On the Expressive Power of Regular Expressions with Backreferences

Taisei Nogami
Waseda University, Tokyo, Japan

Tachio Terauchi
Waseda University, Tokyo, 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 two claims: there exists a rewb whose language does not belong to the class of stack languages, which is a proper subclass of indexed languages, and 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. 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.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Regular expressions, Backreferences, Expressive power

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.71

Related Version *Full Version*: `https://arxiv.org/abs/2307.08531` [13]

Funding This work was supported by JSPS KAKENHI Grant Numbers JP20H04162, JP20K20625, and JP22H03570.

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 (one-way[1] nondeterministic, or 1N) nested stack automata (NSA) [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].

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

Berglund and van der Merwe [4], and Câmpeanu et al. [5] 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.

Meanwhile, there is a class of the languages called stack languages [8, 7]. This class corresponds to the model (1N) stack automata (SA), a restriction of NSA. 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 [8, 11, 14], and its language class is known to be a proper subclass of stack languages [14].

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 $({}_j\alpha)_j$) 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. Finally, Larsen [12] 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 16)
(b) Every rewb without a captured reference describes a nonerasing stack language. (Section 4, Corollary 17)
(c) There exists a rewb that describes a non-stack language. (Section 5, Theorem 18)
(d) The hierarchy given by Larsen [12] is within the class of nonerasing stack languages. (Section 6, Theorem 20)

Note that by (b) and (c), it follows that there is a rewb that *needs* capturing of references (Section 5, Corollary 19). See also Figure 2 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 NSA. Sections 4, 5, and 6 formally state and prove the paper's main contributions listed above. Section 7 concludes the paper with a discussion on future work. For space, the proofs are in the full paper [13].

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 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 notion of ref-words and dereferences [15]. 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* [6]. 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. [5] 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 [12].

Next, we discuss related work on the three automata used throughout the paper, namely SA, NESA, and NSA. 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 [8]. Hopcroft and Ullman discovered a type of Turing machine corresponding to the class of two-way NESA [11]. Ogden proposed a pumping lemma for stack languages and nonerasing stack languages [14]. Aho proposed NSA with a proof of the fact that (1N) NSA and indexed grammars given by himself in [1] are equivalent in their expressive powers, and recognized PDA and SA as special cases of NSA [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 [9].

3 Preliminaries

In this section, we formalize the syntax and the semantics of rewbs following the formalization given in [6]. 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, written $REWB_k$, and the mapping $\mathrm{var} : REWB_k \to \mathcal{P}([k])$ are defined as follows, where $a \in \Sigma_\varepsilon$ and $i \in [k]$:

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

We also write $REWB_0$ for the set REG of regular expressions over Σ, and $REWB$ for the set of all rewbs, namely $\bigcup_{k\geq 0} REWB_k$.

▶ **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 $\mathcal{R}_k : REWB_k \to \mathcal{P}((\Sigma \uplus B_k \uplus [k])^*)$ 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 $\bigcup_{\alpha\in REWB_k} \mathcal{R}_k(\alpha)$.

▶ **Example 5.** $\mathcal{R}_k((_1(a+b)^*)_1 \backslash 1) = \{[_1\}\,\{a,b\}^*\,\{]_1\}\,\{1\} = \left\{\, [_1\, w\,]_1\, 1 \;\middle|\; w \in \{a,b\}^* \right\}$.

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 (see the full version [13] for the formal definitions of $\mathcal{D}_k$ and $v_{[r]}$).

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 $\alpha \in REWB_k$ 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).

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.*

Next, we recall the notions of SA, NESA, and NSA. In this paper, we unify their definitions based on [2, 7] to clarify the different capabilities of these models. First, we review NFA. Here is the definition in the textbook by Hopcroft et al. [10]:

▶ **Definition 9** ([10], 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 $q \overset{w}{\underset{N}{\Longrightarrow}} q'$ denote $q' \in \hat{\delta}(q, w)$. With this notation, the language of an NFA N can be written as follows: $L(N) = \left\{ w \in \Sigma^* \;\middle|\; \exists q_f \in F.\, q_0 \overset{w}{\underset{N}{\Longrightarrow}} q_f \right\}$.

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 9-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, \$, F)$ satisfying the following conditions: the components Q, Σ, q_0 and F are the same as those of NFA. Γ ($\neq \emptyset$) is a finite set of stack symbols, and $Z_0 \in \Gamma$ is an initial stack symbol. The stack 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.[2] The transition function δ has the following two modes, where $\mathrm{L}, \mathrm{S}, \mathrm{R} \notin (\Sigma \cup \Gamma) \uplus \{\#, \$\}$, $\Delta_i \triangleq \{\mathrm{S}, \mathrm{R}\}$, and $\Delta_s \triangleq \{\mathrm{L}, \mathrm{S}, \mathrm{R}\}$:

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

Intuitively, δ works as follows (Definition 10 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), $(q', d, w\$) \in \delta(q, a, Z\$)$ implies $w \in Z\Gamma^*$. 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 10.** Let A be an SA $(Q, \Sigma, \Gamma, \delta, q_0, \#, \$, F)$. An element of the set $I = Q \times \Sigma^* \times \{\#\} (\Gamma \uplus \{\restriction\})^* \{\$\}$ 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:

[2] These special symbols $\#, \$$ representing "bottom" and "top" of the stack respectively do not appear in [7] and are introduced anew in this paper to define NESA and NSA, which will be defined later, in the style of [2]. In fact, SA defined in [7] 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 [11, 14]. Therefore, to make it easy to see that NESA is a restriction of SA, we define SA to also directly have the ability.

(i) $(q, a_i \cdots a_k, \#yZ\restriction\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#yw\restriction\$)$ if $(q', d, w\$) \in \delta(q, a_i, Z\$)$.[3]
(ii) **(a)** $(q, a_i \cdots a_k, \#yZ\restriction\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#y\restriction Z\$)$ if $(q', d, \mathrm{L}) \in \delta(q, a_i, Z\$)$.
(b) if $(q', d, e) \in \delta(q, a_i, Z)$ and $Z = Z_j$, $1 \leq j < n$, then
$(q, a_i \cdots a_k, \#Z_1 \cdots Z_j\restriction \cdots Z_n\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#Z_1 \cdots Z_{j+\overline{e}}\restriction \cdots Z_n\$)$.
(c) $(q, a_i \cdots a_k, \#\restriction Zy\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#Z\restriction y\$)$ if $(q', d, \mathrm{R}) \in \delta(q, a_i, \#)$.

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 $y_1, y_2 \in \Gamma^*$, and $q_f \in F$ such that $(q_0, w, \#Z_0\restriction\$) \vdash_A^* (q_f, \varepsilon, \#y_1\restriction y_2\,\$)$. Let $L(A)$ denote the set of all strings accepted by A.

We next define *nested stack automaton* (NSA) 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:

$$\underline{\#a_1\mathbin{¢}\, b_1b_2\restriction\$}\, a_2a_3\$. \tag{1}$$

Note that the new substack $¢\, b_1b_2\,\$$ 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 $\#a_1¢\, b_1b_2\restriction\$$. 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:

$$\underline{\#a_1¢\, b_1b_2\restriction\$}\, a_2a_3\$ \overset{\mathrm{L}}{\to} \underline{\#a_1¢\, b_1\restriction b_2\,\$}\, a_2a_3\$ \overset{\text{create}}{\longrightarrow} \underline{\#a_1¢\, ¢\, c_1c_2\restriction\$}\, b_1b_2\$\, a_2a_3\$, \tag{2}$$

$$\underline{\#a_1¢\restriction b_1b_2\,\$}\, a_2a_3\$ \overset{\mathrm{L}}{\to} \underline{\#a_1\restriction ¢\, b_1b_2\,\$}\, a_2a_3\$ \overset{\text{create}}{\longrightarrow} \underline{\#¢\, c_1c_2\restriction\$}\, a_1¢\, b_1b_2\,\$\, a_2a_3\$. \tag{3}$$

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:

$$\underline{\#a_1¢\, b_1b_2\restriction\$}\, a_2a_3\$ \overset{\text{pop}}{\to} \underline{\#a_1¢\, b_1\restriction\$}\, a_2a_3\$ \overset{\text{pop}}{\to} \underline{\#a_1¢\restriction\$}\, a_2a_3\$ \overset{\text{destruct}}{\longrightarrow} \underline{\#a_1a_2\restriction a_3\$}. \tag{4}$$

Notice that the stack pointer moves to the right after removing the inner substack. We now define NSA 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, Σ, Γ, 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.[4] The transition function δ has the following four modes, where $\Gamma' \triangleq \Gamma \uplus \{¢\}$:

(i) (pushdown mode) $Q \times \Sigma \times \Gamma\$ \to \mathcal{P}(Q \times \Delta_i \times \Gamma^*\$)$.
(ii) (stack reading mode) (a) $Q\times\Sigma\times\Gamma'\$ \to \mathcal{P}(Q\times\Delta_i\times\{\mathrm{L}\})$, (b) $Q\times\Sigma\times\Gamma' \to \mathcal{P}(Q\times\Delta_i\times\Delta_s)$, (c) $Q \times \Sigma \times \{\#\} \to \mathcal{P}(Q \times \Delta_i \times \{\mathrm{R}\})$.
(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 NSA works with ID and $\vdash$ in the same manner as SA. Given an NSA $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, ¢, \$, F)$, we define ID, $\vdash_A$, and $L(A)$ in the same way as Definition 10 (however, we let I be $Q \times \Sigma^* \times \{\#\}\, (\Gamma \uplus \{¢, \$, \restriction\})^*\, \{\$\}$). 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):

---

[3] We regard $a_{k+1} \cdots a_k$ as ε.
[4] Note that the bottom of the entire stack is always represented by # and not ¢, as mentioned above.

(iii) if $(q', d, ¢w\$) \in \delta(q, a_i, Z)$ and $Z = Z_j$, $1 \leq j < n$, then
$(q, a_i \cdots a_k, \#Z_1 \cdots Z_j \restriction \cdots Z_n\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#Z_1 \cdots ¢w \restriction \$Z_j \cdots Z_n\$)$,
and $(q, a_i \cdots a_k, \#yZ \restriction \$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#y\,¢w \restriction \$Z\$)$ if $(q', d, ¢w\$) \in \delta(q, a_i, Z\$)$.

(iv) $(q, a_i \cdots a_k, \#y_1 ¢ \restriction \$Zy_2\$) \vdash_A (q', a_{i+\overline{d}} \cdots a_k, \#y_1 Z \restriction y_2\$)$ if $(q', d) \in \delta(q, a_i, ¢\$)$.

4 Every rewb describes an indexed language

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 an NSA A_α recognizing the language $L(\alpha)$ as follows.

The NSA 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 $q \xrightarrow{\varepsilon/\$ \to b\$} q'$.[5] 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 NSA 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; NSA can insert such marks anywhere by creating substacks, but due to the restriction of NSA, it cannot go above the position of the mark, much less climb up to the top. Therefore, NSA cannot directly push the result of a dereference 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 dereference of j. The following lemma is immediate.

▶ **Lemma 11.** *Let $k \geq 1$ and $\alpha \in REWB_k$. 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, $\forall q \in Q.\, \exists w \in (\Sigma \uplus B_k \uplus [k])^*.\, \exists q_f \in F.\, q \xRightarrow[N]{w} q_f$.*

[5] Strictly speaking, our NFA (cf. Definition 9) does not allow consuming the empty string ε. However, we can realize the transition $q \xrightarrow{\varepsilon/\$ \to b\$} q'$ alternatively by adding $q \xrightarrow{c/\$ \to b\$, \mathrm{S}} q'$ for each $c \in \Sigma$, i.e., moving by c with the input cursor fixed.

▶ **Corollary 12.** *Let N be the NFA in Lemma 11. For all $q \in Q$ and for all $w \in (\Sigma \uplus B_k \uplus [k])^*$, if $q_0 \xRightarrow[N]{w} q$ then w is matching (see the full version [13] for the proof).*

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

▶ **Theorem 13.** *For every rewb $\alpha \in REWB$, there exists an NSA that recognizes $L(\alpha)$.*

The claim obviously holds when α is a pure regular expression (i.e., $\alpha \in REWB_0$). Suppose that $\alpha \in REWB_k$ with $k \geq 1$. By Lemma 11, 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 12. We construct an NSA $A_\alpha = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \#, ¢, \$, F)$ as follows. Let $Q \triangleq Q_N \uplus \{c_i, e_i, r_i \mid i \in [k]\} \uplus \{W_q \mid q \in Q_N\} \uplus \{E_{p,i}, L_{p,i} \mid p \in Q_N \uplus \{e_i \mid i \in [k]\}, i \in [k]\}$, $\Gamma \triangleq \Sigma \uplus B_k \uplus [k] \uplus Q \uplus \{Z_0\}$, and let δ be the smallest relation that, for all $a \in \Sigma$, $b \in B_k$, $c \in \Sigma$, $i, j \in [k]$, $q, q' \in Q_N$, $Z \in \Gamma$ 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) $\delta(W_q, c, i\$) = \{(c_i, \mathrm{S}, ¢q\$)\}$
(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) $\delta(e_i, c, j) = \{(c_j, \mathrm{S}, ¢e_i\$)\}$ where $i \neq j$
(13) $\delta(r_i, c, Z) = \{(r_i, \mathrm{S}, \mathrm{R})\}$
(14) $\delta(r_i, c, p\$) = \{(E_{p,i}, \mathrm{S}, \$)\}$
(15) $\delta(E_{p,i}, c, ¢\$) = \{(L_{p,i}, \mathrm{S})\}$
(16) $\delta(L_{e_j,i}, c, i) = \{(e_j, \mathrm{S}, \mathrm{R})\}$
(17) $\delta(L_{q,i}, c, i\$) = \{(q, \mathrm{S}, \mathrm{S})\}$

Rule (1) translates $q \xrightarrow[N]{a} q'$ into $q \xrightarrow{a/\$ \to a\$} q'$, (2) translates $q \xrightarrow[N]{b} q'$ into $q \xrightarrow{\varepsilon/\$ \to b\$} q'$, and rules (3)–(17) translate $q \xrightarrow[N]{i} q'$ into a large "transition" to consume the string that corresponds to the dereference 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 $¢q'\$$ 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 $¢e_i\$$ 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 $¢e_j\$$ 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).

We state two lemmas used to prove Theorem 13. Let $\vdash_{(n)}$ denote the subrelation of $\vdash$ derived from the rule (n). The following lemma is immediate from the definition of $\vdash_{(n)}$.

▶ **Lemma 14.** *For all $q, q' \in Q_N$, $w, w' \in \Sigma^*$, $\gamma, \gamma' \in \Gamma^*$,*

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

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

(b) **1.** *for each $b \in B_k$, $(q, w, \#Z_0\gamma\restriction\$) \vdash_{(2)} (q', w, \#Z_0\gamma b\restriction\$)$ if $q \xrightarrow[N]{b} q'$,*

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

In particular, letting $\vdash_{(1),(2)} = \vdash_{(1)} \uplus \vdash_{(2)}$, we obtain the following statement by repeating $(a)_1$ and $(b)_1$ zero or more times: For all $v \in (\Sigma \uplus B_k)^*$, $(q, g(v)\, w, \#Z_0\gamma\restriction\$) \vdash^*_{(1),(2)} (q', w, \#Z_0\gamma v\restriction\$)$ if $q \xRightarrow[N]{v} q'$.

▶ **Lemma 15.** *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 $\beta \in (\Gamma \uplus \{\mathfrak{c}, \$, \restriction\})^*$, the following (a) and (b) are equivalent (see Appendix A for the proof):*

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

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

Proof of Theorem 13 (sketch). 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 $q_0 \xRightarrow[N]{v_0} q_{(0)} \xRightarrow[N]{n_1 v_1} q_{(1)} \xRightarrow[N]{n_2 v_2} \cdots \xRightarrow[N]{n_m v_m} q_{(m)} \in F$. 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 $C_{(1)} = (q_0, w, \#Z_0\restriction\$) \vdash \cdots \vdash C_{(r)} \vdash \cdots \vdash C_{(m)} = (p_m, \varepsilon, \#\beta_m\$)$, where $p_m \in F$ and $C_{(r)} = (p_r, w_r, \#\beta_r\$)$ for each $r \in \{1, \ldots, m\}$. By induction on $r = 1, \ldots, m$, we extract an underlying transition $q_0 \xRightarrow[N]{\gamma_r} p_r$ 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 $p_r \in Q_N$ (the formal proof is available in the full version [13]). ◀

▶ **Corollary 16.** *Every rewb describes an indexed language, but not vice versa.*

Proof. The first half follows by Theorem 13 since 1N NSA and indexed grammars are equivalent [2]. The second half also follows since the class of CFLs is a subclass of indexed languages [1], and the class of rewbs and that of CFLs are incomparable [4]. ◀

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 $(_j \ldots)_j$), 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 an NSA 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 $\mathfrak{c}q'\$$ 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:

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

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 17.** *Every rewb without a captured reference describes a nonerasing stack language, but not vice versa.*[6]

Note that the converse of Corollary 17 fails to hold. In other words, there is 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 6, NESA can recognize nontrivial language (hierarchy) with a captured reference such as Larsen's hierarchy [12].

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 [14], 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 [6] (Example 1) which shows that the rewb $\alpha = (({}_1\backslash 2)_1({}_2\backslash 1a)_2)^*$ 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:

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

The rewb $(({}_1\backslash 4\,a)_1\,({}_2\backslash 3)_2\,({}_3\backslash 2\,a)_3\,({}_4\backslash 1\backslash 3)_4)^*$ describes $\left\{ a^{n(n+7)(2n+1)/6} \,\middle|\, n \in \mathbb{N} \right\}$ and extends the length by a quadratic in n instead (see the full version [13] for the calculation). Thus,

▶ **Theorem 18.** *There exists a rewb that describes a non-stack language.*

From this and Corollary 17, this rewb needs a captured reference, in the sense that:

▶ **Corollary 19.** *There exists a rewb that describes a language that no rewb without a captured reference can describe.*

[6] For the latter part, we can take the language $\{a^n b^n \mid n \in \mathbb{N}\}$ that can be described by an NESA (see the full version [13]) but not by any rewb [4].

6 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 over the alphabet $\Sigma = \{a_0^l, a_0^m, a_0^r, a_1^l, a_1^m, a_1^r, \dots\}$ is given by Larsen [12] and defined 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 $L(x_i)$ [12], 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.[7]

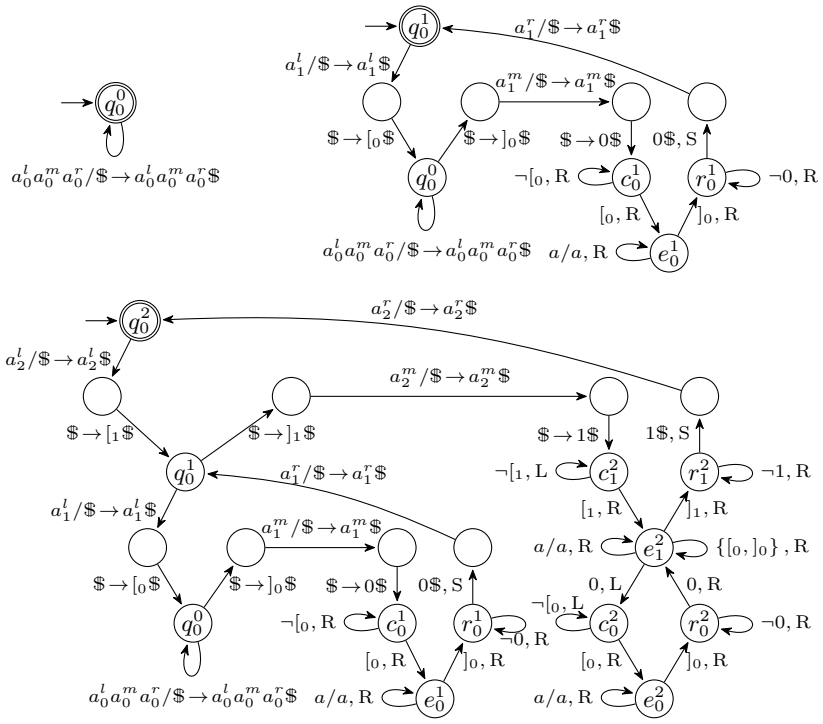

Figure 1 A_0 (upper left), A_1 (upper right), A_2 (lower).

The NESA A_i has the start state q_0^i which is also its only final state. Figure 1 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 dereference 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 dereference of $\backslash 1$ because A_2 must handle the dereference 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

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

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

7 Conclusions

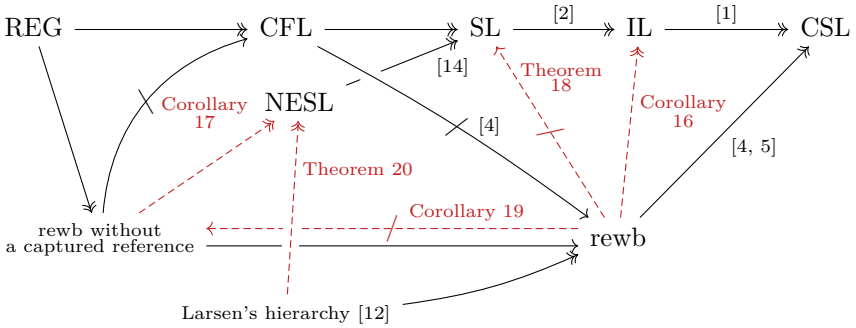

Figure 2 The inclusion relations between the classes.

In this paper, we have shown the following five results: (1) that every rewb describes an indexed language (Corollary 16), (2) in particular that every rewb without a captured reference describes a nonerasing stack language (Corollary 17), (3) however that there exists a rewb that describes a non-stack language (Theorem 18), (4) therefore that there exists a rewb that needs a captured reference (Corollary 19), and (5) finally that Larsen's hierarchy $\{L(x_i) \mid i \in \mathbb{N}\}$ given in [12] is within the class of nonerasing stack languages (Theorem 20). We have obtained the results by using three automata models, namely NESA, SA, and NSA, and using the semantics of rewbs given in [15, 6] that treats a rewb as a regular expression allowing us to obtain the underlying NFA. Figure 2 depicts the inclusion relations between the classes mentioned in the paper. Here, $A \to B$ stands for $A \subseteq B$, $A \twoheadrightarrow B$ for $A \subsetneq B$, and $A \not\twoheadrightarrow B$ 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 17 and a pumping lemma for NESA [14]. 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 20 is useful for not just x_i of [12] 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*, pages 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 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.

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

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

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

9 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.

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

11 John E. Hopcroft and Jeffrey D. Ullman. Nonerasing stack automata. *Journal of Computer and System Sciences*, 1(2):166–186, 1967.

12 Kim S Larsen. Regular expressions with nested levels of back referencing form a hierarchy. *Information Processing Letters*, 65(4):169–172, 1998.

13 Taisei Nogami and Tachio Terauchi. On the expressive power of regular expressions with backreferences, July, 2023. `arXiv:2307.08531`.

14 William F Ogden. Intercalation theorems for stack languages. In *Proceedings of the first annual ACM symposium on Theory of computing*, pages 31–42, 1969.

15 Markus L Schmid. Characterising regex languages by regular languages equipped with factor-referencing. *Information and Computation*, 249:1–17, 2016.

A Proof of Lemma 15

First, we recall the notation $v_{[r]}$ (cf. Section 3) and explain a new notation $v^{(r)}$ informally (see the full version [13] for the formal definitions of $v_{[r]}$ and $v^{(r)}$). Let k be a positive integer and $v = v_0 n_1 v_1 \dots n_m v_m$ ($m = \operatorname{cnt} v$) a matching string over $\Sigma \uplus B_k \uplus [k]$. For each $r = 1, 2, \dots$, 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 follows, therefore $v_{[1]} = b$ and $v_{[2]} = a[_2 b]_2 b$: $v^{(0)} = [_1 a\, \underline{[_2 b]_2}\, 2\,]_1\, 1 \to v^{(1)} = \underline{[_1 a\, [_2 b]_2\, b\,]_1}\, 1 \to v^{(2)} = [_1 a\, [_2 b]_2\, b\,]_1\, abb \to abbabb$. We can easily prove the following claim (see also the full version [13]): For each $r \in \{0, 1, \dots, m\}$,

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

We prepare 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 $I = Q \times \Sigma^* \times \{\#\}\, (\Gamma \uplus \{¢, \$, \dagger\})^*\, \{\$\}$ to $I_\perp \triangleq Q \times \Sigma^*_\perp \times \{\#\}\, (\Gamma \uplus \{¢, \$, \dagger\})^*\, \{\$\}$, and we let $C(w)$ denote an ID $C = (\cdot, w, \cdots)$ and $\vdash'_{(n)}$ denote the following binary relation over $I_\perp$:

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

In addition, 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 $w' \in \Sigma^*$, $C(w) \vdash' C'(w')$ implies $C(w) \vdash C'(w')$.*

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

▶ **Definition 22.** Given $C, C' \in I_\perp$, we write $C \models_{(n)} C'$ if $C \vdash'_{(n)} C'$ and $\forall j, C''. C \vdash'_{(j)} C'' \Longrightarrow j = n \wedge C'' = C'$. 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 $\gamma \in (\Sigma \uplus B_k \uplus [k])^*$, $i \in [k]$, $w \in \Sigma^*$, $\beta \in (\Gamma \uplus \{¢, \$\})^*$ and $p \in Q_N \uplus \{ e_i \mid i \in [k] \}$. Let $m = \operatorname{cnt}(\gamma i)\ (\geq 1)$. If γi is matching,*

$$(c_i, w, \#Z_0\gamma ¢ p \restriction \$ i \beta \$) \models \cdots \models (r_i, w/g((\gamma i)_{[m]}), \#Z_0\gamma ¢ p \restriction \$ i \beta \$)$$

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 $\gamma \notin [_i$, it holds that

$$\begin{aligned}(c_i, w, \#Z_0\gamma ¢ p \restriction \$ i \beta \$) &\models^* (c_i, w, \#Z_0 \restriction \gamma ¢ p \$ i \beta \$) \\ &\models (r_i, w, \#Z_0 \restriction \gamma ¢ p \$ i \beta \$) \models^* (r_i, w, \#Z_0\gamma ¢ p \restriction \$ i \beta \$),\end{aligned}$$

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 $\gamma = \gamma_0 [_i \gamma_1$ ($\gamma_1 \notin [_i$). Moreover, we can further decompose $\gamma_1 = \gamma_2]_i \gamma_3$ ($\gamma_2 \notin [_i,]_i$, $\gamma_3 \notin [_i$) because γi is matching. We prove by induction on m.

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

$$\begin{aligned}&(c_i, w, \#Z_0\gamma_0 [_i \gamma_1 ¢ p \restriction \$ i \beta \$) \models^* (c_i, w, \#Z_0\gamma_0 [_i \restriction \gamma_1 ¢ p \$ i \beta \$) \models (e_i, w, \#Z_0\gamma_0 [_i \restriction \gamma_1 ¢ p \$ i \beta \$) \\ &\models^* (e_i, w', \#Z_0\gamma_0 [_i \gamma_2]_i \restriction \gamma_3 ¢ p \$ i \beta \$) \models (r_i, w', \#Z_0\gamma_0 [_i \gamma_2]_i \restriction \gamma_3 ¢ p \$ i \beta \$) \\ &\models^* (r_i, w', \#Z_0\gamma_0 [_i \gamma_2]_i \gamma_3 ¢ p \restriction \$ i \beta \$).\end{aligned}$$

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 $(\gamma i)^{(0)} = \gamma i = \gamma_0 [_i \gamma_2]_i \gamma_3 i$, $\gamma_3 \notin [_i$.

Case $\{1, \ldots, m\} \Longrightarrow m + 1$: Let $m_0 \triangleq \operatorname{cnt}\gamma_0$ and $l \triangleq \operatorname{cnt}\gamma_2\ (\geq 0)$. Now, $m_0 + l \leq m = \operatorname{cnt}\gamma$ 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 $\gamma_2 \notin]_i$). Thus, letting $w_0 \triangleq w$, $w'_r \triangleq w_{r-1}/g(\lambda_{r-1})$, $w_r \triangleq w'_r/g((\eta_r)_{[m_0+r]})$ and $w' = w_l/g(\lambda_l)$, we have

$$\begin{aligned}&(c_i, w, \#Z_0\gamma ¢ p \restriction \$ i \beta \$) \\ &\models^* (c_i, w, \#Z_0\gamma_0 [_i \restriction \lambda_0 n_{m_0+1} \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\models (e_i, w_0, \#Z_0\gamma_0 [_i \restriction \lambda_0 n_{m_0+1} \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\models^* (e_i, w'_1, \#Z_0\gamma_0 [_i \lambda_0 n_{m_0+1} \restriction \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\models (c_{n_{m_0+1}}, w'_1, \#Z_0\gamma_0 [_i \lambda_0 ¢ r_i \restriction \$ n_{m_0+1} \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\models^* (r_{n_{m_0+1}}, w_1, \#Z_0\gamma_0 [_i \lambda_0 ¢ r_i \restriction \$ n_{m_0+1} \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\qquad\qquad \text{(by } \eta_1 \text{ being matching and induction hypothesis)} \\ &\models (E_{e_i, n_{m_0+1}}, w_1, \#Z_0\gamma_0 [_i \lambda_0 ¢ \restriction \$ n_{m_0+1} \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\models (L_{e_i, n_{m_0+1}}, w_1, \#Z_0\gamma_0 [_i \lambda_0 n_{m_0+1} \restriction \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\models (e_i, w_1, \#Z_0\gamma_0 [_i \lambda_0 n_{m_0+1} \restriction \lambda_1 \cdots n_{m_0+l} \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ \models^* \cdots \models^* &(e_i, w_l, \#Z_0\gamma_0 [_i \lambda_0 n_{m_0+1} \lambda_1 \cdots n_{m_0+l} \restriction \lambda_l]_i \gamma_3 ¢ p \$ i \beta \$) \\ &\qquad\qquad \text{(by similar calculation and induction hypothesis)}\end{aligned}$$

$$\begin{aligned}
&\models^* (e_i, w', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i \restriction \gamma_3 ¢ p\$ i\beta\$)\\
&\models\ \ (r_i, w', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i \restriction \gamma_3 ¢ p\$ i\beta\$)\\
&\models^* (r_i, w', \#Z_0\gamma_0[_i\lambda_0 n_{m_0+1}\lambda_1 \cdots n_{m_0+l}\lambda_l]_i\gamma_3 ¢ p \restriction \$ i\beta\$),
\end{aligned}$$

and

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

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 equation $(*)$, we can write $(\gamma i)^{(m)}$ as

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

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 $w' = w/g((\gamma i)_{[m+1]})$. ◀

Proof of Lemma 15. For arbitrary $w \in \Sigma^*$, by Lemma 23,

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

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 $(q, w, \#Z_0\gamma \restriction \$) \vdash_{(3)} (W_{q'}, w, \#Z_0\gamma i \restriction \$) \vdash' \cdots \vdash' (p, w', \#\beta\$)$, 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 $p = q'$, $w' = w/g((\gamma i)_{[m]})$ and $\beta = Z_0\gamma i \restriction$ by the equality of their rightmost IDs, and thus, (a) follows because $w' \in \Sigma^*$. ◀