prefix of* the input string by matching with the regular expression $ab\Sigma^*$ without consuming any input. Subsequently, the expression $\Sigma\,\Sigma\,\Sigma$ will match any unfiltered three length strings. In short, $(\texttt{?=}\,\alpha)$ tests for free if the matching between $\alpha\Sigma^*$ and the rest of the input string succeeds or not. Similarly, $(\texttt{?!}\,\alpha)$ tests if the matching fails or not.

In this section, in contrast with the fact that every rewb language is indexed (Section 4, Corollary 24), we show that it is not the case for rewbs extended with lookaheads. We do not give a formal semantics of rewbs with lookaheads (see [8] for such a semantics) and only assume the following claim which is sufficient for our purpose:

Claim 37. *Let α_1 and α_2 denote rewbs. Without loss of generality, we assume α_1 and α_2 have no common capturing group indexes. Fix a fresh symbol \$ not appearing in α_1 and α_2. The rewbl_p $(\texttt{?=}\alpha_1\$)\,\alpha_2\$$ (resp. rewbl_n $(\texttt{?!}\,(\texttt{?!}\,\alpha_1\$))\,\alpha_2\$$) describes the language $L(\alpha_1\$) \cap L(\alpha_2\$)$.*

[10] One may think that the proof can be simplified by only comparing the suffix $(w\#)^k \preceq (x\#)^m$ to derive both $|x| \geq |w|$ and $m \geq k$ at once. However, that does not derive the former inequality $|x| \geq |w|$ because, for example, $aaa\# \preceq aa(\#a)a\# = (aa\#)^2$ but not $|aaa| \leq |aa|$.

Note that whether the language class of $\mathrm{rewbl_p}$s is closed under intersections remains open but those of $\mathrm{rewbl_n}$s and rewbls are known to be closed under intersections [8].

Lemma 38. *For an arbitrary recursively enumerable language E, there exist a rewbl_p α and a GSM mapping*[11] *g such that $E = g(L(\alpha))$.*

Proof. This proof is quite similar to the proofs in [12, 7]. Fix a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ that recognizes E, where $\sqcup \in \Gamma$ is the blank symbol and F is a set of accepting states (see [15] for the definition of the other symbols and semantics). Without loss of generality, we can assume that the tape of M is semi-infinite and M never attempts to move left at the leftmost tape position. Let $c, \mathbin{¢}, \$$ be fresh symbols and $\bigvee_{i\in[l]} e_i$ denote $e_1 + e_2 + \cdots + e_l$. Define rewbs

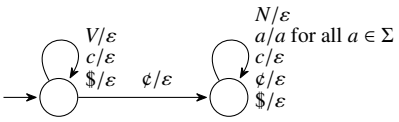
$$\alpha_1 \triangleq \left(\bigvee_{\substack{q\in Q, a\in\Gamma \\ \delta(q,a)=(p,b,\mathrm{L})}} ({}_1\Gamma^*)_1({}_2\Gamma)_2 qa({}_3\Gamma^*)_3\, c\, \backslash 1 p \backslash 2 b \backslash 3\, c + \bigvee_{\substack{q\in Q, a\in\Gamma \\ \delta(q,a)=(p,b,\mathrm{R})}} ({}_1\Gamma^*)_1 qa({}_2\Gamma^*)_2\, c\, \backslash 1 bp \backslash 2\, \sqcup^*\, c \right) \mathbin{¢}\Sigma^*\$,$$

$$\alpha_2 \triangleq q_0({}_1\Sigma^*)_1 \sqcup c \left(({}_2\Gamma \uplus Q)_2\, c\, \backslash 2\, c\right)^* \Gamma^* F \Gamma^*\, c\, \mathbin{¢}\backslash 1\$.$$

Notice that a string that α_1 matches stands for a valid computation of M that is separated by the delimiter c and ends with $¢ \ldots \$$, and α_2 imposes three constraints on the strings it matches: (1) the first component must be equal to the initial configuration of M, (2) for each $m \geq 1$ both $2m^{\text{th}}$ and $(2m+1)^{\text{st}}$ components must coincide, and (3) the penultimate component must be an accepting configuration. A subtle trick is the extra copy of the first component surrounded by $¢$ and \$, which lies in the last component. It is used to indicate by $¢$ the cutting position for a GSM and by \$ where the string ends for a positive lookahead. Thus, we can easily show that $L(\alpha_1) \cap L(\alpha_2)$ is

$$\left\{ q_0 w \sqcup\, c\, t_1\, c\, t_1\, c\, \ldots\, t_{m-1}\, c\, t_{m-1}\, c\, t_m\, c\, \mathbin{¢} w \$ \;\middle|\; q_0 w \sqcup \vdash t_1 \vdash \cdots \vdash t_m \text{ is an accepting computation}, m \geq 1, w \in \Sigma^* \right\}.$$

Let g be the GSM mapping defined as follows:



By Claim 37, $E = g(L((?{=}\,\alpha_1)\,\alpha_2))$ holds. □

Corollary 39. *None of the classes of rewbl_p, rewbl_n, and rewbl is a subclass of indexed languages.*

Proof. Since the class of indexed languages is closed under GSM mappings [1], the class of $\mathrm{rewbl_p}$ cannot be a subclass of indexed languages by Lemma 38. As for $\mathrm{rewbl_n}$, replace $(?{=}\,\alpha_1)$ with $(?!\,(?!\,\alpha_1))$. The last is immediate. □

7. Larsen's hierarchy is within the class of nonerasing stack language

In this section, we construct an NESA A_i that describes $L(x_i)$, where the rewb x_i is given by Larsen [17] and defined over the alphabet $\Sigma = \left\{a_0^l, a_0^m, a_0^r, a_1^l, a_1^m, a_1^r, \ldots\right\}$ as follows: $x_0 \triangleq (a_0^l a_0^m a_0^r)^*$, $x_{i+1} \triangleq (a_{i+1}^l ({}_i\, x_i\,)_i\, a_{i+1}^m \backslash i\, a_{i+1}^r)^*$ $(i \geq 0)$. Our result implies that Larsen's hierarchy is within the class of nonerasing stack languages. Since Larsen showed that no rewb with its nested level less than i can describe $L(x_i)$ [17], it also implies that for every $i \in \mathbb{N}$, there is a nonerasing stack language that needs a rewb of nested level at least i.[12]

[11] A *generalized sequential machine* (GSM) [11] is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$, where Q is a finite set of states, Σ a finite set of input symbols, Δ a finite set of output symbols, $\delta : Q \times \Sigma \to Q$ a transition function, $\lambda : Q \times \Sigma \to \Delta^*$ an output function, and $q_0 \in Q$ a start state. The functions δ and λ can be extended to $\hat{\delta} : Q \times \Sigma^* \to Q$ and $\hat{\lambda} : Q \times \Sigma^* \to \Delta^*$ with $\hat{\delta}(q, \varepsilon) = q$; $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ and $\hat{\lambda}(q, \varepsilon) = \varepsilon$; $\hat{\lambda}(q, wa) = \hat{\lambda}(q, w)\lambda(\hat{\delta}(q, w), a)$, respectively. We write $q \xrightarrow{a/x} q'$ for $\delta(q, a) = q'$ and $\lambda(q, a) = x$. A *GSM mapping* is a mapping $\Sigma^* \to \Delta^*$, $w \mapsto \hat{\lambda}(q_0, w)$ for some GSM $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$.

[12] Technically, Larsen [17] adopts a syntax that excludes unbound references, and so this implied result applies only to rewbs with no unbound references.

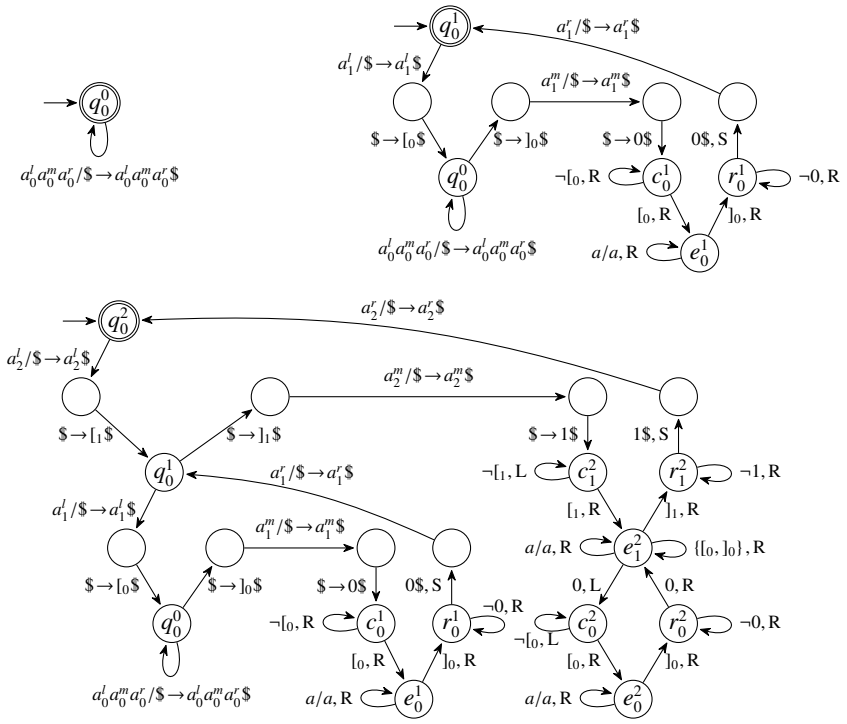


Figure 3: A_0 (upper left), A_1 (upper right), A_2 (lower) (see Notation 15 for the notations)

The NESA A_i has the start state q_0^i which is also its only final state. Figure 3 depicts A_0, A_1, and A_2. A_0 is easy. A_1 is obtained by connecting the eight states to q_0^0 and making q_0^1 the start/final state, as shown in the figure. The five states on the right handle the dereferencing of $\backslash 0$ in x_1. That is, at c_0^1, A_1 first seeks the left-nearest $[_0$, passes the control to e_0^1, checks the input string against the stack at e_0^1, passes the control to r_0^1, and at r_0^1, finally goes back to the right-nearest 0 which must be written on the stack top. In much the same way, A_2 is obtained from A_1 but we must be sensitive to the handling of the dereferencing of $\backslash 1$ because A_2 must handle the dereferencing of not only $\backslash 1$ but also $\backslash 0$ that appears in a string captured by $[_1]_1$ whereas no backreference appears in a string captured by $[_0]_0$ in the case of A_1. To deal with this issue, we connect the three new states c_0^2, e_0^2 and r_0^2 to e_1^2. At e_1^2, if A_2 encounters 0 in a checking, A_2 suspends the checking and first goes to c_0^2 to seek $[_0$, goes to e_0^2 to check the input against the stack by reading out a $]_0$ (no number appears in this checking), and finally goes to r_0^2 to go back to 0 which passed the control to c_0^2. We repeat this modification until A_i is obtained. (Thus, A_i has such states c_j^i, e_j^i, r_j^i for each $j \in \{0, \ldots, i-1\}$.) Therefore,

Theorem 40. *There exists an NESA A_i that recognizes $L(x_i)$.*

8. Conclusions

In this paper, we have shown the following seven results: (1) that every rewb describes an indexed language (Corollary 24), (2) in particular that every rewb without a captured reference describes a nonerasing stack language (Corollary 25), however that there exists (3) a (non-circular) rewb that describes a non-stack language (Theorem 26 and Corollary 31) and (4) a (non-circular) rewb that describes a stack language but not a nonerasing stack language (Corollary 33), (5) therefore that there exists a (non-circular) rewb that needs a captured reference (Corollaries 27 and 32), (6) in contrast with (1) that a rewb with lookaheads can describe a non-indexed language (Corollary 39), and (7) finally that Larsen's hierarchy $\{L(x_i) \mid i \in \mathbb{N}\}$ given in [17] is within the class of nonerasing stack languages (Theorem 40). We have obtained the results by using three automata models, namely NESA, SA, and Nested-SA, and using the semantics of rewbs given in [21, 10] that treats a rewb as a regular expression allowing us to obtain the underlying NFA.

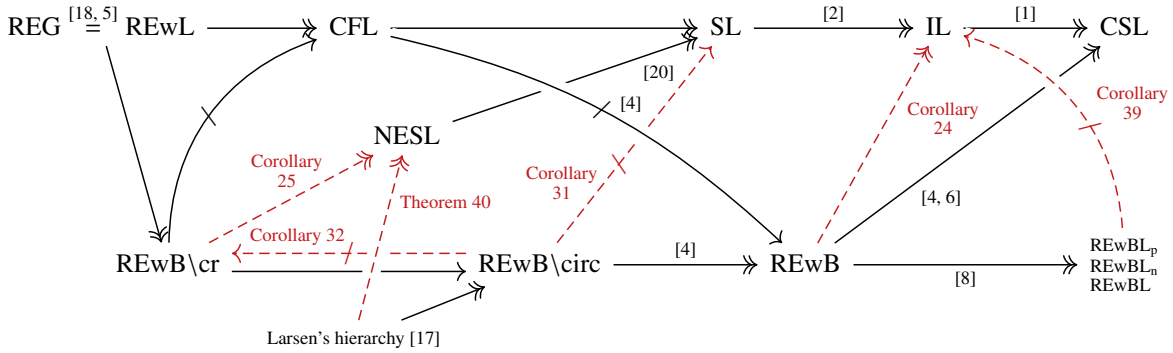


Figure 4: The inclusion relations between the language classes

Figure 4 depicts the inclusion relations between the classes mentioned in the paper. Here, REG stands for the class of regular languages, CFL context free languages, NESL nonerasing stack languages, SL stack languages, IL indexed languages, CSL context sensitive languages, REwL the language class of regular expressions with lookaheads, REwB the language class of rewbs, REwB\cr the language class of rewbs without a captured reference, REwB\circ the language class of rewbs without a circular, $\mathrm{REwBL_p}$ (resp. $\mathrm{REwBL_n}$, REwBL) the language class of rewbs with positive (resp. negative, both positive and negative) lookaheads. For the arrows, $A \to B$ stands for $A \subseteq B$, $A \twoheadrightarrow B$ for $A \subsetneq B$, and $A \nrightarrow B$ for $A \not\subseteq B$, respectively. A label on an arrow refers to the evidence. A red dashed arrow indicates a novel result proved in this paper, where for a strict inclusion, we show for the first time the inclusion itself in addition to the fact that it is strict.

As future work, we would like to investigate the use of the pumping lemma for rewbs without a captured reference that can be derived from the contraposition of our Corollary 25 and a pumping lemma for NESA [20]. We expect it to be a useful tool for discerning which rewbs need captured references. Additionally, we suspect that our construction of NESA in Theorem 40 is useful for not just x_i of [17] but also for more general rewbs that have only one $\backslash i$ for each $(_i\,)_i$, and we would like to investigate further uses of the construction.

References

[1] Alfred V Aho. Indexed grammars—an extension of context-free grammars. *Journal of the ACM (JACM)*, 15(4):647–671, 1968.

[2] Alfred V Aho. Nested stack automata. *Journal of the ACM (JACM)*, 16(3):383–406, 1969.

[3] Alfred V. Aho. *Algorithms for finding patterns in strings*, page 255–300. MIT Press, Cambridge, MA, USA, 1991.

[4] Martin Berglund and Brink van der Merwe. Re-examining regular expressions with backreferences. *Theoretical Computer Science*, 940:66–80, 2023.

[5] Martin Berglund, Brink van Der Merwe, and Steyn van Litsenborgh. Regular expressions with lookahead. *Journal of universal computer science (Online)*, 27(4):324–340, 2021.

[6] Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14(06):1007–1018, 2003.

[7] Benjamin Carle and Paliath Narendran. On extended regular expressions. In *Language and Automata Theory and Applications: Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings 3*, pages 279–289. Springer, 2009.

[8] Nariyoshi Chida and Tachio Terauchi. On lookaheads in regular expressions with backreferences. In *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[9] ECMA International. Ecmascript® 2023 language specification, 2022. `https://tc39.es/ecma262/multipage/\#sec-intro`.

[10] Dominik D Freydenberger and Markus L Schmid. Deterministic regular expressions with back-references. *Journal of Computer and System Sciences*, 105:1–39, 2019.

[11] Seymour Ginsburg. *The mathematical theory of context-free languages*. McGraw-Hill Book, 1966.

[12] Seymour Ginsburg, Sheila A Greibach, and Michael A Harrison. One-way stack automata. *Journal of the ACM (JACM)*, 14(2):389–418, 1967.

[13] Seymour Ginsburg, Sheila A Greibach, and Michael A Harrison. Stack automata and compiling. *Journal of the ACM (JACM)*, 14(1):172–201, 1967.

[14] Takeshi Hayashi. On derivation trees of indexed grammars—an extension of the uvwxy-theorem—. *Publications of the Research Institute for Mathematical Sciences*, 9(1):61–92, 1973.

[15] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Introduction to automata theory, languages, and computation, 2nd Edition*. Addison-Wesley, 2001.

[16] John E. Hopcroft and Jeffrey D. Ullman. Nonerasing stack automata. *Journal of Computer and System Sciences*, 1(2):166–186, 1967.
[17] Kim S Larsen. Regular expressions with nested levels of back referencing form a hierarchy. *Information Processing Letters*, 65(4):169–172, 1998.
[18] Akimasa Morihata. Translation of regular expression with lookahead into finite state automaton. *Computer Software*, 29(1):147–158, 2012.
[19] Taisei Nogami and Tachio Terauchi. On the expressive power of regular expressions with backreferences. In *48th International Symposium on Mathematical Foundations of Computer Science (MFCS 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
[20] William F Ogden. Intercalation theorems for stack languages. In *Proceedings of the first annual ACM symposium on Theory of computing*, pages 31–42, 1969.
[21] Markus L Schmid. Characterising regex languages by regular languages equipped with factor-referencing. *Information and Computation*, 249:1–17, 2016.