The Power of Non-deterministic Reassignment in Infinite-Alphabet Pushdown Automata

Yulia Dubov[1] and Michael Kaminski[2,⋆]

[1] Intel Israel
MTM – Advanced Technology Center,
P.O.B. 1659, Haifa 31015, Israel
yulia.dubov@intel.com
[2] Department of Computer Science,
Technion – Israel Institute of Technology,
Haifa 32000, Israel
kaminski@cs.technion.ac.il

Abstract. In this paper we compare two models of pushdown automata over infinite alphabets, one with non-deterministic reassignment and the other with the deterministic one, and show that the former model is stronger than the latter.

1 Introduction

The study of extension of classical models of automata to languages over infinite alphabets that has been started in the earlier 1990s counts numerous works. Being purely theoretical in the beginning, it soon found a number of applications. Naturally, every time a sequence of words is considered – be it messages passing through the network, URLs clicked by the internet surfer, or XML tags - words are treated as atomic symbols, i.e., elements of an appropriate alphabet, and, in absence of a bound on the length of the words, the alphabet becomes infinite.

A number of models of computation over infinite alphabets is known from the literature, e.g., see [1,3,4,5,6,7,8,9,10,11]. A particular example of a model of computation over infinite alphabets tightly related to this paper is context-free grammars over infinite alphabets introduced in [3]. These grammars look very similar to Document Type Definitions (DTDs) used for defining XML documents, see [2], even though they have been invented earlier than the latter.

Actually, this paper deals with two models of pushdown-automata over infinite alphabets, one of which is the semantics counterpart of context-free grammars over infinite alphabets. Both models are analogous to the classical model in that their sets of states are finite. However, the ability to deal with infinite alphabets is achieved by equipping the machine with a finite number of registers, in which the automaton can store any letter from the infinite input alphabet. While reading the topmost stack and the input symbols, the automaton compares them with

⋆ A part of this paper was written during the author's sabbatical leave to the Division of Mathematical Sciences of the Nanyang Technological University.

O. Grumberg et al. (Eds.): Francez Festschrift, LNCS 5533, pp. 107–127, 2009.

the content of its registers and proceeds according to the results. It also has a mechanism, called reassignment, for updating the register content.

There are (at least) two possibilities of updating the content of the automaton registers. One possibility is to replace the content of one of the registers with the topmost stack symbol or the currently scanned new input symbol, similarly to the approach in [7], and the other is to replace the content of one of the registers with any new symbol from the infinite input alphabet, independently of the current input symbol or the topmost stack symbol, as was done in [3,8,9]. That is, in the latter case, the automaton does not necessarily have to arrive at the symbol in order to store it in its registers. Such ability will be referred to as a non-deterministic reassignment. Infinite alphabet pushdown automata introduced in [3] are a particular example of the latter (non-deterministic reassignment) type. Thus, it is very natural to ask whether the ability of non-deterministic reassignment is necessary for the equivalence of the automata to context-free grammars over infinite alphabets.[1] That is, whether restricting the computation model by requiring the reassignment to be deterministic would result in the computation model of the same power.

In this paper we show that the answer to this question is negative by presenting a language accepted by infinite alphabet pushdown automata with non-deterministic reassignment (NR-IAPDA), but not accepted by their deterministic counterpart – infinite alphabet pushdown automata with deterministic reassignment (DR-IAPDA).[2]

Our example is based on the following idea. We observe that NR-IAPDA can perform certain tasks, using both its stack and "guessing" (i.e., non-deterministic reassignment) abilities, which are beyond the DR-IAPDA computation power. Namely, we consider a computation involving an unknown future input symbol that appears only once in the input. NR-IAPDA can perform the computation by "guessing" the future input input and using it for comparison while reading the preceding portion of the input word. However, DR-IAPDA seemingly would need to use their pushdown store to perform the same task by storing there the input word, until it reaches the "unknown" input symbols (which it is unable to guess) and then comparing the stack symbols with that symbol by popping them out. Now, if we add a different task that requires the use of the pushdown stack (e.g., comparison of two subwords of the input) and has to be performed simultaneously, both of the tasks together cannot be performed by DR-IAPDA, but can be performed by NR-IAPDA.

This paper is organized as follows. The next section contains the basic notation used throughout the paper. In Sections 3 and 4 we recall the defini-

[1] Models of computation are said to be equivalent, if they accept/generate the same class of languages.

[2] One might have the impression that it is quite expected, because, over finite alphabets, nondeterministic pushdown automata are more expressive than deterministic ones. Nevertheless, there are very similar models of finite and pushdown automata over infinite alphabets for which nondeterministic reassignment does not increase the computation power, see [6].

tion (from [3]) of infinite alphabet context-free languages and infinite alphabet pushdown automata (with non-deterministic reassignment), respectively. Infinite-alphabetpushdown automata with deterministic reassignment are defined in Section 5. Finally, in Section 6 we prove that infinite-alphabet pushdown automata with deterministic reassignment are weaker than those with non-deterministic one.

2 Notation

In what follows Σ is a fixed infinite alphabet. An assignment is a word $u_1u_2\cdots u_r$ over Σ such that $u_i \neq u_j$ for $i \neq j$. That is, each symbol from Σ occurs in an assignment at most ones. We denote the set of all assignments of length r by $\Sigma^{r\neq}$. Assignments correspond to the content of all registers of an automaton or a grammar.

For a word $\boldsymbol{w} = w_1w_2\cdots w_n \in \Sigma^*$, we define the content of $\boldsymbol{w}$, denoted $[\boldsymbol{w}]$, by $[\boldsymbol{w}] = \{w_i : i = 1, 2, \ldots, n\}$. That is, $[\boldsymbol{w}]$ consists of all the symbols of Σ which occur in the word $\boldsymbol{w}$.

Throughout this paper we use the following convention.

- Words are always denoted by boldface letters, possibly indexed or primed.
- Boldface low-case Greek letters denote words over Σ.
- Symbols which occur in a word denoted by a boldface letter are always denoted by the same non-boldface letter with some subscript. That is, symbols which occur in $\boldsymbol{\sigma}$ are denoted by σ_i, symbols which occur in $\boldsymbol{w}$ are denoted by w_i, symbols which occur in $\boldsymbol{X}$ are denoted by X_i, etc.

3 Infinite-Alphabet Context-Free Grammars

In this section we recall the definition of infinite-alphabet context-free grammars which motivate infinite-alphabet pushdown automata (with non-deterministic reassignment) defined in the next section.

Definition 1. ([3, Definition 1]) *An infinite-alphabet context-free grammar is a system $G = \langle V, \boldsymbol{u}, P, S\rangle$ whose components are defined as follows.*

- *V is a finite set of variables disjoint from Σ.*
- *$\boldsymbol{u} = u_1u_2\cdots u_r \in \Sigma^{r\neq}$ is the initial assignment.*
- *$P \subseteq (V \times \{1, 2, \ldots, r\}) \times (V \cup \{1, 2, \ldots, r\})^*$ is a finite set of productions. For $A \in V$, $k = 1, 2, \ldots, r$, and $\boldsymbol{a} \in (V \cup \{1, 2, \ldots, r\})^*$, we write the triple $(A, k, \boldsymbol{a})$ as $(A, k) \to \boldsymbol{a}$.*
- *$S \in V$ is the start symbol.*

For $A \in V$, $\boldsymbol{w} = w_1w_2\cdots w_r \in \Sigma^{r\neq}$, and $\boldsymbol{X} = X_1X_2\cdots X_n \in (\Sigma \cup (V \times \Sigma^{r\neq}))^*$, we write $(A, \boldsymbol{w}) \Rightarrow \boldsymbol{X}$, if there exist a production $(A, k) \to \boldsymbol{a} \in P$, $\boldsymbol{a} = a_1a_2\cdots a_n \in (V \cup \{1, 2, \ldots, r\})^*$, and $\sigma \notin [\boldsymbol{w}] \setminus \{w_i\}$ such that the condition below is satisfied.

Let $\boldsymbol{w'} \in \Sigma^{r\neq}$ be obtained from $\boldsymbol{w}$ by replacing w_k with σ. Then all for $i = 1, 2, \ldots, n$ the following holds.

- If $a_i = m \in \{1, 2, \ldots, r\}$, then $X_i = w'_m$.
- If $a_i = A' \in V$, then $X_i = (A', \boldsymbol{w}')$.

For two words $\boldsymbol{X}, \boldsymbol{Y} \in (\Sigma \cup (V \times \Sigma^{r\neq}))^*$, we write $\boldsymbol{X} \Rightarrow \boldsymbol{Y}$, if there exist words $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3 \in (\Sigma \cup (V \times \Sigma^{r\neq}))^*$ and $(A, \boldsymbol{w}) \in V \times \Sigma^{r\neq}$ such that $\boldsymbol{X} = \boldsymbol{X}_1(A, \boldsymbol{w})\boldsymbol{X}_2$, $\boldsymbol{Y} = \boldsymbol{X}_1\boldsymbol{X}_3\boldsymbol{X}_2$ and $(A, \boldsymbol{w}) \Rightarrow \boldsymbol{X}_3$. As usual, the reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$ and the language $L(G)$ generated by G is defined by

$$L(G) = \{\boldsymbol{\sigma} \in \Sigma^* : (S, \boldsymbol{u}) \Rightarrow^* \boldsymbol{\sigma}\}.$$

Example 1. Let $G = \langle V, \boldsymbol{u}, P, S\rangle$, where

- $V = \{S, A\}$,
- $\boldsymbol{u} = u_1u_2\$$, and
- P consists of
  - $(S, 1) \to S$,
  - $(S, 2) \to A2$, and
  - $(A, 1) \to 3A1|\epsilon$.

It is not hard to verify that $L(G) = L$, where

$$L = \{\$^{|\boldsymbol{w}|}\boldsymbol{w}\delta : \$ \notin [\boldsymbol{w}] \text{ and } \delta \notin [\boldsymbol{w}] \cup \{\$\}\}.^3 \tag{1}$$

For example, the word $\$^3\sigma_1\sigma_2\sigma_3\delta$, where $\delta \notin \{\sigma_1, \sigma_2, \sigma_3\}$,[4] is derived as follows.

$$(S, u_1u_2\$) \Rightarrow (S, \tau u_2\$) \Rightarrow (A, \tau\delta\$)\delta \Rightarrow \$(A, \sigma_1\delta\$)\sigma_1\delta$$

$$\Rightarrow \$\$(A, \sigma_2\delta\$)\sigma_1\sigma_2\delta \Rightarrow \$\$\$(A, \sigma_3\delta\$)\sigma_1\sigma_2\sigma_3\delta \Rightarrow \$\$\$\sigma_1\sigma_2\sigma_3\delta,$$

where $\tau \notin \{\delta, \sigma_1, \sigma_2, \sigma_3\}$. That is, by the production $(S, 1) \to S$, the first register is reset to τ, which allows G to store δ in the second register at the following derivation step.

We conclude this section with the following closure property of context-free languages over infinite alphabets that was somehow missed in [3].

Proposition 1. *The class of context-free languages over infinite alphabets is closed under reversing.*

Proof. Let $G = \langle V, \boldsymbol{u}, P, S\rangle$ be an infinite-alphabet context-free grammar. A straightforward induction on the derivation length shows that the reversal of the language $L(G)$ is generated by the infinite-alphabet context-free grammar $G^R = \langle V, \boldsymbol{u}, P^R, S\rangle$, where

$$P^R = \{(A, k) \to \boldsymbol{a}^R : (A, k) \to \boldsymbol{a} \in P\}.$$

[3] As we shall see in Section 6, this language is not accepted by infinite alphabet pushdown automata with deterministic reassignment defined in Section 5.

[4] Of course, $\sigma_1, \sigma_2, \sigma_3$ are not necessarily pairwise distinct.

4 Infinite-Alphabet Pushdown Automata with Non-deterministic Reassignment

Here we recall the definition of infinite-alphabet pushdown automata with non-deterministic reassignment which constitute the semantics counterpart of the infinite-alphabet context-free grammars introduced in the previous section.

Definition 2. ([3, Definition 2]) *An infinite-alphabet pushdown automaton with non-deterministic reassignment (NR-IAPDA)*[5] *is a system* $\mathcal{A} = \langle Q, s_0, \boldsymbol{u}, \rho, \mu \rangle$, *whose components are defined as follows.*

- Q *is a finite set of states.*
- $s_0 \in Q$ *is the initial state.*
- $\boldsymbol{u} = u_1 u_2 \cdots u_r \in \Sigma^{r\neq}$, *is the initial assignment to the* r *registers of* $\mathcal{A}$.
- $\rho : Q \to \{1, 2, \ldots, r\}$ *is a partial function from* Q *to* $\{1, 2, \ldots, r\}$ *called the reassignment. Intuitively, if* $\mathcal{A}$ *is in state* q *and* $\rho(q)$ *is defined, then* $\mathcal{A}$ *may non-deterministically replace the content of the* $\rho(q)$*th register with a new symbol of* Σ *not occurring in any other register.*
- μ *is a transition function from* $Q \times (\{1, 2, \ldots, r\} \cup \{\epsilon\}) \times \{1, 2, \ldots, r\}$ *to finite subsets of* $Q \times \{1, 2, \ldots, r\}^*$. *Intuitively, if* $(p, j_1 j_2 \cdots j_\ell) \in \mu(q, \epsilon, j)$, *then* $\mathcal{A}$, *whenever it is in state* q, *with content of the* j*th register at the top of the stack, may replace the top symbol on the stack with the content of* j_1*th,*j_2*th,*$\ldots$ j_ℓ*th registers, enter state* p *(without reading the input symbol), and pass to the next input symbol (possibly* ϵ*). Similarly, if* $(p, j_1 j_2 \cdots j_\ell) \in \mu(q, k, j)$, *then* $\mathcal{A}$, *whenever it is in state* q, *with content of the* j*th register at the top of the stack, and the input symbol being equal to the content of* k*th register, may replace the top symbol of the stack with the content of the* j_1*th,*j_2*th,*$\ldots$*,*j_ℓ*th registers (in this order, read top-down), enter the state* p, *and pass to the next input symbol (possibly* ϵ*).*

An instantaneous description of $\mathcal{A}$ (on a given input word) is a member of $Q \times \Sigma^{r\neq} \times \Sigma^* \times \Sigma^*$. The first component of an instantaneous description is the (current) state of the automaton, the second is the assignment consisting of the content of its registers (in the increasing order of their indexes), the third component is the portion of the input yet to be read, and the last component is the content of the pushdown store read top down.

Next we define the relation $\vdash$ (yielding in one step) between two instantaneous descriptions $(p, v_1 v_2 \cdots v_r, \sigma\boldsymbol{\sigma}, \tau\boldsymbol{\tau})$ and $(q, w_1 w_2 \cdots w_r, \boldsymbol{\sigma}, \boldsymbol{\alpha\tau})$, $\sigma \in \Sigma \cup \{\epsilon\}$ and $\tau \in \Sigma$. We write

$$(p, v_1 v_2 \cdots v_r, \sigma\boldsymbol{\sigma}, \tau\boldsymbol{\tau}) \vdash (q, w_1 w_2 \cdots w_r, \boldsymbol{\sigma}, \boldsymbol{\alpha\tau})$$

if and only if the following holds.

- If $\rho(p)$ is not defined, then $w_k = v_k, k = 1, 2, \ldots, r$. Otherwise, $w_k = v_k$ for $k \neq \rho(p)$, and $w_{\rho(p)} \in \Sigma \setminus \{v_1, \ldots, v_{\rho(p)-1}, v_{\rho(p)+1}, \ldots, v_r\}$.

[5] In [3] the model is called just infinite-alphabet pushdown automata.

- If $\sigma = \epsilon$, then for some $j = 1, \ldots, r$, $\tau = w_j$ and there is a transition $(q, j_1 j_2 \cdots j_\ell) \in \mu(p, \epsilon, j)$ such that $\boldsymbol{\alpha} = w_{j_1} w_{j_2} \cdots w_{j_\ell}$.
- If $\sigma \neq \epsilon$, then for some $k, j = 1, \ldots, r$, $\sigma = w_k$, $\tau = w_j$, and there is a transition $(q, j_1 j_2 \cdots j_\ell) \in \mu(p, k, j)$ such that $\boldsymbol{\alpha} = w_{j_1} w_{j_2} \cdots w_{j_n}$.

We denote the reflexive and transitive closure of $\vdash$ by $\vdash^*$ and say that $\mathcal{A}$ accepts a word $\boldsymbol{\sigma} \in \Sigma^*$, if $(s_0, \boldsymbol{u}, \boldsymbol{\sigma}, u_r) \vdash^* (q, \boldsymbol{v}, \epsilon, \epsilon)$, for some $q \in Q$ and some $\boldsymbol{v} \in \Sigma^{r\neq}$.[6] Finally, the language $L(\mathcal{A})$ accepted by $\mathcal{A}$ is defined by

$$L(\mathcal{A}) = \{\boldsymbol{\sigma} \in \Sigma^* : \text{for some } q \in Q \text{ and some } \boldsymbol{v} \in \Sigma^{r\neq}, (s_0, \boldsymbol{u}, \boldsymbol{\sigma}, u_r) \vdash^* (q, \boldsymbol{v}, \epsilon, \epsilon)\}.$$

Example 2. Consider the NR-IAPDA $\mathcal{A} = \langle Q, s_0, \boldsymbol{u}, \rho, \mu \rangle$, where

- $Q = \{s_0, s, p, q\}$;
- $\boldsymbol{u} = u_1 u_2 \$$;
- $\rho(s_0) = \rho(q) = 1$ and $\rho(s) = 2$; and
- μ is defined as follows.

(i) $\mu(s_0, \epsilon, 3) = \{(s, 3)\}$
(ii) $\mu(s, \epsilon, 3) = \{(p, 2)\}$
(iii) $\mu(p, 3, 2) = \{(p, 32)\}$
(iv) $\mu(p, 3, 3) = \{(p, 33)\}$
(v) $\mu(p, \epsilon, 2) = \{(q, 2)\}$
(vi) $\mu(p, \epsilon, 3) = \{(q, 3)\}$
(vii) $\mu(q, 1, 3) = \{(q, \epsilon)\}$
$(viii)$ $\mu(q, 2, 2) = \{(q, \epsilon)\}$

It is not hard to verify that $L(G) = L$, where L is the language from Example 1 defined by (1). The intuition lying behind the accepting run of $\mathcal{A}$ on an input of the form $\$^{|\boldsymbol{w}|}\boldsymbol{w}\delta$, $\delta \notin [\boldsymbol{w}] \cup \{\$\}$, is as follows.

- By the transition of type (i), $\mathcal{A}$ resets the first register to some $\tau \neq \delta$, which allows it to store δ in the second register at the following computation step, cf. Example 1.
- Using the transitions of type (ii), $\mathcal{A}$ "guesses" δ, stores it in the second register, and replaces \$ with δ at the bottom of the stack.
- Then, using the transitions of types (iii) and (iv), $\mathcal{A}$ pushes \$ into the stack.
- After reading the last \$, by the transitions of types (v) or (vi), $\mathcal{A}$ enters the popping state q.
- Then, for each input symbol from $\boldsymbol{w}$ is compared with δ and, if they differ, $\mathcal{A}$ pops \$ out of the stack using the transition of type (vii).
- Finally, the transition of type $(viii)$ applies to verify that the last input symbol is indeed δ and that the number of occurrences of \$ in the input is indeed $|\boldsymbol{w}|$.

[6] Recall that the initial assignment $\boldsymbol{u} = u_1 u_2 \cdots u_r$ is of length r.

For example, $\mathcal{A}$ accepts the word $\$^3\sigma_1\sigma_2\sigma_3\delta$, where $\delta \notin \{\sigma_1, \sigma_2, \sigma_3\}$, by the following sequence of computation steps.

$$(s_0, u_1u_2\$, \$\$\$\sigma_1\sigma_2\sigma_3\delta, \$) \vdash (s, \tau u_2\$, \$\$\$\sigma_1\sigma_2\sigma_3\delta, \$) \vdash (p, \tau\delta\$, \$\$\$\sigma_1\sigma_2\sigma_3\delta, \delta)$$

$$\vdash (p, \tau\delta\$, \$\$\sigma_1\sigma_2\delta_3\delta, \$\delta) \vdash (p, \tau\delta\$, \$\sigma_1\sigma_2\sigma_3\delta, \$\$\delta) \vdash (p, \tau\delta\$, \sigma_1\sigma_2\sigma_3\delta, \$\$\$\delta)$$

$$\vdash (q, \tau\delta\$, \sigma_1\sigma_2\sigma_3\delta, \$\$\$\delta) \vdash (q, \sigma_1\delta\$, \sigma_2\sigma_3\delta, \$\$\delta) \vdash (q, \sigma_2\delta\$, \sigma_3\delta, \$\delta)$$

$$\vdash (q, \sigma_3\delta\$, \delta, \delta) \vdash (q, \sigma_3\delta\$, \epsilon, \epsilon),$$

where $\tau \notin \{\delta, \sigma_1, \sigma_2, \sigma_3\}$, cf. Example 1.

Theorem 1. ([3], see also [6, Appendix D].) *A language is generated by an infinite-alphabet context-free grammar if and only if it is accepted by an NR-IAPDA.*

5 Infinite-Alphabet Pushdown Automata with Deterministic Reassignment

In this section we define infinite-alphabet pushdown automata with deterministic reassignment (DR-IAPDA). Instead of the ability to "guess" symbols for reassignment, this model of pushdown automata over infinite alphabets is limited to altering its registers only by replacing their content with the current input symbol (similarly to the finite-memory automata introduced in [7]) or with the symbol at the top of the stack.

Definition 3. *An infinite-alphabet pushdown automaton with deterministic reassignment (DR-IAPDA) is a system* $\mathcal{A} = \langle Q, s_0, \boldsymbol{u}, \pi, \rho, \mu\rangle$ *whose components are defined as follows.*

- Q *is a finite set of states.*
- $s_0 \in Q$ *is the initial state.*
- $\boldsymbol{u} = u_1u_2\cdots u_r \in \Sigma^{r\neq}$ *is the initial assignment to the* r *registers of* $\mathcal{A}$*.*
- $\pi, \rho : Q \to \{1, 2, \ldots, r\}$ *are partial functions from* Q *to* $\{1, 2, \ldots, r\}$ *called stack-based reassignment and input-based reassignment, respectively.*
- μ *is a transition function from* $Q \times (\{1, 2, \ldots, r\} \cup \{\epsilon\}) \times \{1, 2, \ldots, r\}$ *to finite subsets of* $Q \times \{1, 2, \ldots, r\}^*$*.*

Intuitively, a computation step of the automaton $\mathcal{A}$ from state q is composed of the following sequence of "primitive" actions.

- If the top stack symbol occurs in no register, the stack-based reassignment is performed. That is, the content of the register $\pi(q)$ is replaced with this symbol. Otherwise the finite memory of $\mathcal{A}$ remains intact.
- After that, $\mathcal{A}$ may
  - either perform an ϵ-transition, if defined, or
  - proceed to the next input symbol.

In the latter case, if the next input symbol occurs in no register, the input-based reassignment is performed. That is, the content of the register $\rho(q)$ is replaced with this symbol. Otherwise the finite memory of $\mathcal{A}$ remains intact. Then (similarly to its NR-IAPDA counterpart) $\mathcal{A}$ performs a transition involving the next input symbol.

The definition of a DR-IAPDA instantaneous description is exactly like that of NR-IAPDA, and the yielding in one step relation between two DR-IAPDA instantaneous descriptions $(p, v_1v_2\cdots v_r, \sigma\boldsymbol{\sigma}, \tau\boldsymbol{\tau})$ and $(q, w_1w_2\cdots w_r, \boldsymbol{\sigma}, \boldsymbol{\alpha\tau})$, $\sigma \in \Sigma \cup \{\epsilon\}$ and $\tau \in \Sigma$, is as defined below. We write

$$(p, v_1v_2\cdots v_r, \sigma\boldsymbol{\sigma}, \tau\boldsymbol{\tau}) \vdash (q, w_1w_2\cdots w_r, \boldsymbol{\sigma}, \boldsymbol{\alpha\tau}),$$

if and only if the following holds.

Let $v'_1, v'_2, \ldots, v'_r$ be defined as follows. If there is a $j = 1, \ldots, r$ such that $v_j = \tau$, then $v'_1v'_2\cdots v'_r = v_1v_2\cdots v_r$. Otherwise, $v'_{\pi(p)} = \tau$ and $v'_m = v_m$, for each $m \neq \pi(p)$.

- If $\sigma = \epsilon$, then $w_1w_2\cdots w_r = v'_1v'_2\cdots v'_r$, and for the $j = 1, \ldots, r$ such that $w_j = \tau$, there is a $(q, j_1j_2\cdots j_\ell) \in \mu(p, \epsilon, j)$, such that $\boldsymbol{\alpha} = w_{j_1}w_{j_2}\cdots w_{j_\ell}$.
- If $\sigma \in \Sigma$, then
  - either there is a $k = 1, \ldots, r$ such that $u_k = \sigma$ and $w_1w_2\cdots w_r = v'_1v'_2\cdots v'_r$, or $w_{\rho(p)} = \sigma$ and $w_m = v'_m$, for each $m \neq \rho(p)$; and
  - for the (unique) $k, j = 1, \ldots, r$ such that $w_k = \sigma$ and $w_j = \tau$ there is a $(q, j_1j_2\cdots j_\ell) \in \mu(p, k, j)$ such that $\boldsymbol{\alpha} = w_{j_1}w_{j_2}\cdots w_{j_\ell}$.

The language accepted by DR-IAPDA $\mathcal{A}$ is defined by

$$L(\mathcal{A}) = \{\boldsymbol{\sigma} : (s_0, \boldsymbol{u}, \boldsymbol{\sigma}, u_r) \vdash^* (p, \boldsymbol{v}, \epsilon, \epsilon)\},$$

where, as usual, $\vdash^*$ is the reflexive and transitive closure of $\vdash$.

Example 3. In this example we present a DR-IAPDA $\mathcal{A}$ that accepts the language

$$L^R = \{\delta\boldsymbol{w}\$^{|\boldsymbol{w}|} : \$ \notin [\boldsymbol{w}] \text{ and } \delta \notin [\boldsymbol{w}] \cup \{\$\}\}.$$

That is, L^R is the reversal of the language L defined by (1) in Example 1. The idea lying behind the definition of A is quite standard. Namely, the computation of $\mathcal{A}$ on an input of the form $\delta\boldsymbol{w}\$^n$, $\boldsymbol{w} = w_1\cdots w_n$ and $\delta \notin [\boldsymbol{w}]$, is as follows.

- First $\mathcal{A}$ stores δ in one of its registers.
- Then, for each $i = 1, \ldots, n$, $\mathcal{A}$ compares w_i with δ and, if they are different, pushes \$ into the stack.
- Finally, $\mathcal{A}$ compares the stack content with the \$-suffix of the input, by popping the \$s out of the stack while reading the suffix.

The only little technical problem with the above description of $\mathcal{A}$ is the register in which δ will be stored. Namely, $\mathcal{A}$ cannot store it in register k, if $\delta = u_{k'}$ for some $k' \neq k$. Therefore, $\mathcal{A}$ starts the computation by an ϵ-move, "guessing" an appropriate register for storing δ.

So, let $\mathcal{A} = \langle Q, s_0, \boldsymbol{u}, \rho, \pi, \mu \rangle$, where

- $Q = \{s_0, q\} \cup \{s_1, p_1\} \cup \{s_2, p_2\}$;[7]
- $\boldsymbol{u} = u_1 u_2 \$$;
- $\rho(s_k) = 3 - k$ and $\rho(p_k) = k$, $k = 1, 2$;
- π is the empty domain function; and
- μ is defined as follows.
  - (i) $\mu(s_0, \epsilon, 3) = \{(s_k, 3) : k = 1, 2\}$;
  - (ii) $\mu(s_k, k, 3) = \{(p_k, 3)\}$, $k = 1, 2$;
  - (iii) $\mu(p_k, k, 3) = \{(p_k, 33)\}$, $k = 1, 2$;
  - (iv) $\mu(p_k, \epsilon, 3) = \{(q, \epsilon)\}$, $k = 1, 2$; and
  - (v) $\mu(q, 3, 3) = \{(q, \epsilon)\}$.

Now, it is not hard to verify that, indeed, $L(\mathcal{A}) = L^R$. The intuition lying behind the accepting run of $\mathcal{A}$ on an input of the form $\delta \boldsymbol{w} \$^n$, $\boldsymbol{w} = w_1 \cdots w_n$ and $\delta \notin [\boldsymbol{w}]$, is as follows.

- By an appropriate transition of type (i), $\mathcal{A}$ enters state s_k such that $\delta \neq u_k$, $k = 1, 2$.
- Then, by the transition of type (ii), $\mathcal{A}$ reads δ, stores it in the register $3 - k$, and enters the state p_i.
- After that, for each $i = 1, \ldots, n$, by the transition of type (iii), $\mathcal{A}$ compares w_i with δ and, if they are different, pushes \$ into the stack. Note that, after the entire $\boldsymbol{w}$ has been read, the stack content is $\$^{n+1}$.
- Next, by the transition of type (iv), $\mathcal{A}$ enters the popping state q. Since in this move, the topmost \$ has been popped out of the stack, the stack content becomes $\$^n$.
- Finally, using the transition of type (v), $\mathcal{A}$ verifies that the number of \$s in the stack is n, by popping them out of the stack while reading the \$-suffix of the input.

For example, $\mathcal{A}$ accepts the word $\$^3 \sigma_1 \sigma_2 \sigma_3 \delta$, where $\delta \notin \{\sigma_1, \sigma_2, \sigma_3\}$ and $\delta \neq u_1$, by the following sequence of computation steps, cf. Example 2.

$$(s_0, u_1 u_2 \$, \delta \sigma_1 \sigma_2 \sigma_3 \$\$\$, \$) \vdash (s, u_1 u_2 \$, \delta \sigma_1 \sigma_2 \sigma_3 \$\$\$, \$) \vdash (p_1, u_1 \delta \$, \sigma_1 \sigma_2 \sigma_3 \$\$\$, \$)$$
$$\vdash (p_1, \sigma_1 \delta \$, \sigma_2 \sigma_3 \$\$\$, \$\$) \vdash (p_1, \sigma_2 \delta \$, \sigma_3 \$\$\$, \$\$\$) \vdash (p_1, \sigma_3 \delta \$, \$\$\$, \$\$\$\$)$$
$$\vdash (q, \sigma_3 \delta \$, \$\$\$, \$\$\$) \vdash (q, \sigma_3 \delta \$, \$\$, \$\$) \vdash (q, \sigma_3 \delta \$, \$, \$) \vdash (q, \sigma_3 \delta \$, \epsilon, \epsilon).$$

Theorem 2. *For any DR-IAPDA $\mathcal{A}$, $L(\mathcal{A}) \neq L$.*

The proof of Theorem 2 is presented in the next section.

Remark 1. Since, as we show in this paper, the language L from Example 1 defined by (1) is not a DR-IAPDA languages and L^R is the reversal of L, the class

[7] As described above, from the initial state s_0, by an ϵ-move, $\mathcal{A}$ enters the state s_1, stores δ in the second register, and then proceeds to the state p_1, if $\delta \neq u_1$; and enters the state s_2, stores δ in the first register, and then proceeds to the state p_2, if $\delta \neq u_2$.

of the DR-IAPDA languages is not closed under reversing. Note that, by Proposition 1 and Theorem 1, the class of the NR-IAPDA languages is closed under reversing.

Next we show how DR-IAPDA can be simulated by NR-IAPDA.[8] That is, we show how deterministic reassignment can be simulated by non-deterministic one. The idea lying behind simulation of a DR-IAPDA $\mathcal{A}$ by an NR-IAPDA $\mathcal{A}^N$ is rather simple: we just postpone the reassignment of $\mathcal{A}^N$ to the very last moment (in all states), and then force $\mathcal{A}^N$ to move consulting the reassigned register, which would correspond to the move of $\mathcal{A}$ after the reassignment has been made.

Proposition 2. *Every DR-IAPDA language is accepted by an NR-IAPDA.*

Proof. Let $\mathcal{A} = \langle Q, s_0, \boldsymbol{u}, \rho, \pi, \mu \rangle$ be a DR-IAPDA. We shall prove that $L(\mathcal{A}) = L(\mathcal{A}^N)$, where the NR-IAPDA $\mathcal{A}^N = \langle Q^N, s_0^N, \boldsymbol{u}^N, \rho^N, \mu^N \rangle$ is defined as follows.

- $Q^N = Q \cup Q^\pi \cup Q^\rho \cup Q^{\pi\rho}$, where
  - $Q^\pi = \{q^\pi : q \in Q\}$,
  - $Q^\rho = \{q^\rho : q \in Q\}$, and
  - $Q^{\pi\rho} = \{q^{\pi\rho} : q \in Q\}$.[9]

  The intuitive meaning of the sets of states Q, Q^π, Q^ρ, and $Q^{\pi\rho}$ is as follows, see also the definition of ρ^N.
  - If no stack- or input-based reassignment has been made at the next computation step, then $\mathcal{A}^N$ enters the corresponding state from Q.
  - If only an input-based reassignment has been made at the next computation step, then $\mathcal{A}^N$ enters a state from Q^ρ.
  - Otherwise, $\mathcal{A}^N$ enters a state from Q^π, in which it performs the stack-based reassignment of $\mathcal{A}$. If, in addition, $\mathcal{A}$ has made an input-based reassignment, then $\mathcal{A}^N$, by an ϵ-move, proceeds to the corresponding state from $Q^{\pi\rho}$, in which it performs the corresponding input-based reassignment.
- $s_0^N = s_0$.
- $\boldsymbol{u}^N = \boldsymbol{u}$.
- ρ^N is defined by
  - $\rho^N(q^\pi) = \pi(q)$; and
  - $\rho^N(q^\rho) = \rho^N(q^{\pi\rho}) = \rho(q)$.
- For the definition of μ^N, let the functions μ^π and μ^ρ be defined by
  - $\mu^\pi(q,k,j) = \{(p^\pi, j_1 j_2 \cdots j_\ell) : (p, j_1 j_2 \cdots j_\ell) \in \mu(q,k,j)\}$; and
  - $\mu^\rho(q,k,j) = \{(p^\rho, j_1 j_2 \cdots j_\ell) : (p, j_1 j_2 \cdots j_\ell) \in \mu(q,k,j)\}$.

[8] Note, that DR-IAPDA are not a special case of NR-IAPDA. This is because nondeterministic reassignment is made *before* seeing the input symbol, whereas the deterministic reassignment is made *after*.

[9] Renaming the elements of Q, if necessary, we may assume that the sets Q, Q^π, Q^ρ, and $Q^{\pi\rho}$ are mutually disjoint.

Then μ^N is defined as follows.

- $\mu^N(q,k,j) = \mu(q,k,j) \cup \mu^\pi(q,k,j) \cup \mu^\rho(q,k,j)$.
- $\mu^N(q^\pi,k,j) = \mu^N(q,k,j)$, if $j = \pi(q)$, and is empty, otherwise. That is, according to the above intuition, after making a stack-based reassignment, $\mathcal{A}^N$ must, immediately, "use" it.
- $\mu^N(q^\pi,\epsilon,j) = \{(q^{\pi\rho},j)\}$.
- $\mu^N(q^\rho,k,j) = \mu^N(q,k,j)$, if $k = \rho(q)$, and is empty, otherwise. Similarly, to a stack-based reassignment, after making an input-based reassignment, $\mathcal{A}^N$ must, immediately, "use" it.
- $\mu^N(q^{\pi\rho},k,j) = \mu^N(q,k,j)$, if $j = \pi(q)$ and $k = \rho(q)$, and is empty, otherwise. Here, $\mathcal{A}^N$ immediately moves according to both stack- and input-based reassignments, simulating in this way the corresponding move of $\mathcal{A}$.

For the proof of the inclusion $L(\mathcal{A}) \subseteq L(\mathcal{A}^N)$, let

$$\begin{aligned}(q_0,\boldsymbol{u}_0,\boldsymbol{\sigma}_0,\boldsymbol{\tau}_0) \vdash \cdots \vdash \\ (q_i,\boldsymbol{u}_i,\boldsymbol{\sigma}_i,\boldsymbol{\tau}_i) \vdash (q_{i+1},\boldsymbol{u}_{i+1},\boldsymbol{\sigma}_{i+1},\boldsymbol{\tau}_{i+1}) \qquad (2)\\ \vdash \cdots \vdash (q_n,\boldsymbol{u}_n,\boldsymbol{\sigma}_n,\boldsymbol{\tau}_n)\end{aligned}$$

be a run of $\mathcal{A}$ on $\boldsymbol{\sigma}$. Introducing a new initial state s with an additional transition $\mu(s,\epsilon,r) = (s_0,r)$, if necessary, we may assume that the first computation step $(q_0,\boldsymbol{u}_0,\boldsymbol{\sigma}_0,\boldsymbol{\tau}_0) \vdash (q_1,\boldsymbol{u}_1,\boldsymbol{\sigma}_1,\boldsymbol{\tau}_1)$ in the above run is by an ϵ-move.

Let q_i^N, $i = 0,\ldots,n$, be defined as follows.

- $q_i^N = q_i$, if no stack- or input-based reassignment has been made at the computation step $(q_i,\boldsymbol{u}_i,\boldsymbol{\sigma}_i,\boldsymbol{\tau}_i) \vdash (q_{i+1},\boldsymbol{u}_{i+1},\boldsymbol{\sigma}_{i+1},\boldsymbol{\tau}_{i+1})$;
- $q_i^N = q_i^\rho$, if a stack-based reassignment, but no input-based reassignment, has been made at the computation step (2); and
- $q_i^N = q_i^\pi$, if an input-based reassignment, but no stack-based reassignment, has been made at the computation step (2).

Then the computation step (2) of $\mathcal{A}$ is simulated by the computation step

$$(q_i^N,\boldsymbol{u}_i,\boldsymbol{\sigma}_i,\boldsymbol{\tau}_i) \vdash (q_{i+1}^N,\boldsymbol{u}_{i+1},\boldsymbol{\sigma}_{i+1},\boldsymbol{\tau}_{i+1})$$

of $\mathcal{A}^N$, if at least one of the stack- or input-based reassignments has not been made at (2), and is simulated by two computation steps

$$(q_i^\pi,\boldsymbol{u}_i,\boldsymbol{\sigma}_i,\boldsymbol{\tau}_i) \vdash (q_i^{\pi\rho},\boldsymbol{u}_i,\boldsymbol{\sigma}_i,\boldsymbol{\tau}_i) \vdash (q_{i+1}^N,\boldsymbol{u}_{i+1},\boldsymbol{\sigma}_{i+1},\boldsymbol{\tau}_{i+1})$$

of $\mathcal{A}^N$, otherwise. That is, if both the stack- and the input-based reassignment have been made at (2).

For the proof of the converse inclusion $L(\mathcal{A}^N) \subseteq L(\mathcal{A})$, let

$$\begin{aligned}(q_0^N,\boldsymbol{u}_0^N,\boldsymbol{\sigma}_0^N,\boldsymbol{\tau}_0^N) \vdash \cdots \vdash \\ (q_i^N,\boldsymbol{u}_i^N,\boldsymbol{\sigma}_i^N,\boldsymbol{\tau}_i^N) \vdash (q_{i+1}^N,\boldsymbol{u}_{i+1}^N,\boldsymbol{\sigma}_{i+1}^N,\boldsymbol{\tau}_{i+1}^N) \qquad (3)\\ \vdash \cdots \vdash (q_n^N,\boldsymbol{u}_n^N,\boldsymbol{\sigma}_n^N,\boldsymbol{\tau}_n^N),\end{aligned}$$

$q_i^N = q_i, q_i^\pi, q_i^\rho, q_i^{\pi\rho}$, be a run of $\mathcal{A}^N$ on $\boldsymbol{\sigma}$. We shall translate this run into a run of $\mathcal{A}$ on $\boldsymbol{\sigma}$ as follows.

- If $q_i^N, q_{i+1}^N \notin Q^{\pi\rho}$, then the computation step (3) of $\mathcal{A}^N$ is simulated by the computation step

$$(q_i, \boldsymbol{u}_i, \boldsymbol{\sigma}_i, \boldsymbol{\tau}_i) \vdash (q_{i+1}, \boldsymbol{u}_{i+1}, \boldsymbol{\sigma}_{i+1}, \boldsymbol{\tau}_{i+1})$$

  of $\mathcal{A}$.
- If $q_i^N = q_i^{\pi\rho}$, then the two consecutive computation steps

$$(q_{i-1}^N, \boldsymbol{u}_{i-1}^N, \boldsymbol{\sigma}_{i-1}^N, \boldsymbol{\tau}_{i-1}^N) \vdash (q_i^N, \boldsymbol{u}_i^N, \boldsymbol{\sigma}_i^N, \boldsymbol{\tau}_i^N) \vdash (q_{i+1}^N, \boldsymbol{u}_{i+1}^N, \boldsymbol{\sigma}_{i+1}^N, \boldsymbol{\tau}_{i+1}^N)$$

  of $\mathcal{A}^N$ are simulated by the computation step

$$(q_{i-1}, \boldsymbol{u}_{i-1}, \boldsymbol{\sigma}_{i-1}, \boldsymbol{\tau}_{i-1}) \vdash (q_{i+1}, \boldsymbol{u}_{i+1}, \boldsymbol{\sigma}_{i+1}, \boldsymbol{\tau}_{i+1})$$

  of $\mathcal{A}$.

Correctness of all simulations above immediately follows from the definition of $\mathcal{A}^N$. We omit the proof.

Combining Proposition 2 with Theorem 2 we obtain the corollary below.

Corollary 1. *The class of the DR-IAPDA languages is a proper subclass of the NR-IAPDA languages.*

We conclude this section with two properties of DR-IAPDA needed for the proof of Theorem 2.

To explain the first property we shall need the following notion. For an instantaneous description $\boldsymbol{id} = (q, \boldsymbol{v}, \boldsymbol{x}, \boldsymbol{\alpha})$, the set of symbols $[\boldsymbol{vx\alpha}]$ is called the active alphabet of $\boldsymbol{id}$. Due to the nature of deterministic reassignment, the instantaneous descriptions of DR-IAPDA possess the property of active alphabet monotony, i.e., the active alphabet of a subsequent instantaneous description is always a subset of the active alphabet of the preceding one.

Proposition 3. *Let $\mathcal{A}$ be a DR-IAPDA. If $(p, \boldsymbol{v}, \boldsymbol{x}, \boldsymbol{\alpha}) \vdash_{\mathcal{A}}^* (q, \boldsymbol{w}, \boldsymbol{y}, \boldsymbol{\beta})$, then $[\boldsymbol{wy\beta}] \subseteq [\boldsymbol{vx\alpha}]$.*

The proof of Proposition 3 is by a straightforward induction on the number of computation steps and is omitted.

The following observation will also be used in the sequel.

Proposition 4. (Cf. [7, Lemma 1].) *Let $\iota : \Sigma \to \Sigma$ be a permutation of Σ. Then for any DR-IAPDA $\mathcal{A}$,*

$$(p, \boldsymbol{v}, \boldsymbol{x}, \boldsymbol{\alpha}) \vdash_{\mathcal{A}}^* (q, \boldsymbol{w}, \boldsymbol{y}, \boldsymbol{\beta})$$

if and only if

$$(p, \iota(\boldsymbol{v}), \iota(\boldsymbol{x}), \iota(\boldsymbol{\alpha})) \vdash_{\mathcal{A}}^* (q, \iota(\boldsymbol{w}), \iota(\boldsymbol{y}), \iota(\boldsymbol{\beta})).$$

Again, the proof is by a straightforward induction on the number of computation steps and is omitted.

6 Proof of Theorem 2

The idea lying behind the proof of Theorem 2 is rather simple. Since a DR-IAPDA cannot guess δ at the beginning of the computation, it leaves it with the only option: to store $\boldsymbol{w}$ on the stack until it arrives at δ, and then compare each symbol of $\boldsymbol{w}$ with δ by popping them one by one out of the stack.[10] The problem is, that the stack is also needed to compare the number of \$s to the length of $\boldsymbol{w}$, and apparently, two of these tasks cannot be performed simultaneously. Essentially, this is the same intuition that leads us to the belief that $\{a^i b^i c^i : i = 1, 2 \ldots\}$ is not a context-free language, and that $\{a^i b^i c^j d^j : i, j = 1, 2 \ldots\}$ is not a linear context-free language, say. Nevertheless, usually it requires a significant effort to formalize a clear intuition, because proving a negative result is considerably harder than proving a positive one.

For the proof, we assume to the contrary, that there is an r-register DR-IAPDA $\mathcal{A} = \langle Q, s_0, \boldsymbol{u}, \pi, \rho, \mu \rangle$ that accepts L. In the following section we establish various properties $\mathcal{A}$ would possess, if it existed, which, as we show in Section 6.2, bring us to a contradiction, that will complete the proof.

6.1 Constraints on the Stack Content

In this section we prove a number of properties of $\mathcal{A}$ and introduce notation need for the proof of Theorem 2 in Section 6.2.

Let $\boldsymbol{z} = \$^{|\boldsymbol{w}|}\boldsymbol{w}\delta \in l$ be such that

- $|\boldsymbol{w}| > r$, where r is the number of registers of $\mathcal{A}$,
- none of the elements of Σ occurs $\boldsymbol{w}$ twice or more, and
- none of the symbols of $\boldsymbol{w}$ or δ occurs in the initial assignment $\boldsymbol{u}_0$ of $\mathcal{A}$.

Let $\boldsymbol{id}_0, \boldsymbol{id}_1, \ldots, \boldsymbol{id}_n$ be an accepting run of $\mathcal{A}$ on z, in which each instantaneous description $\boldsymbol{id}_i$, $i = 0, \ldots, n$, of $\mathcal{A}$ is of the form $(q_i, \boldsymbol{u}_i, \boldsymbol{z}_i, \boldsymbol{\alpha}_i)$, where

- q_i is the current state of $\mathcal{A}$,
- $\boldsymbol{u}_i$ is the current register assignment,
- $\boldsymbol{z}_i$ is the remaining input (suffix of $\boldsymbol{z}$), and
- $\boldsymbol{\alpha}_i$ is the content of the stack read top down.

In particular, $\boldsymbol{id}_0 = (s_0, \boldsymbol{u}, \boldsymbol{z}, u_r)$, and $\boldsymbol{z}_n = \boldsymbol{\alpha}_n = \epsilon$.

We shall need the following notation. For each $i = 0, \ldots, n$, such that $\boldsymbol{z}_i \neq \epsilon$, the suffix $\boldsymbol{w}_i$ of $\$^{|\boldsymbol{w}|}\boldsymbol{w}$ is defined by $\boldsymbol{z}_i = \boldsymbol{w}_i\delta$.

Lemma 1. *Let $\boldsymbol{id}_i$, $i = 0, \ldots, n$, be as above. Then for each i such that $\boldsymbol{z}_i \neq \epsilon$, $[\boldsymbol{w}] \subseteq [\boldsymbol{u}_i \boldsymbol{z}_i \boldsymbol{\alpha}_i]$.*

Proof. Assume to the contrary that for some $i = 0, \ldots, n$, $\boldsymbol{z}_i \neq \epsilon$ and $[\boldsymbol{w}] \not\subseteq [\boldsymbol{u}_i \boldsymbol{z}_i \boldsymbol{\alpha}_i]$, and let $\delta' \in [\boldsymbol{w}]$ be such that

$$\delta' \notin [\boldsymbol{u}_i \boldsymbol{z}_i \boldsymbol{\alpha}_i].$$

[10] This intuition is formalized by Lemma 1.

Since $\boldsymbol{z}_i = \boldsymbol{w}_i\delta$, where $\boldsymbol{w}_i$ is a suffix of $\$^{|\boldsymbol{w}|}\boldsymbol{w}$, $\boldsymbol{id}_0 \vdash^* \boldsymbol{id}_i$ implies

$$(s_0, \boldsymbol{u}_0, \$^{|\boldsymbol{w}|}\boldsymbol{w}, u_r) \vdash^* (q_i, \boldsymbol{u}_i, \boldsymbol{w}_i, \boldsymbol{\alpha}_i). \tag{4}$$

Let the permutation ι switch between δ and δ', and fix all other symbols of Σ. Then, applying Proposition 4 to (4), we obtain

$$(s_0, \iota(\boldsymbol{u}_0), \iota(\$^{|\boldsymbol{w}|}\boldsymbol{w}), \iota(u_r)) \vdash^* (q_i, \iota(\boldsymbol{u}_i), \iota(\boldsymbol{w}_i), \iota(\boldsymbol{\alpha}_i)) \tag{5}$$

Since, by the definition of $\boldsymbol{w}$, $\delta, \delta' \notin [\boldsymbol{u}] \cup \{\$\}$ and $|\iota(\boldsymbol{w})| = |\boldsymbol{w}|$,

$$(s_0, \iota(\boldsymbol{u}_0), \iota(\$^{|\boldsymbol{w}|}\boldsymbol{w}), \iota(u_r)) = (s_0, \boldsymbol{u}_0, \$^{|\iota(\boldsymbol{w})|}\iota(\boldsymbol{w}), u_r). \tag{6}$$

Also, by Proposition 3 (and the definitions of $\boldsymbol{w}$, δ, and δ'),

$$(q_i, \iota(\boldsymbol{u}_i), \iota(\boldsymbol{w}_i), \iota(\boldsymbol{\alpha}_i)) = (q_i, \boldsymbol{u}_i, \boldsymbol{w}_i, \boldsymbol{\alpha}_i). \tag{7}$$

Thus, by (5), (6), and (7),

$$(s_0, \boldsymbol{u}_0, \$^{\iota(|\boldsymbol{w}|)}\iota(\boldsymbol{w}), u_r) \vdash^* (q_i, \boldsymbol{u}_i, \boldsymbol{w}_i, \boldsymbol{\alpha}_i), \tag{8}$$

where $\iota(\boldsymbol{w})$ is obtained from $\boldsymbol{w}$ by replacing δ' with δ.

Now, appending δ to $\$^{\iota(|\boldsymbol{w}|)}\iota(\boldsymbol{w})$ and $\boldsymbol{w}_i$ in (8) results in

$$(s_0, \boldsymbol{u}_0, \$^{|\iota(\boldsymbol{w})|}\iota(\boldsymbol{w})\delta, u_r) \vdash^* (q_i, \boldsymbol{u}_i, \boldsymbol{z}_i, \boldsymbol{\alpha}_i) = \boldsymbol{id}_i \vdash^* \boldsymbol{id}_n, \tag{9}$$

where $\boldsymbol{id}_i \vdash^* \boldsymbol{id}_n$ is by the definition of $\boldsymbol{id}_0, \ldots, \boldsymbol{id}_n$. The first instantaneous description in (9) is, in fact, the initial instantaneous description of $\mathcal{A}$ on the input $\$^{|\iota(\boldsymbol{w})|}\iota(\boldsymbol{w})\delta \notin L$, and since, by definition, $\boldsymbol{id}_n$ is an accepting instantaneous description, $\$^{|\iota(\boldsymbol{w})|}\iota(\boldsymbol{w})\delta \in L(\mathcal{A})$, in spite of $\delta \in [\iota(\boldsymbol{w})]$. This contradicts the equality $L = L(\mathcal{A})$, which completes the proof.

The following series of lemmas formalizes the intuition that somewhere between reading the first and the rth symbols of $\boldsymbol{w}$, $\mathcal{A}$ enters an instantaneous description whose bottom stack symbol remains intact until the automaton arrives at the end of the input δ.

Let $\boldsymbol{w} = \boldsymbol{u}'\boldsymbol{u}''$, where $|\boldsymbol{u}'| = r$, and let

$$i_0 = \max\{i : \boldsymbol{z}_i = \boldsymbol{u}''\delta\}. \tag{10}$$

That is, at the computation step $\boldsymbol{id}_{i_0} \vdash \boldsymbol{id}_{i_0+1}$, $\mathcal{A}$ arrives at the $(r+1)$st symbol of $\boldsymbol{w}$.

Remark 2. Note that, by the definition of i_0, $\boldsymbol{w}_{i_0}$ is a suffix of $\boldsymbol{w}$ and $|\boldsymbol{w}| - |\boldsymbol{w}_{i_0}| = r$.

Lemma 2. $[\boldsymbol{u}'] \cap [\boldsymbol{\alpha}_{i_0}] \neq \emptyset$.

Proof. Assume the contrary that $[\boldsymbol{u}'] \cap [\boldsymbol{\alpha}_{i_0}] = \emptyset$. Then, since $[\boldsymbol{u}'] \cap [\boldsymbol{u}''] = \emptyset$, by Propositions 3 and Lemma 1, all symbols occurring in $\boldsymbol{u}'$ are stored in the automaton registers, i.e. $[\boldsymbol{u}_{i_0}] = [\boldsymbol{u}']$. Since $[\boldsymbol{\alpha}_{i_0}] \cap [\boldsymbol{u}'] = \emptyset$, to perform the next computation step $\boldsymbol{id}_{i_0} \vdash \boldsymbol{id}_{i_0+1}$, $\mathcal{A}$ must make the stack-based reassignment. Namely, it has to reassign one of the registers with the first symbol of $\boldsymbol{\alpha}_{i_0}$. However, the reassignment would cause $\mathcal{A}$ to "loose" a symbol of $\boldsymbol{u}'$ stored in one of its registers, in contradiction with Lemma 1, because $|\boldsymbol{w}| > r$.

Let $\boldsymbol{\alpha}$ be the shortest suffix of $\boldsymbol{\alpha}_{i_0}$ that contains a symbol from $[\boldsymbol{u}']$. In particular, $\boldsymbol{\alpha} \neq \epsilon$.

Remark 3. By definition, only one symbol from $[\boldsymbol{u}']$ occurs in $\boldsymbol{\alpha}$, and this symbol is the first symbol of the latter.

Lemma 3. *$\boldsymbol{\alpha}_{i_0} = \boldsymbol{\beta}_{i_0}\boldsymbol{\alpha}$, where $\boldsymbol{\beta}_{i_0} \neq \epsilon$.*

Proof. Assume to the contrary that $\boldsymbol{\beta}_{i_0} = \epsilon$. Then the first symbol of $\boldsymbol{\alpha}$ is the only one from $\boldsymbol{u}'$ occurring in $\boldsymbol{\alpha}_{i_0}$. Thus, by Lemma 1, the other $r-1$ symbols of $\boldsymbol{u}'$ must be stored in $r-1$ registers of $\mathcal{A}$. Consequently, by the same lemma, in the stack-based reassignment before the next computation step, the only register that may be reassigned with the first symbol of $\boldsymbol{\alpha}$, or already contains it, is the remaining register. Also, after this reassignment, the content of the registers coincides with $[\boldsymbol{u}']$. Since, by the definition of i_0, the computation step $\boldsymbol{id}_{i_0} \vdash \boldsymbol{id}_{i_0+1}$ is not an ϵ-move, one of the registers has to be reassigned with the first symbol of $\boldsymbol{u}''$ in the input-based reassignment.

This register cannot be the one containing the first symbol of $\boldsymbol{\alpha}$, because in this case none of the register content would be equal to the top stack symbol, and, therefore, $\mathcal{A}$ would not be able to perform the computation step. Consequently, one of the symbols of $[\boldsymbol{u}']$ is erased from the registers, in contradiction with Lemma 1. That is, $\boldsymbol{\beta}_{i_0} \neq \epsilon$ and the proof is complete.

Lemma 4. *If $|\boldsymbol{w}| > r+1$, then $\boldsymbol{\alpha}_{i_0+1} = \boldsymbol{\beta}_{i_0+1}\boldsymbol{\alpha}$, where $\boldsymbol{\beta}_{i_0+1} \neq \epsilon$.*

Proof. By definition, $\boldsymbol{\alpha}_{i_0+1}$ is obtained from $\boldsymbol{\alpha}_{i_0}$ by replacing its first symbol with a word that consists of symbols from $[\boldsymbol{u}_i]$. Since, by Lemma 3, $\boldsymbol{\alpha}_{i_0} = \boldsymbol{\beta}_{i_0}\boldsymbol{\alpha}$, where $\boldsymbol{\beta}_i \neq \epsilon$, $\boldsymbol{\alpha}_{i_0+1}$ is of the form $\boldsymbol{\beta}_{i_0+1}\boldsymbol{\alpha}$, and we only need to show that $\boldsymbol{\beta}_{i_0+1} \neq \epsilon$. If $|\boldsymbol{\beta}_{i_0}| > 1$, this is immediate, because the stack hight may decrease by at most one in one computation step. Otherwise, $\boldsymbol{\beta}_{i_0} = \alpha \in \Sigma$ and we argue that $\alpha \in [\boldsymbol{u}']$.

To show this, assume to the contrary that $\alpha \notin [\boldsymbol{u}']$. Then there is only one register which α may be assigned to without loss of "valuable" symbols. Applying one more time the argument from the proof of Lemma 3, we see that this case is impossible.

So, $\alpha \in [\boldsymbol{u}']$ and we have to show that the word $\boldsymbol{\beta}_{i_0+1}$ pushed into the stack in the computation step $\boldsymbol{id}_{i_0} \vdash \boldsymbol{id}_{i_0+1}$ is not empty.

For the proof, assume to the contrary that $\boldsymbol{\beta}_{i_0+1} = \epsilon$. Then, after the computation step $\boldsymbol{id}_{i_0} \vdash \boldsymbol{id}_{i_0+1}$ is complete, r of the first $r+1$ symbols of $\boldsymbol{w}$ are stored in the r registers of $\mathcal{A}$ (and occur only in the registers), and the other (of the $r+1$ symbols, that is the first symbol of $\boldsymbol{\alpha}$) is at the top of the stack. Therefore, the stack-based reassignment at the next computation step would erase a symbol stored in the registers, in contradiction with Lemma 1 for $\boldsymbol{id}_{i+1}$. This is because, by the lemma prerequisite $|\boldsymbol{w}| > r+1$, the $(r+2)$nd symbol of $\boldsymbol{w}$ is not δ.

Lemma 5. *If $|\boldsymbol{w}| > r+1$ and $|\boldsymbol{w}| - |\boldsymbol{w}_i| = r+1$,[11] then $\boldsymbol{\alpha}_i = \boldsymbol{\beta}_i\boldsymbol{\alpha}$, where $\boldsymbol{\beta}_i \neq \epsilon$.*

[11] That is, $\boldsymbol{id}_{i_0+1} \vdash^* \boldsymbol{id}_i$ by ϵ-moves only.

The proof of the lemma is by induction on i. The basis, $i = i_0 + 1$, is Lemma 4, and the induction step is word by word repetition of the proof of Lemma 4 and is left to the reader.

Lemma 6. *If* $|\boldsymbol{w}| - |\boldsymbol{w}_i| \geq r + 2$, *then* $\boldsymbol{\alpha}_i = \boldsymbol{\beta}_i \boldsymbol{\alpha}$, *where* $\boldsymbol{\beta}_i \neq \epsilon$.

Proof. The proof is by induction on i. For the basis, let i be such that $|\boldsymbol{w}| - |\boldsymbol{w}_{i-1}| = r+1$, but $|\boldsymbol{w}| - |\boldsymbol{w}_i| = r+2$. That is, the number of symbols read from $\boldsymbol{w}$ is exactly $r+2$. By Lemma 1, each of these symbols occurs either in the stack or in the registers. Namely,

1. r of the symbols are stored in the registers;
2. one symbol occurs in $\boldsymbol{\alpha}$, because, by Lemma 5, $\boldsymbol{\beta}_{i-1} \neq \epsilon$, which implies that $\boldsymbol{\alpha}$ remains intact at this computation step; and
3. one symbol occurs in $\boldsymbol{\beta}_i$.

That is, $\boldsymbol{\beta}_i \neq \epsilon$.

The induction step is treated similarly to the basis. We just replace "$r + 2$" with "at least $r + 2$" and replace "one" in clause 3 with "at least one".

Let

$$i_1 = \max\{i < i_0 : \boldsymbol{\alpha} \text{ is not a proper suffix of } \boldsymbol{\alpha}_{i_1}\} + 1, \tag{11}$$

where i_0 is defined by (10), and let

$$M = \max\{\ell : (p, j_1 j_2 \dots j_\ell) \in \mu(q; k, j),\ p, q \in Q \text{ and } k, j, j_1, \dots, j_\ell = 1, \dots, r\}.$$

That is, $\boldsymbol{id}_{i_1-1}$ is the last instantaneous description before $\boldsymbol{id}_{i_0}$, whose stack content does not contain $\boldsymbol{\alpha}$ as a proper suffix and M is the maximum length of a word pushed into the stack in one computation step.

Combining Lemmas 3, 4, 5, and 6, we obtain Lemma 7 below.

Lemma 7

1. $|\boldsymbol{w}| - |\boldsymbol{w}_{i_1}| \leq r$.
2. $\boldsymbol{\alpha}_{i_1} = \boldsymbol{\beta}_{i_1} \boldsymbol{\alpha}$, *where* $0 < |\boldsymbol{\beta}_{i_1}| \leq M$.
3. *For each* $i \geq i_1$ *such that* $\boldsymbol{z}_j \neq \epsilon$, $\boldsymbol{\alpha}_i = \boldsymbol{\beta}_i \boldsymbol{\alpha}$, *where* $\boldsymbol{\beta}_i \neq \epsilon$.

Proof. By definition, $i_1 \leq i_0$, and clause 1 follows from the definition of i_0 according to which $|\boldsymbol{w}| - |\boldsymbol{w}_{i_0}| = r$.

Since $\boldsymbol{\alpha}$ is a proper suffix of $\boldsymbol{\alpha}_{i_1}$, $\boldsymbol{\beta}_{i_1} \neq \epsilon$. To prove that $|\boldsymbol{\beta}_{i_1}| \leq M$, we proceed as follows. Let $\boldsymbol{\alpha}_{i_1-1} = \alpha\boldsymbol{\alpha}'$. Then, $\boldsymbol{\alpha}_{i_1}$ is obtained from $\boldsymbol{\alpha}_{i_1-1}$ by replacing α with some word $\boldsymbol{\alpha}''$, i.e.

$$\boldsymbol{\alpha}_{i_1} = \boldsymbol{\alpha}''\boldsymbol{\alpha}'. \tag{12}$$

By the definition of M,

$$|\boldsymbol{\alpha}''| \leq M, \tag{13}$$

and, since $\boldsymbol{\alpha}$ is not a proper suffix of $\boldsymbol{\alpha}_{i_1-1}$, $\boldsymbol{\alpha}'$ is a suffix of $\boldsymbol{\alpha}$. Thus, by (12), $\boldsymbol{\beta}_{i_1}$ is a proper suffix of $\boldsymbol{\alpha}''$, and clause 2 of the lemma follows from (13).

Finally, clause 3 follows immediately from Lemmas 3, 4, 5, and 6, and the definition of $\boldsymbol{id}_{i_1}$, according to which $\boldsymbol{\alpha}$ is a proper suffix of all $\boldsymbol{\alpha}_i$ for $i_1 \leq i < i_0$.

So, by Lemma 7, we have located the instantaneous description $\boldsymbol{id}_{i_1}$, with $\boldsymbol{\alpha}$ at the bottom of the stack which remains intact until the entire $\boldsymbol{w}$ is read. The following corollary to Lemma 7 shows that, actually, it remains intact until the entire input $\boldsymbol{z}$ is read. Namely, let

$$i_2 = \max\{i : \text{ for all } i',\ i_0 \le i' \le i,\ \boldsymbol{\alpha} \text{ is a suffix of } \boldsymbol{\alpha}_{i'}\}. \tag{14}$$

Corollary 2. *If* $|\boldsymbol{w}| > r+1$, *then* $\boldsymbol{z}_{i_2} = \epsilon$.

Proof. Let i be the last computation step before reading δ. Then, by clause 3 of Lemma 7, $\boldsymbol{\alpha}_i = \boldsymbol{\beta}_i\boldsymbol{\alpha}$, where $\boldsymbol{\beta}_i \neq \epsilon$. Since at most one symbol may be removed from the stack in one computation step, $\boldsymbol{\alpha}$ is a suffix of $\boldsymbol{\alpha}_{i+1}$. Therefore, $i_2 \ge i+1$, and the corollary follows.

6.2 The Final Match

Now, to arrive at the desired contradiction and prove Theorem 2, we consider two words, $\boldsymbol{z}' = \$^{|\boldsymbol{w}'|}\boldsymbol{w}'\delta$ and $\boldsymbol{z}'' = \$^{|\boldsymbol{w}''|}\boldsymbol{w}''\delta$ in L which satisfy the following conditions.

1. $|\boldsymbol{w}'|, |\boldsymbol{w}''| > r+1$.
2. $|\boldsymbol{w}'| \neq |\boldsymbol{w}''|$.
3. None of the symbols of Σ occur in $\boldsymbol{w}'$ or $\boldsymbol{w}''$ twice or more, and none of the symbols of $\boldsymbol{w}'$, $\boldsymbol{w}''$ or δ occurs in the initial assignment of $\mathcal{A}$.
4. The prefixes of $\boldsymbol{w}'$ and $\boldsymbol{w}''$ of length $r+1$ are equal to the same word, $\sigma_1\sigma_2\cdots\sigma_{r+1}$, say. That is,

$$\boldsymbol{w}' = \sigma_1\sigma_2\cdots\sigma_{r+1}\boldsymbol{v}' \tag{15}$$

and

$$\boldsymbol{w}'' = \sigma_1\sigma_2\cdots\sigma_{r+1}\boldsymbol{v}'', \tag{16}$$

for the appropriate $\boldsymbol{v}', \boldsymbol{v}'' \in \Sigma^*$.

Let $\boldsymbol{id}'_0, \ldots, \boldsymbol{id}'_{n'}$, $\boldsymbol{id}'_i = (q'_i, \boldsymbol{u}'_i, \boldsymbol{z}'_i, \boldsymbol{\alpha}'_i)$, $i = 0, \ldots, n'$, and $\boldsymbol{id}''_0, \ldots, \boldsymbol{id}''_{n''}$ $\boldsymbol{id}''_i = (q''_i, \boldsymbol{u}''_i, \boldsymbol{z}''_i, \boldsymbol{\alpha}''_i)$, $i = 0, \ldots, n''$, be accepting runs of $\mathcal{A}$ on $\boldsymbol{z}'$ and $\boldsymbol{z}''$, respectively. Let i'_1, i'_2, i''_1, and i''_2, be the positive integers provided by Lemma 7 and Corollary 2 for $\boldsymbol{w}'$ and $\boldsymbol{w}''$, respectively. That is, we use the primed and double primed versions of the constants and parameters defined in Section 6.1.

We shall also need the following definitions and notation.

- The alphabet $[\boldsymbol{u}] \cup \{\sigma_1, \sigma_2, \ldots, \sigma_{r+1}\} \cup \{\$\}$ is denoted by Δ.
- Let $\# \notin \Sigma$. We define the function $\boldsymbol{p} : \Sigma \to \Delta \cup \{\#\}$ by

$$\boldsymbol{p}(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Delta \\ \#, & \text{otherwise} \end{cases},$$

and, for a word $\boldsymbol{v} = v_1 \cdots v_m \in \Sigma^*$, we define the word $\boldsymbol{p}(\boldsymbol{v}) \in (\Delta \cup \{\#\})^*$ by

$$\boldsymbol{p}(\boldsymbol{v}) = \boldsymbol{p}(v_1)\cdots\boldsymbol{p}(v_m).$$

That is, $\boldsymbol{p}(\boldsymbol{v})$ results from $\boldsymbol{v}$ in replacing all its symbols which are not in Δ with $\#$.

– The positive integers r' and r'' are defined by $r' = |\boldsymbol{w}'| - |\boldsymbol{w}'_{i'_1}|$ and $r'' = |\boldsymbol{w}''| - |\boldsymbol{w}''_{i''_1}|$. That is, r' and r'' are the number of symbols read from $\boldsymbol{w}'$ and $\boldsymbol{w}''$ up to the computation steps $\boldsymbol{id}'_{i'_1} \vdash \boldsymbol{id}'_{i'_1+1}$ and $\boldsymbol{id}''_{i''_1} \vdash \boldsymbol{id}''_{i''_1+1}$, respectively.
  Recall that, by clause 1 of Lemma 7, $r', r'' \leq r$.
– For a word $\boldsymbol{v} \in \Sigma^*$, $\boldsymbol{f}(\boldsymbol{v})$ is the first symbol of $\boldsymbol{v}$, if $\boldsymbol{v} =\neq \epsilon$ and is ϵ, otherwise. That is,

$$\boldsymbol{f}(\boldsymbol{v}) = \begin{cases} v_1, \text{ if } \boldsymbol{v} = v_1 \cdots v_m \neq \epsilon \\ \epsilon, \quad \text{if } \boldsymbol{v} = \epsilon \end{cases} .$$

– Finally, the tuples

$$t_{\boldsymbol{z}'}, t_{\boldsymbol{z}''} \in Q \times \Delta^r \times \{1, \ldots, r\} \times \Delta^{\leq M} \times Q \times (\Delta \cup \{\#\})^r \times \Delta$$

are defined by

$$t_{\boldsymbol{z}'} = (q'_{i'_1}, u'_{i'_1}, r', \boldsymbol{\beta}'_{i'_1}, q'_{i'_2}, \boldsymbol{p}(\boldsymbol{u}'_{i'_2}), \boldsymbol{f}(\boldsymbol{\alpha}'))$$

and

$$t_{\boldsymbol{z}''} = (q''_{i''_1}, u''_{i''_1}, r'', \boldsymbol{\beta}''_{i''_2}, q''_{i''_2}, \boldsymbol{p}(\boldsymbol{u}''_{i''_2}), \boldsymbol{f}(\boldsymbol{\alpha}'')).$$

Since the set

$$Q \times \Delta^r \times \{1, \ldots, r\} \times \Delta^{\leq M} \times Q \times (\Delta \cup \{\#\})^r \times \Delta$$

is finite and Σ is infinite, there are words $\boldsymbol{w}'$ and $\boldsymbol{w}''$ satisfying conditions 1 – 4 in the beginning of this section such that $t_{\boldsymbol{z}'} = t_{\boldsymbol{z}''}$. That is,

$$(q'_{i'_1}, u'_{i'_1}, r', \boldsymbol{\beta}'_{i'_1}, q'_{i'_2}, \boldsymbol{p}(\boldsymbol{u}'_{i'_2}), \boldsymbol{f}(\boldsymbol{\alpha}')) = (q''_{i''_1}, u''_{i''_1}, r'', \boldsymbol{\beta}''_{i''_2}, q''_{i''_2}, \boldsymbol{p}(\boldsymbol{u}''_{i''_2}), \boldsymbol{f}(\boldsymbol{\alpha}'')). \quad (17)$$

Let $\boldsymbol{u}'_{i'_2} = u'_{i'_2,1} \cdots u'_{i'_2,r}$, $\boldsymbol{u}''_{i''_2} = u''_{i''_2,1} \cdots u''_{i''_2,r}$. and let ι be the permutation of Σ that is defined as follows.

$$\iota(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \notin ([\boldsymbol{u}'_{i'_2}] \cup [\boldsymbol{u}''_{i''_2}]) \setminus \Delta \\ u''_{i''_2,k}, & \text{if } \sigma = u'_{i'_2,k} \in [\boldsymbol{u}'_{i'_2}] \setminus \Delta, \ k = 1, \ldots, r \\ u'_{i'_2,k}, & \text{if } \sigma = u''_{i''_2,k} \in [\boldsymbol{u}''_{i''_2}] \setminus \Delta, \ k = 1, \ldots, r \end{cases} .$$

That is, ι switches $u'_{i'_2,k}$ and $u''_{i''_2,k}$, $k = 1, \ldots, r$, if both are in $([\boldsymbol{u}'_{i'_2}] \cup [\boldsymbol{u}''_{i''_2}]) \setminus \Delta$. Since $\boldsymbol{u}'_{i'_2}, \boldsymbol{u}''_{i''_2} \in \Sigma^{r\mp}$ and $\boldsymbol{p}(\boldsymbol{u}'_{i'_2}) = \boldsymbol{p}(\boldsymbol{u}''_{i''_2})$, ι is well-defined.

Let

$$\boldsymbol{z} = \$^{|\boldsymbol{w}'|}\iota(\boldsymbol{w}'')\delta. \quad (18)$$

Since $|\iota(\boldsymbol{w}'')| = |\boldsymbol{w}''|$ and $|\boldsymbol{w}'| \neq |\boldsymbol{w}''|$, $\boldsymbol{z} \notin L$. However, we shall construct an accepting run of $\mathcal{A}$ on $\boldsymbol{z}$, in contradiction with our assumption $L(\mathcal{A}) = L$. To construct this run we need one more bit of notation.

Let $\boldsymbol{v}_i$, $i = 0, \ldots, i'_1$, be defined by

$$\boldsymbol{z}'_i = \boldsymbol{v}_i \boldsymbol{v}' \delta, \quad (19)$$

where $\boldsymbol{v}'$ is defined by (15), and let the instantaneous descriptions $\boldsymbol{id}^{(a)}_i$, $i = 0, \ldots, i'_1$, $\boldsymbol{id}^{(b)}_i$, $i = i''_1, \ldots, i''_2$, and $\boldsymbol{id}^{(c)}_i$, $i = i'_2, \ldots, n'$ be defined as follows.

(a) $\boldsymbol{id}_i^{(a)} = (q_i', \boldsymbol{u}_i', \boldsymbol{v}_i\iota(\boldsymbol{v}'')\delta, \boldsymbol{\alpha}_i')$, where $\boldsymbol{v}_i$s and $\boldsymbol{v}''$ are defined by (19) and (16), respectively.
(b) $\boldsymbol{id}_i^{(b)} = (q_i'', \iota(\boldsymbol{u}_i''), \iota(\boldsymbol{z}_i''), \iota(\boldsymbol{\beta}_i'')\boldsymbol{\alpha}')$.
(c) $\boldsymbol{id}_i^{(c)} = (q_i', \boldsymbol{u}_i', \epsilon, \boldsymbol{\alpha}_i')$.

Theorem 2 immediately follows from Lemma 8 below.

Lemma 8

$$\underbrace{\boldsymbol{id}_0^{(a)} \vdash \cdots \vdash \overbrace{\boldsymbol{id}_{i_1'}^{(a)}}^{(ab)}}_{(a)} = \underbrace{\boldsymbol{id}_{i_1''}^{(b)} \vdash \boldsymbol{id}_{i_1''+1}^{(b)} \vdash \cdots \vdash \overbrace{\boldsymbol{id}_{i_2''}^{(b)}}^{(bc)}}_{(b)} = \underbrace{\boldsymbol{id}_{i_2'}^{(c)} \vdash \boldsymbol{id}_{i_2'+1}^{(c)} \vdash \cdots \vdash \boldsymbol{id}_{n'}^{(c)}}_{(c)} \tag{20}$$

Indeed, by Lemma 8 (and the definition of the instantaneous descriptions in it, of course), (20) is an accepting run of $\mathcal{A}$ on $\boldsymbol{z}$. That is, $\boldsymbol{z} \in L$, which contradicts the definition of L, see (1).

Proof of Lemma 8.

Proof of part (a). By the definition of $\boldsymbol{id}_i'$s, (11), and (19),

$$(q_0', \boldsymbol{u}_0', \boldsymbol{v}_0\boldsymbol{v}'\delta, \boldsymbol{\alpha}_0') \vdash \cdots \vdash (q_{i_1'}', \boldsymbol{u}_{i_1'}', \boldsymbol{v}_{i_1'}\boldsymbol{v}'\delta, \boldsymbol{\alpha}_{i_1'}'),$$

implying

$$(q_0', \boldsymbol{u}_0', \boldsymbol{v}_0, \boldsymbol{\alpha}_0') \vdash \cdots \vdash (q_{i_1'}', \boldsymbol{u}_{i_1'}', \boldsymbol{v}_{i_1'}, \boldsymbol{\alpha}_{i_1'}'),$$

which, in turn, implies

$$(q_0', \boldsymbol{u}_0', \boldsymbol{v}_0\iota(\boldsymbol{v}'')\delta, \boldsymbol{\alpha}_0') \vdash \cdots \vdash (q_{i_1'}', \boldsymbol{u}_{i_1'}', \boldsymbol{v}_{i_1'}\iota(\boldsymbol{v}'')\delta, \boldsymbol{\alpha}_{i_1'}'),$$

and part (a) of the lemma follows from the definition of $\boldsymbol{id}_i^{(a)}$s.

Proof of part (ab). By definition of $\boldsymbol{id}_{i_1'}^{(a)}$ and $\boldsymbol{id}_{i_1''}^{(b)}$, we have to show that

$$(q_{i_1'}', \boldsymbol{u}_{i_1'}', \boldsymbol{v}_{i_1'}\iota(\boldsymbol{v}'')\delta, \boldsymbol{\alpha}_{i_1'}') = (q_{i_1''}'', \iota(\boldsymbol{u}_{i_1''}''), \iota(\boldsymbol{z}_{i_1''}''), \iota(\boldsymbol{\beta}_{i_1''}'')\boldsymbol{\alpha}').$$

- $q_{i_1'}' = q_{i_1''}''$ immediately follows from (17).
- The proof of $\boldsymbol{u}_{i_1'}' = \iota(\boldsymbol{u}_{i_1''}'')$ is as follows. By Proposition 3, $\boldsymbol{u}_{i_1''}'' \in \Delta^*$. Therefore, by the definition of ι, $\iota(\boldsymbol{u}_{i_1''}'') = \boldsymbol{u}_{i_1''}''$, and again, the equality follows from (17).
- For the proof of

$$\boldsymbol{v}_{i_1'}\iota(\boldsymbol{v}'')\delta = \iota(\boldsymbol{z}_{i_1''}''), \tag{21}$$

we observe that, by (15),

$$\boldsymbol{z}_{i_1'}' = \boldsymbol{v}_{i_1'}\boldsymbol{v}'\delta = \sigma_{r'+1}\cdots\sigma_{r+1}\boldsymbol{v}'\delta,$$

implying $\boldsymbol{v}_{i'_1} = \sigma_{r'+1} \cdots \sigma_{r+1}$. Since

$$\boldsymbol{z}''_{i''_1} = \boldsymbol{w}''_{i''_1}\delta = \sigma_{r''+1} \cdots \sigma_{r+1}\boldsymbol{v}''\delta$$

and, by (17), $r' = r''$, we have

$$\boldsymbol{z}''_{i''_1} = \boldsymbol{v}_{i'_1}\boldsymbol{v}''\delta. \tag{22}$$

Now, applying ι to both sides of (22) we obtain (21).

– The proof of the last equality $\boldsymbol{\alpha}'_{i'_1} = \iota(\boldsymbol{\beta}''_{i''_1})\boldsymbol{\alpha}'$ is equally easy. Namely,

$$\boldsymbol{\alpha}'_{i'_1} = \boldsymbol{\beta}'_{i'_1}\boldsymbol{\alpha}' = \boldsymbol{\beta}''_{i''_1}\boldsymbol{\alpha}', \tag{23}$$

where the second equality follows from (17). Since, by Proposition 3, $\boldsymbol{\beta}''_{i''_1} \in \Delta^*$, by the definition of ι, $\iota(\boldsymbol{\beta}''_{i''_1}) = \boldsymbol{\beta}''_{i''_1}$, which, together with (23), completes the proof.

Proof of part (b). By the definition of $\boldsymbol{id}''_i$s,

$$(q''_{i''_1}, \boldsymbol{u}''_{i''_1}, \boldsymbol{z}''_{i''_1}, \boldsymbol{\beta}''_{i''_1}\boldsymbol{\alpha}'') \vdash \cdots \vdash (q''_{i''_2}, \boldsymbol{u}''_{i''_2}, \boldsymbol{z}''_{i''_2}, \boldsymbol{\beta}''_{i''_2}\boldsymbol{\alpha}''),$$

implying

$$(q''_{i''_1}, \boldsymbol{u}''_{i''_1}, \boldsymbol{z}''_{i''_1}, \boldsymbol{\beta}''_{i''_1}) \vdash \cdots \vdash (q''_{i''_2}, \boldsymbol{u}''_{i''_2}, \boldsymbol{z}''_{i''_2}, \boldsymbol{\beta}''_{i''_2}),$$

because, by clause 3 of Lemma 7 and (14), $\boldsymbol{\beta}''_{i''} \neq \epsilon$, $i = i''_1, \ldots, i''_2 - 1$. Therefore, by Proposition 4,

$$(q''_{i''_1}, \iota(\boldsymbol{u}''_{i''_1}), \iota(\boldsymbol{z}''_{i''_1}), \iota(\boldsymbol{\beta}''_{i''_1}) \vdash \cdots \vdash (q''_{i''_2}, \iota(\boldsymbol{u}''_{i''_2}), \iota(\boldsymbol{z}''_{i''_2}), \iota(\boldsymbol{\beta}''_{i''_2})),$$

implying

$$(q''_{i''_1}, \iota(\boldsymbol{u}''_{i''_1}), \iota(\boldsymbol{z}''_{i''_1}), \iota(\boldsymbol{\beta}''_{i''_1})\boldsymbol{\alpha}') \vdash \cdots \vdash (q''_{i''_2}, \iota(\boldsymbol{u}''_{i''_2}), \iota(\boldsymbol{z}''_{i''_2}), \iota(\boldsymbol{\beta}''_{i''_2})\boldsymbol{\alpha}'),$$

and part (b) of the lemma follows from the definition of $\boldsymbol{id}^{(b)}_i$s.

Proof of part (bc). By the definition of $\boldsymbol{id}^{(b)}_{i''_2}$ and $\boldsymbol{id}^{(c)}_{i'_2}$, we have to show that

$$(q''_{i''_2}, \iota(\boldsymbol{u}''_{i''_2}), \boldsymbol{z}''_{i''_2}, \iota(\boldsymbol{\beta}''_{i''_2}), \boldsymbol{\alpha}') = (q'_{i'_2}, \boldsymbol{u}'_{i'_2}, \epsilon, \boldsymbol{\alpha}'_{i'_2}).$$

– $q'_{i'_2} = q''_{i''_2}$ immediately follows from (17).
– $\boldsymbol{u}'_{i'_2} = \iota(\boldsymbol{u}''_{i''_2})$ immediately follows from the definition of ι.
– The equality $\boldsymbol{z}''_{i''_2} = \epsilon$ is provided by Corollary 2.
– Finally, the equality $\boldsymbol{\alpha}' = \boldsymbol{\alpha}'_{i'_2}$ is by (14).

Proof of part (c). This part of the lemma follows immediately from the definition of $\boldsymbol{id}'_i$s and $\boldsymbol{id}^{(c)}_i$s, because, by Corollary 2, for $i = i'_2, \ldots, n'$, $\boldsymbol{z}'_i = \epsilon$.

References

1. Bojanczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006), pp. 7–16. IEEE Computer Society, Los Alamitos (2006)
2. Bray, T., Paoli, J., Sperberg-McQueen, C.M.: Extensible Markup Language (XML) 1.0. W3C Recommendation (1998), `http://www.w3.org/TR/REC-xml`
3. Cheng, E., Kaminski, M.: Context-free languages over infinite alphabets. Acta Informatica 35, 245–267 (1998)
4. David, C.: Mots et données infinies. Master's thesis, Université Paris 7, LIAFA (2004)
5. Demri, S., Lazic, R.: LTL with the freeze quantifier and register automata. ACM Transactions on Computational logic 10 (2009) (to appear)
6. Dubov, Y.: Infinite alphabet pushdown automata: various approaches and comparison of their consequences. Master's thesis, Department of Computer Science, Technion – Israel Institute of Technology (2008)
7. Kaminski, M., Francez, N.: Finite-memory automata. Theoretical Computer Science 138, 329–363 (1994)
8. Kaminski, M., Tan, T.: Tree automata over infinite alphabets. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 386–423. Springer, Heidelberg (2008)
9. Kaminski, M., Zeitlin, D.: Extending finite-memory automata with non-deterministic reassignment. In: Csuhaj-Varjú, E., Ézik, Z. (eds.) Proceedings of the 12th International Conference on Automata and Formal Languages – AFL 2008, Computer and Automation Research Institute, Hungarian Academy of Science, pp. 195–207 (2008)
10. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM Transactions on Computational Logic 5, 403–435 (2004)
11. Shemesh, Y., Francez, N.: Finite-state unification automata and relational languages. Information and Computation 114, 192–213 (1994)