Acta Informatica 35, 245–267 (1998)

Context-free languages over infinite alphabets*

Edward Y.C. Cheng[1], **Michael Kaminski**[2]

[1] Department of Computer and Information Sciences and Engineering, University of Florida, Gainesville, FL 32611-2024, USA (e-mail: yccheng@cise.ufl.edu)
[2] Department of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel (e-mail: kaminski@as.technion.ac.il)

Received November 27, 1995 / March 4, 1997

Abstract. In this paper we introduce context-free grammars and pushdown automata over infinite alphabets. It is shown that a language is generated by a context-free grammar over an infinite alphabet if and only if it is accepted by a pushdown automaton over an infinite alphabet. Also the generated (accepted) languages possess many of the properties of the ordinary context-free languages: decidability, closure properties, etc.. This provides a substantial evidence for considering context-free grammars and pushdown automata over infinite alphabets as a natural extension of the classical ones.

1 Introduction

In this paper we introduce context-free grammars and pushdown automata over *infinite* alphabets. In so doing, we are aiming towards a model capable of generating and recognizing only the "natural" analog of *context-free* languages over finite alphabets[1] [3]. We prove that, like in the classical case, the two computational models are equivalent.

In the definition of a context-free grammar, in contrast to [1], we do not want to allow every alphabet symbol to appear in the grammar productions, otherwise generating a simple language Σ^* (for an infinite alphabet Σ) would require infinitely many productions. Thus, each instance of a terminal in a context-free grammar is associated with a register each containing one alphabet symbol. The grammar productions then refer to that *finite set* of registers to generate words. In addition, each grammar production provides a chance to change the content of

* The paper was written when the authors were with Department of Computer Science of the Hong Kong University of Science and Technology.

[1] Since there are no (and cannot be any) formal criteria for accepting a definition as a *natural* extension of context-free languages, it is, to some extent, a matter of taste and intuition what the "natural" analog of context-free languages is.

the registers non-deterministically. This allows the grammar to generate symbols which initially are not in the registers.

The basic idea behind our definition of pushdown automata is similar to that in [5]. An automaton, in addition to a stack and a finite set of proper states (like in the classical model, in which the "real computation" is done) is equipped with a finite number of registers, each of them is capable of storing a symbol from the infinite alphabet. The automaton can non-deterministically change symbols stored in its registers, and the automaton move depends on the current state and the registers containing the input symbol and the top symbol on the stack.

In this paper we prove that a language is generated by a context-free grammar over an infinite alphabet if and only if it is accepted by a pushdown automaton over an infinite alphabet.

Two other different (non-equivalent) definitions of context-free languages over infinite alphabets are known from the literature. In [1] the authors consider an almost straightforward extension of the definition of context-free grammars to infinite alphabets. Their grammars have only finitely many variables, but may have infinitely many productions. The only constraint is that the right-hand side lengths of the productions are uniformly bounded. It can be easily seen that the generated languages do not need to be recursive. Therefore the containment problem for these languages is undecidable. It can be shown that the emptiness problems is also undecidable. On the contrary, the containment and the emptiness problem for the languages defined in this paper are decidable. It should be noted, however, that the languages introduced in [1] possess all other properties of the ordinary context-free languages. Nevertheless, since the pumping lemma does not hold for the context-free languages introduced in this paper, see [5, Example 3], they are not context-free in the sense of [1]. Thus, there are no inclusion relations between the class of languages introduced in [1] and the class of languages considered in this paper.

In [7] context-free languages over infinite alphabets are defined as the inverse homomorphism images of ordinary context-free languages over the alphabet $\{0,1\}$, where the homomorphism maps the symbol σ_i to the word $0^{i-1}1$, $i = 1, 2, \ldots$. These languages possess all the properties of the ordinary context-free languages, but, as it was pointed out in [1], the definition in [7] is too weak. According to this definition, for an infinite alphabet Σ, the language $\{\sigma\sigma\sigma : \sigma \in \Sigma\}$ is not context-free, but it is context-free according to our definition. On the other hand, for an infinite alphabet $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \ldots\}$, the language $\{\sigma_2, \sigma_4, \sigma_6, \ldots\}$ is context-free in the sense of [7], whereas it is not context-free according to our definition, see Example 4 in Sect. 3. Thus, like above, there are no inclusion relations between the class of languages introduced in [7] and the class of languages considered in this paper. The reason is that, by the approach in [7], comparison of two alphabet symbols, in general, is "hard." Indeed, the comparison of σ_i and σ_j requires $\min\{i, j\}$ steps. On the contrary, we assume that comparison of the alphabet symbols is "easy." Namely, in our approach, all, but finitely many, alphabet symbols are initially *indistinguishable*

(see Proposition 2 in Sect. 3), and any two of them can be compared in a constant time.

The paper is organized as follows. In the next section we define context-free grammars and pushdown automata over infinite alphabets and give some examples. In the third section we present some basic properties of context-free languages over infinite alphabets. The last three sections contain the proof of the equivalence of context-free grammars and pushdown automata over infinite alphabets.

2 Context-free grammars and pushdown automata over infinite alphabets

Let Σ be an infinite alphabet. An *assignment* is a word $w_1 w_2 \cdots w_r \in \Sigma^*$ such that $w_i \neq w_j$ for $i \neq j$. That is, an assignment is a word over Σ, where each symbol from Σ appears at most one time. We denote the set of all assignments of length r by $\Sigma^{r\neq}$.

For a word $\boldsymbol{w} = w_1 w_2 \cdots w_n \in \Sigma^*$, 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 symbols which appear in the word $\boldsymbol{w}$.

Definition 1. *An* infinite-alphabet context-free grammar *is a system* $G = (V, \boldsymbol{u}, R, S)$, *where*

- V *is a finite set of* variables, $V \cap \Sigma = \emptyset$.
- $\boldsymbol{u} = u_1 u_2 \cdots u_r \in \Sigma^{r\neq}$ *is the* initial *assignment.*
- $R \subseteq V \times \{1, 2, \ldots, r\} \times (V \cup \{1, 2, \ldots, r\})^*$ *is a set of* productions. *For* $A \in V$, $i = 1, 2, \ldots, r$, *and* $\boldsymbol{a} \in (V \cup \{1, 2, \ldots, r\})^*$, *we write the triple* $(A, i, \boldsymbol{a})$ *as* $(A, i) \to \boldsymbol{a}$.
- $S \in V$ *is the* start symbol.

For $A \in V$, $\boldsymbol{w} = w_1 w_2 \cdots w_r \in \Sigma^{r\neq}$, and $\boldsymbol{X} = X_1 X_2 \cdots X_n \in (\Sigma \cup (V \times \Sigma^{r\neq}))^*$, we write $(A, \boldsymbol{w}) \Rightarrow \boldsymbol{X}$ if there exist a production $(A, i) \to \boldsymbol{a} \in R$, $\boldsymbol{a} = a_1 a_2 \cdots a_n \in (V \cup \{1, 2, \ldots, r\})^*$ and $\sigma \notin [\boldsymbol{w}] - \{w_i\}$ such that the condition below is satisfied.

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

- If $a_j = k$ for some $k = 1, 2, \ldots, r$, then $X_j = w'_k$.
- If $a_j = B$ for some $B \in V$, then $X_j = (B, \boldsymbol{w}')$.

For two words $\boldsymbol{X}$ and $\boldsymbol{Y}$ over $\Sigma \cup (V \times \Sigma^{r\neq})$, we write $\boldsymbol{X} \Rightarrow \boldsymbol{Y}$ if there exist words $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$ over $\Sigma \cup (V \times \Sigma^{r\neq})$, 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 $\Rightarrow^*$. The language $L(G)$ generated by G is defined by $L(G) = \{\boldsymbol{\sigma} \in \Sigma^* : (S, \boldsymbol{u}) \Rightarrow^* \boldsymbol{\sigma}\}$ and is referred to as a *quasi-context-free* language.

Throughout this paper we shall use the following conventions.

- Words are always denoted by boldface letters (possibly indexed or primed).

- Bold low-case Greek letters $\boldsymbol{\alpha}$, $\boldsymbol{\sigma}$, and $\boldsymbol{\tau}$ denote words over Σ.
- Assignments and words of fixed length over Σ are denoted by $\boldsymbol{u}$, $\boldsymbol{v}$, $\boldsymbol{w}$, and $\boldsymbol{x}$.
- Words over $V \cup \{1, 2, \ldots, r\}$ are denoted by $\boldsymbol{a}$.
- Words over $\Sigma \cup (V \times \Sigma^{r\neq})$ are denoted by $\boldsymbol{X}$ and $\boldsymbol{Y}$.
- For a finite alphabet Δ, elements of Δ^r are denoted by $\boldsymbol{\delta}$, always indexed.
- Elements of Σ are denoted by σ, τ, u, v, w, x, and y, usually indexed.
- Variables are denoted by A and B.
- Elements of $V \cup \{1, 2, \ldots, r\}$ are denoted by a, b, and c, usually indexed (and possibly primed).
- Elements of $\Sigma \cup (V \times \Sigma^{r\neq})$ are denoted by X, usually indexed.
- Elements of Δ are denoted by δ, possibly indexed.
- Symbols which appear in a word denoted by a boldface letter are always denoted by the same *non-boldface* letter with some index. That is, symbols which appear in $\boldsymbol{\sigma}$ are denoted by σ_i, symbols which appear in $\boldsymbol{u}$ are denoted by u_i, symbols which appear in $\boldsymbol{X}$ are denoted by X_i, etc.

Remark 1. An r-register infinite-alphabet context-free grammar $G = (V, \boldsymbol{u}, R, S)$ is a "p-grammar" considered in [1] whose set of variables is $V \times \Sigma^{r\neq}$ and whose set of productions consists of all $(A, \boldsymbol{w}) \to \boldsymbol{X}$ such that $(A, \boldsymbol{w}) \underset{G}{\Rightarrow} \boldsymbol{X}$. Thus, an "actual" variable of G is a pair $(A, \boldsymbol{w}) \in V \times \Sigma^{r\neq}$ that will be referred to as a *configuration of* G.

Example 1. Let $G = (\{S\}, \boldsymbol{u}, R, S)$ be an infinite-alphabet context-free grammar, where $\boldsymbol{u} = u_1 \in \Sigma^1$ (that is, $\boldsymbol{u}$ is a word of length one), and R consists of two productions $(S, 1) \to 1S1 \mid \varepsilon$.[2] One can easily verify that $L(G) = \{\boldsymbol{\sigma}\boldsymbol{\sigma}^R : \boldsymbol{\sigma} \in \Sigma^*\}$, where $\boldsymbol{\sigma}^R$ denotes the reversal of $\boldsymbol{\sigma}$. For example, the word $\sigma_1\sigma_2\sigma_3\sigma_3\sigma_2\sigma_1$ is derived as follows.

$$\begin{aligned}(S, \boldsymbol{u}) &\Rightarrow \sigma_1(S, \sigma_1)\sigma_1 \Rightarrow \sigma_1\sigma_2(S, \sigma_2)\sigma_2\sigma_1 \\ &\Rightarrow \sigma_1\sigma_2\sigma_3(S, \sigma_3)\sigma_3\sigma_2\sigma_1 \Rightarrow \sigma_1\sigma_2\sigma_3\sigma_3\sigma_2\sigma_1\,.\end{aligned}$$

Example 2. In this example we show that ordinary context-free languages are quasi-context-free. Let $\Sigma' = \{u_1, u_2, \ldots, u_r\}$ be an r-element subset of Σ and let $G' = (V, \Sigma', R', S)$ be a context-free grammar over Σ'. Consider an infinite-alphabet context-free grammar $G = (V, u_1u_2 \ldots u_r\sigma, R, S)$, where $\sigma \notin \Sigma'$ and R consists of all productions of the form $(A, r + 1) \to a_1a_2 \ldots a_n$ for which there exists $A \to X_1X_2 \ldots X_n \in R'$ such that $a_i = X_i$, if $X_i \in V$, and $a_i = k$, if $X_i = u_k \in \Sigma'$, $i = 1, 2, \ldots, n$. It immediately follows from the definition of G that $L(G') = L(G)$.

Next we define pushdown automata over infinite alphabets.

Definition 2. *An* infinite-alphabet pushdown automaton *is a system* $\mathscr{A} = (Q, s_0, \boldsymbol{u}, \rho, \mu)$*, where*

[2] As usual, ε denotes the empty word.

- 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* $\mathscr{A}$*.*
- $\rho : Q \to \{1, 2, \ldots, r\}$ *is a partial function from* Q *to* $\{1, 2, \ldots, r\}$ *called the* reassignment*. Intuitively, if* $\mathscr{A}$ *is in state* q*, and* $\rho(q)$ *is defined, then* $\mathscr{A}$ *can* non-deterministically *replace the content of the* $\rho(q)$*th register with a new symbol of* Σ *not appearing in any other register. Note that, unlike in [5], we allow* $\mathscr{A}$ *to guess the replacement. This is essential, because grammars can guess symbols they generate.*
- μ *is a mapping from* $Q \times (\{1, 2, \ldots, r\} \cup \{\varepsilon\}) \times \{1, 2, \ldots, r\}$ *to finite subsets of* $Q \times \{1, 2, \ldots, r\}^*$ *called the* transition function*. Intuitively, if* $(p, j_1 j_2 \cdots j_n) \in \mu(q, k, i)$*,* $n \geq 0$*, then (after reassigning the* $\rho(q)$*th register)* $\mathscr{A}$*, whenever it is in the state* q*, with content of the* i*th register at the top of the stack, and the input symbol equal to the content of the* k*th register, can replace the top symbol on the stack with the content of* j_1*th,* j_2*th, . . . ,* j_n*th registers (in this order, read top-down), enter the state* p*, and pass to the next input symbol (possibly* ε*). Similarly, if* $(p, j_1 j_2 \cdots j_n) \in \mu(q, \varepsilon, i)$*, then A, whenever it is in the state* q*, with content of the* i*th register at the top of the stack, can replace the top symbol on the stack with the content of* j_1*th,* j_2*th, . . . ,* j_n*th registers, enter the state p (without reading the input symbol), and pass to the next input symbol (possibly* ε*).*

An *instantaneous description* 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 contents of the registers (in the increasing order of their indices), the third component is the portion of the input yet to be read, and the last one is the contents of the pushdown store, read top-down.

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

- If $\rho(p)$ is not defined, then $v_k = w_k$, $k = 1, 2, \ldots, r$. Otherwise $v_k = w_k$ for $k \neq \rho(p)$ and $v_{\rho(p)} \in \Sigma - \{w_1, \ldots, w_{\rho(p)-1}, w_{\rho(p)+1}, \ldots, w_r\}$.
- If $\sigma = \varepsilon$, then for some i, $\tau = v_i$ and there is $(q, j_1 j_2 \cdots j_n) \in \mu(p, \varepsilon, i)$ such that $\boldsymbol{\alpha} = v_{j_1} v_{j_2} \cdots v_{j_n}$.
- If $\sigma \neq \varepsilon$, then for some k and i, $\sigma = v_k$, $\tau = v_i$, and there is $(q, j_1 j_2 \cdots j_n) \in \mu(p, k, i)$ such that $\boldsymbol{\alpha} = v_{j_1} v_{j_2} \cdots v_{j_n}$.

We denote the reflexive and transitive closure of $\vdash$ by $\vdash^*$ and say that $\mathscr{A}$ accepts a word $\boldsymbol{\sigma} \in \Sigma^*$ if $(s_0, \boldsymbol{u}, \boldsymbol{\sigma}, u_r) \vdash^* (p, \boldsymbol{v}, \varepsilon, \varepsilon)$,[3] for some $p \in Q$ and some $\boldsymbol{v} \in \Sigma^{r\neq}$. Recall that u_r is the last symbol of the initial assignment $\boldsymbol{u} = u_1 u_2 \cdots u_r$.

Remark 2. An infinite-alphabet pushdown automaton $\mathscr{A} = (Q, s_0, \boldsymbol{u}, \rho, \mu)$ is a "pushdown automaton" whose both input and stack alphabets are Σ, the set of

[3] That is, we adopt acceptance by *empty stack*.

states is $Q \times \Sigma^{r_\neq}$, and whose transition function maps $((q, \boldsymbol{w}), \sigma, \tau)$ to the set of all $((p, \boldsymbol{v}), \boldsymbol{\alpha})$ such that $(q, \boldsymbol{w}, \sigma, \tau) \vdash (p, \boldsymbol{v}, \varepsilon, \boldsymbol{\alpha})$, $\sigma \in \Sigma \cup \{\varepsilon\}$. Like in the case of infinite-alphabet context-free grammars, an "actual" state of $\mathscr{A}$ is a pair $(q, \boldsymbol{w}) \in Q \times \Sigma^{r_\neq}$ which will be referred to as a *configuration of* $\mathscr{A}$.

Example 3. In this example we describe an infinite-alphabet pushdown automaton $\mathscr{A}$ such that $L(\mathscr{A}) = \{\boldsymbol{\sigma}\boldsymbol{\sigma}^R : \boldsymbol{\sigma} \in \Sigma^*\}$. That is, $L(\mathscr{A}) = L(G)$, where G is the grammar defined in Example 1. The automaton works in a standard manner. First, it pushes a prefix of the input into the stack. Then it non-deterministically switches to pop the stored symbols which are compared with the rest of the input. Formally, the automaton is defined by $\mathscr{A} = (\{s, p_1, p_2, q\}, s, u_1u_2, \rho, \mu)$, where

- $\rho(s) = 1$, $\rho(p_1) = 2$, $\rho(p_2) = 1$, and $\rho(q) = 1$.
- μ consists of the following four groups of transitions:

1. $\mu(s, 1, 2) = \{(p_1, 1)\}$, and $\mu(s, 2, 2) = \{(p_2, 2)\}$. In the beginning of a computation, these transitions replace u_2 at the stack with the first input symbol. If the first input symbol appears in the first register (possibly after a reassignment), $\mathscr{A}$ enters the state p_1, and if the first input symbol appears in the second register, i.e., the first input symbol is u_2, $\mathscr{A}$ enters the state p_2.

2. $\mu(p_1, 1, 1) = \{(p_1, 11)\}$, $\mu(p_1, 2, 1) = \{(p_2, 21)\}$, $\mu(p_2, 1, 2) = \{(p_1, 12)\}$, and $\mu(p_2, 2, 2) = \{(p_2, 22)\}$. That is, the states p_1 and p_2 are used to push a prefix of the input into the stack. When $\mathscr{A}$ is in the state p_1 (p_2), the top symbol on the stack appears in the first (second) register. If the input symbol appears in the same register, $\mathscr{A}$ pushes it into the stack using the first (fourth) transition. Otherwise, using the reassignment ρ, $\mathscr{A}$ "guesses" the next input symbol at the second (first) register and pushes the symbol into the stack using the second (third) transition.

3. $\mu(p_1, \varepsilon, 1) = \{(q, 1)\}$ and $\mu(p_2, \varepsilon, 2) = \{(q, 2)\}$. These transition are used to switch (non-deterministically) into the "popping" state q. Note that when $\mathscr{A}$ is in the state p_1 (p_2) the top symbol on the stack is stored in the first (second) register.

4. $\mu(q, 1, 1) = \{(q, \varepsilon)\}$ and $\mu(q, 2, 2) = \{(q, \varepsilon)\}$. Being in the state q, the automaton, using the reassignment ρ, "guesses" the next input symbol at the first register and compares it with the top symbol on stack. If the equality holds, $\mathscr{A}$ pops, using the first transition. If the next input symbol appears in the second register, the second transition is used.

For example, $\mathscr{A}$ accepts the word $\sigma_1\sigma_2\sigma_3\sigma_3\sigma_2\sigma_1$, $\sigma_1 \neq \sigma_2$, $\sigma_2 \neq \sigma_3$, $\sigma_3 \neq \sigma_1$, and $\{\sigma_1, \sigma_2, \sigma_3\} \cap \{u_1, u_2\} = \emptyset$, by the following sequence of moves.

$$
\begin{aligned}
&(s, u_1u_2, \sigma_1\sigma_2\sigma_3\sigma_3\sigma_2\sigma_1, u_2) \vdash \\
&(p_1, \sigma_1u_2, \sigma_2\sigma_3\sigma_3\sigma_2\sigma_1, \sigma_1) \vdash \\
&(p_2, \sigma_1\sigma_2, \sigma_3\sigma_3\sigma_2\sigma_1, \sigma_2\sigma_1) \vdash \\
&(p_1, \sigma_3\sigma_2, \sigma_3\sigma_2\sigma_1, \sigma_3\sigma_2\sigma_1) \vdash \\
&(q, \sigma_3\sigma_2, \sigma_3\sigma_2\sigma_1, \sigma_3\sigma_2\sigma_1) \vdash
\end{aligned}
$$

$$(q, \sigma_3\sigma_2, \sigma_2\sigma_1, \sigma_2\sigma_1) \vdash$$
$$(q, \sigma_3\sigma_2, \sigma_1, \sigma_1) \vdash$$
$$(q, \sigma_1\sigma_2, \varepsilon, \varepsilon)$$

The main result of the paper is the equivalence of infinite-alphabet context-free grammars and infinite-alphabet pushdown automata.

Theorem 1. *If a language is quasi-context-free, then it is accepted by an infinite-alphabet pushdown automaton.*

Theorem 2. *If a language is accepted by an infinite-alphabet pushdown automaton, then it is quasi-context-free.*

Theorems 1 and 2 are proved in Sects. 4 and 7, respectively. Whereas the basic idea behind the proofs is not new (see [3] and [6]), the proofs are much more technically involved. In particular, for the proof of Theorem 2 we introduce in Sect. 5 a different kind of context-free grammars over infinite alphabets which allow to store the same symbol in many registers and prove in Sect. 6 that they are equivalent to the original ones. (Recall that the latter must contain different symbols in different registers.) Then we show how infinite-alphabet pushdown automata can be simulated by the new grammars.

3 Properties of quasi-context-free languages

In this section we consider the decision and closure properties of quasi-context-free languages and their relationship to the classical context-free languages.

Proposition 1 below "completes" Example 2. It states that the restriction of a quasi-context-free language to a finite alphabet is context-free.

Proposition 1. *Let G be an infinite-alphabet context-free grammar and let Σ' be a finite subset of Σ. Then one can effectively construct a context-free grammar G' over Σ' such that $L(G) \cap \Sigma'^* = L(G')$.*

Proof. Let $G = (V, \boldsymbol{u}, R, S)$. Consider a context-free grammar $G' = (V', \Sigma', R', S')$ that is defined as follows.

$V' = V \times (\Sigma' \cup [\boldsymbol{u}])^{r\neq}$. Since Σ' is finite, V' is finite as well.
$R' = \{(A, \boldsymbol{w}) \to \boldsymbol{X} : (A, \boldsymbol{w}) \underset{G}{\Rightarrow} \boldsymbol{X}, (A, \boldsymbol{w}) \in V', \text{ and } \boldsymbol{X} \in (\Sigma' \cup V')^*\}$.
$S' = (S, \boldsymbol{u})$.

It follows from the construction above by a straightforward induction on the length k of derivations in G and G' that for $\boldsymbol{X} \in (\Sigma' \cup V')^*$, $(S, \boldsymbol{u}) \underset{G}{\Rightarrow}^k \boldsymbol{X}$ if and only if $(S, \boldsymbol{u}) \underset{G'}{\Rightarrow}^k \boldsymbol{X}$. Therefore, $L(G) \cap \Sigma'^* = L(G')$. □

Propositions 2 and 3 reflect the fact that an infinite-alphabet context-free grammar does not distinguish between different "new" symbols. This property

of infinite-alphabet context-free grammars is a useful tool for proving that a language is not quasi-context-free, see Examples 4 and 5 below. Note that an occurrence of an input symbol that belonged to a previous assignment and was "forgotten" in a reassignment is also "new."

Proposition 2. *Let $G = (V, \boldsymbol{u}, R, S)$ be an infinite-alphabet context-free grammar. Then for each automorphism $\iota : \Sigma \to \Sigma$ we have $\iota(L(G)) = L(G^\iota)$, where $G^\iota = (V, \iota(\boldsymbol{u}), R, S)$.*

Proof. The proof is by induction on the length of a derivation in G. It immediately follows from the definition of $\Rightarrow$ that $(A, \boldsymbol{w}) \underset{G}{\Rightarrow} \boldsymbol{X}$ if and only if $(A, \iota(\boldsymbol{w})) \underset{G^\iota}{\Rightarrow} \iota(\boldsymbol{X})$. This proves the induction basis. Now the induction step follows in a standard manner. □

Corollary . *(Closure under automorphisms) Let G be an infinite-alphabet context-free grammar. Then for each automorphism $\iota : \Sigma \to \Sigma$ that is an identity on $[\boldsymbol{u}]$ and each $\boldsymbol{\sigma} \in \Sigma^*$, $\boldsymbol{\sigma} \in L(G)$ if and only if $\iota(\boldsymbol{\sigma}) \in L(G)$.*

Proof. The result immediately follows from Proposition 2, because, in the conditions of the corollary, the grammars G^ι and G coincide. □

Example 4. Let $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \ldots\}$. Then the language $L = \{\sigma_2, \sigma_4, \sigma_6, \ldots\}$ is not quasi-context-free.[4] Indeed, assume to the contrary, that it is quasi-context-free, and let $G = (V, \boldsymbol{u}, R, S)$ be an infinite-alphabet context-free grammar that generates L. Let i be such that neither σ_{2i} nor σ_{2i+1} belong to $[\boldsymbol{u}]$, and let ι be an automorphism of Σ that permutes σ_{2i} with σ_{2i+1} and leaves all other symbols unchanged. Since ι is an identity on $[\boldsymbol{u}]$, by the corollary to Proposition 2, $\sigma_{2i+1} = \iota(\sigma_{2i}) \in L$, in contradiction with the definition of L.

Also an infinite-alphabet context-free grammar sometimes cannot distinguish between a new symbol and a symbol stored in one of its registers. Such a situation occurs when a new symbol replaces an "old" one after a production is applied. Then the behavior of the grammar is exactly like when it generates the symbol stored in the corresponding register. We illustrate this property of infinite-alphabet context-free grammars by the following proposition.

Proposition 3. *Let $G = (V, \boldsymbol{u}, R, S)$ be an r-register infinite-alphabet context-free grammar. If G generates a word of length n, then it generates a word of length n all of whose symbols appear in $\boldsymbol{u}$. In particular, this word contains at most r distinct symbols.*

Proof. Let $(A, \boldsymbol{u}) \Rightarrow^k \boldsymbol{X} = X_1 X_2 \cdots X_n$. We prove by induction on k that $(A, \boldsymbol{u}) \Rightarrow^k \boldsymbol{X}' = X_1' X_2' \cdots X_n' \in ([\boldsymbol{u}] \cup (V \times \{\boldsymbol{u}\}))^*$.

Let $k = 1$, and let $(A, \boldsymbol{u}) \Rightarrow \boldsymbol{X}$ by a production $(A, i) \to \boldsymbol{a}$. By reassigning the ith register with u_i and applying the same production, we obtain that $(A, \boldsymbol{u}) \Rightarrow \boldsymbol{X}'$, where $|\boldsymbol{X}'| = |\boldsymbol{X}|$ and $\boldsymbol{X}' \in ([\boldsymbol{u}] \cup (V \times \{\boldsymbol{u}\}))^*$.

[4] Recall that L is context-free in the sense of [7].

Let $k > 1$, and let $(A, \boldsymbol{u}) \Rightarrow^k \boldsymbol{X} \Rightarrow \boldsymbol{Y}$. Then there are words $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$ over $\Sigma \cup (V \times \Sigma^{r}_{\neq})$, and $(B, \boldsymbol{w}) \in V \times \Sigma^{r}_{\neq}$ such that $\boldsymbol{X} = \boldsymbol{X}_1(B, \boldsymbol{w})\boldsymbol{X}_2$, $\boldsymbol{Y} = \boldsymbol{X}_1\boldsymbol{X}_3\boldsymbol{X}_2$, and $(B, \boldsymbol{w}) \Rightarrow \boldsymbol{X}_3$. By the induction hypothesis (and the basis), $(A, \boldsymbol{u}) \Rightarrow^k \boldsymbol{X}'_1(B, \boldsymbol{u})\boldsymbol{X}'_2 \Rightarrow^1 \boldsymbol{X}'_1\boldsymbol{X}'_3\boldsymbol{X}'_2$ such that $\boldsymbol{X}' = \boldsymbol{X}'_1\boldsymbol{X}'_3\boldsymbol{X}'_2 \in ([\boldsymbol{u}] \cup (V \times \{\boldsymbol{u}\}))^*$. □

Using Proposition 3 we can easily show that some languages are not quasi-context-free, like in the following example.

Example 5. Consider a language L that consists of all words in which every symbol is different from any of the others. That is, $L = \{\sigma_1\sigma_2 \cdots \sigma_n : \sigma_i \neq \sigma_j,\ i \neq j\}$. We contend that L is not quasi-context-free. To prove our contention, assume to the contrary that L is generated by an r-register infinite-alphabet context-free grammar G. Let $\sigma_1, \sigma_2, \ldots, \sigma_{r+1}$ be different symbols of Σ. Then $\boldsymbol{\sigma} = \sigma_1\sigma_2 \cdots \sigma_{r+1} \in L(G)$. By Proposition 3, G also generates a word $\boldsymbol{\sigma}'$ of length $r + 1$ that contains at most r distinct symbols. Therefore, some symbol appears in $\boldsymbol{\sigma}'$ at least two times. This contradicts the assumption that $L(G) = L$.

Remark 3. It should be pointed out that the straightforward translation of Proposition 3 to infinite-alphabet pushdown automata is not true. Example 6 below shows that for any positive integer r there exists an r-register infinite-alphabet pushdown automaton $\mathscr{A}_r$ such that $L(\mathscr{A}_r) = \Sigma^{2r-1}_{\neq}$. That is, $L(\mathscr{A}_r)$ consists of all words of length $2r - 1$ in which every symbol is different from any of the others. Therefore the counterpart of Proposition 3 for infinite-alphabet pushdown automata should be like the following: *For every infinite-alphabet pushdown automaton $\mathscr{A}$ there exists a constant k such that if $\mathscr{A}$ accepts a word of length n, then it accepts a word of length n that contains at most k distinct symbols.* However, it is not clear at all how to prove the above statement. Thus, Theorem 2 is not trivial.

Example 6. In this example we describe an r-register infinite-alphabet pushdown automaton, $r > 1$, $\mathscr{A}_r = (Q, p_1, u_1u_2 \cdots u_{r-1}u_r, \rho, \mu)$ such that $L(\mathscr{A}_r) = \{\sigma_1\sigma_2 \cdots \sigma_{2r-1} : \sigma_i \neq \sigma_j,\ i \neq j\}$. The automaton computation can be divided into the following four major stages. First the automaton replaces the register content with symbols which do not appear in the input. The replacement is "guessed" using an appropriate reassignment. Since Σ is infinite, there is a "lucky guess." Then the automaton verifies that the first r symbols of the input are pairwise different (by storing them in different registers) and pushes them into the stack. After that it reads the next $r - 1$ symbols of the input, stores them in the first $r - 1$ registers and verifies that they are pairwise different. Finally, the automaton pops, one by one, its stack symbols into the last register and verifies that each of them differs from the symbols stored in the first $r - 1$ registers. Thus, the automaton can empty the stack if and only if its input is of length $2r - 1$ and the input symbols are pairwise different. Accordingly, we break the set of states Q into four groups which correspond to the above stages of computation: $Q = \{p_1, p_2, \ldots, p_r\} \cup \{q_1, q_2, \ldots, q_{r-1}\} \cup \{s_1, s_2, \ldots, s_{r-1}\} \cup \{f\}$, and μ and ρ are defined as follows.

1. $\mu(p_1, \varepsilon, r) = \{(p_2, 1)\}$, where $\rho(p_1) = 1$.
   $\mu(p_k, \varepsilon, 1) = \{(p_{k+1}, 1)\}$, where $\rho(p_k) = k$, $k = 2, 3, \ldots, r-1$.
   $\mu(p_r, \varepsilon, 1) = \{(q_1, r)\}$, where $\rho(p_r) = r$.
   The first reassignment and transition replace both the content of the first register and u_r at the bottom of the stack with a new symbol. The following reassignments and transitions replace the content of the kth register, $k = 2, 3, \ldots, r-1$, with a new symbol. The last reassignment and transition replace the content of the last register and the symbol in the stack with the same new symbol and pass the control to the state q_1 that belongs to the second group of states.
2. $\mu(q_1, 1, r) = \{(q_2, 1)\}$, where $\rho(q_1) = 1$.
   $\mu(q_k, k, k-1) = \{(q_{k+1}, k(k-1))\}$,[5] where $\rho(q_k) = k$, $k = 2, 3, \ldots, r-1$.
   $\mu(q_r, r, r-1) = \{(s_1, r(r-1))\}$, where $\rho(q_r) = r$.
   The first reassignment and transition replace both the content of the first register and the symbol at the stack with the first input symbol. The following reassignments and transitions replace the content of the kth register with the kth input symbol symbol, $k = 2, 3, \ldots, r-1$, and push it into the stack. The last reassignment and transition replace the content of the last register with the rth input symbol, push it into the stack, and pass the control to the state s_1 that belongs to the third group of states. Note that the above sequence of operations can be performed if and only if the kth input symbol, $k = 1, 2, \ldots, r$, differs from each of the symbols already stored in the registers, except the kth one.
3. $\mu(s_k, k, r) = \{(s_{k+1}, r)\}$, where $\rho(s_k) = k$, $k = 1, 2, \ldots, r-2$.
   $\mu(s_{r-1}, r-1, r) = \{(f, r)\}$, where $\rho(s_{r-1}) = r-1$.
   The first $r-2$ reassignments and transitions replace the content of the kth register with the $(r+k)$th input symbol, $k = 1, 2, \ldots, r-2$, and the last reassignment and transition replace the content of the $(r-1)$st register with the last input symbol and pass the control to the state f – the only state in the last group. Again, the above sequence of operations can be performed if and only if the $(r+k)$th input symbol, $k = 1, 2, \ldots, r-1$, differs from each of the symbols already stored in the registers, except the kth register.
4. $\mu(f, \varepsilon, r) = \{(f, \varepsilon)\}$, where $\rho(f) = r$.
   Here the automaton "guesses" the top symbol on the stack, pops it from the stack, and puts it into the last register. Note that the operation is possible if and only if the top symbol on the stack differs from each of the symbols stored in the first $r-1$ registers, i.e., from the last $r-1$ input symbols.

Combining points 1–4 above, we obtain that $L(\mathscr{A}) = \{\sigma_1\sigma_2 \ldots \sigma_{2r-1} : \sigma_i \neq \sigma_j,\ i \neq j\}$.

Propositions 1 and 3 easily imply the following decidability results.

Proposition 4. *The containment problem for infinite-alphabet context-free grammars is decidable.*

[5] Of course, $k(k-1)$ is a word of length 2 the first and the second symbols of which are k and $k-1$, respectively.

Proof. Let G be an infinite-alphabet context-free grammar. Then $\boldsymbol{\sigma} \in L(G)$ if and only if $\boldsymbol{\sigma} \in L(G) \cap [\boldsymbol{\sigma}]^*$; and the result follows from Proposition 1. □

Proposition 5. *The emptiness problem for infinite-alphabet context-free grammars is decidable.*

Proof. Let G be an infinite-alphabet context-free grammar. We contend that $L(G) \neq \emptyset$ if and only if $L(G) \cap [\boldsymbol{u}]^* \neq \emptyset$. The if part is immediate. Let $\boldsymbol{\sigma} \in L(G)$. By Proposition 3, there exists a $\boldsymbol{\sigma}' \in L(G)$, $|\boldsymbol{\sigma}'| = |\boldsymbol{\sigma}|$, such that $\boldsymbol{\sigma}' \in [\boldsymbol{u}]^*$. Thus, $\boldsymbol{\sigma}' \in L(G) \cap [\boldsymbol{u}]^*$.

Now the result follows from the above contention and Proposition 1. □

Finally, Proposition 6 below states that quasi-context-free languages possess closure properties of the classical context-free languages.

Proposition 6. *The quasi-context-free languages are closed under union, concatenation, and the Kleene star.*

The proof of Proposition 6 is based on the following lemma.

Lemma 1. *Let $G = (V, \boldsymbol{u}, R, S)$ be an r-register infinite-alphabet context-free grammar, let $r' \geq r$, and let $f : \{1, 2, \ldots, r\} \to \{1, 2, \ldots, r'\}$ be an injective function. Let $G^f = (V, \boldsymbol{u}', R^f, S)$, $\boldsymbol{u}' = u'_1 \cdots u'_{r'}$, be an r'-register infinite-alphabet context-free grammar such that $u'_{f(i)} = u_i$, $i = 1, 2, \ldots, r$, and R^f is defined as follows.*

For each production $(A, i) \to a_1 a_2 \cdots a_n$ in R there is a production $(A, f(i)) \to a'_1 a'_2 \cdots a'_n$ in R^f, where $a'_j = f(a_j)$, if $a_j \in \{1, 2, \ldots, r\}$, and $a'_j = a_j$, if $a_j \in V$, $j = 1, 2, \ldots, n$. Also for each $i \in \{1, 2, \ldots, r'\}$ that is not in the range of f, R^f contains the production $(S, i) \to S$.

Then $L(G^f) = L(G)$.

Proof. Let $\boldsymbol{\sigma} \in L(G)$. We transform a derivation of $\boldsymbol{\sigma}$ from G into a derivation from G^f in the following manner. First, using the productions of the type $(S, i) \to S$, we reassign all the registers whose indices are not in the range of f with symbols which do not appear in $\boldsymbol{\sigma}$. Then any register whose index is in the range of f can be assigned any symbol that appears in $\boldsymbol{\sigma}$. Now by replacing each application of the production $(A, i) \to a_1 a_2 \cdots a_n$ in $(S, \boldsymbol{u}) \Rightarrow^* \boldsymbol{\sigma}$ with the application of $(A, f(i)) \to a'_1 a'_2 \cdots a'_n$ (as defined in the statement of the lemma) with the same reassignment (but now to the $f(i)$th register), we obtain a derivation of $\boldsymbol{\sigma}$ from $(S, \boldsymbol{u}')$ in G^f. The converse is also true, because the new productions $(S, i) \to S$ can only reassign the registers not in the range of f. □

Proof of Proposition 6. Let $G_1 = (V_1, \boldsymbol{u}_1, R_1, S_1)$ and $G_2 = (V_2, \boldsymbol{u}_2, R_2, S_2)$ be infinite-alphabet context-free grammars with r_1 and r_2 registers, respectively. Renaming the variables, if necessary, we may assume that V_1 and V_2 are disjoint. Let r be the number of elements in $[\boldsymbol{u}_1] \cup [\boldsymbol{u}_2]$ and let $\boldsymbol{u}$ be an assignment of length r such that $[\boldsymbol{u}] = [\boldsymbol{u}_1] \cup [\boldsymbol{u}_2]$. Let $f_1 : \{1, 2, \ldots, r_1\} \to$

$\{1, 2, \ldots, r\}$ and $f_2 : \{1, 2, \ldots, r_2\} \to \{1, 2, \ldots, r\}$ be injective functions such that $u_{f_1(1)}u_{f_1(2)} \cdots u_{f_1(r_1)} = \boldsymbol{u}_1$ and $u_{f_2(1)}u_{f_2(2)} \ldots u_{f_2(r_2)} = \boldsymbol{u}_2$. Finally, let $\sigma \notin [\boldsymbol{u}]$.

Closure under union. Let $G_1^{f_1} = (V_1, \boldsymbol{u}\sigma, R_1^{f_1}, S_1)$ and $G_2^{f_2} = (V_2, \boldsymbol{u}\sigma, R_2^{f_2}, S_2)$ be as in Lemma 1. Consider a grammar $G = (V, \boldsymbol{u}\sigma, R, S)$ such that S is a new variable, $V = V_1 \cup V_2 \cup \{S\}$, and $R = R_1^{f_1} \cup R_2^{f_2} \cup \{(S, r+1) \to S_1 \mid S_2\}$. We contend that $L(G) = L(G_1) \cup L(G_2)$. Since the only productions involving S are $(S, r+1) \to S_1$ and $(S, r+1) \to S_2$, $(S, \boldsymbol{u}\sigma) \underset{G}{\Rightarrow}^* \boldsymbol{\sigma}$ if and only if $(S, \boldsymbol{u}\sigma) \underset{G}{\Rightarrow} (S_1, \boldsymbol{u}\sigma') \underset{G}{\Rightarrow}^* \boldsymbol{\sigma}$ or $(S, \boldsymbol{u}\sigma) \underset{G}{\Rightarrow} (S_2, \boldsymbol{u}\sigma') \underset{G}{\Rightarrow}^* \boldsymbol{\sigma}$ for some $\sigma' \in \Sigma$. Since V_1 and V_2 are disjoint, by Lemma 1, $(S_1, \boldsymbol{u}\sigma') \underset{G}{\Rightarrow}^* \boldsymbol{\sigma}$ if and only if $(S_1, \boldsymbol{u}_1) \underset{G_1}{\Rightarrow}^* \boldsymbol{\sigma}$, and $(S_2, \boldsymbol{u}\sigma') \underset{G}{\Rightarrow}^* \boldsymbol{\sigma}$ if and only if $(S_2, \boldsymbol{u}_2) \underset{G_2}{\Rightarrow}^* \boldsymbol{\sigma}$. Thus, the contention follows.

Closure under concatenation and the Kleene star. The proof is similar to the previous one: $L(G_1)L(G_2)$ is generated by $(V_1 \cup V_2 \cup \{S\}, \boldsymbol{u}\sigma, R_1^{f_1} \cup R_2^{f_2} \cup \{(S, r+1) \to S_1S_2\}, S)$, and $(L(G_1))^*$ is generated by $(V_1 \cup \{S\}, \boldsymbol{u}\sigma, R_1^{f_1} \cup \{(S, r+1) \to S_1S \mid \varepsilon\}, S)$. □

Remark 4. Examples in [5] show that quasi-context-free languages are not closed under homomorphisms and inverse homomorphisms. Also the pumping lemma does not hold for quasi-context-free languages.

4 From grammars to automata

In this section we prove Theorem 1. Let $G = (V, \boldsymbol{u}, R, S)$ be an r-register infinite-alphabet context-free grammar. We shall construct an infinite-alphabet pushdown automaton $\mathscr{A}$ that accepts $L(G)$, by simulating leftmost derivations of G. The construction is similar to that of [6, Lemma 3.4.3, pp. 113–116]. We shall use the notation below.

Let $f : V \to \Sigma$ be an injective function from V into Σ, and let $X \in \Sigma \cup (V \times \Sigma^{r\sharp})$. Then X^f denotes the following word over $V \cup \Sigma$.

$$X^f = \begin{cases} \sigma, & \text{if } X = \sigma \in \Sigma \\ f(A)\boldsymbol{x}, & \text{if } X = (A, \boldsymbol{x}) \in (V \times \Sigma^{r\sharp}) \end{cases}.$$

As usual, for $\boldsymbol{X} = X_1X_2 \cdots X_n \in \{\Sigma \cup (V \times \Sigma^{r\sharp})\}^*$, $\boldsymbol{X}^f$ is defined by $\boldsymbol{X}^f = X_1^f X_2^f \cdots X_n^f$.

The "macro" description of the automaton $\mathscr{A}$ is as follows.

1. It starts by replacing the start stack symbol with $f(S)\boldsymbol{u}$.
2. On each subsequent step it either
   (i) replaces the top portion of the stack $f(A)\boldsymbol{x}$ with $\boldsymbol{X}^f$ such that $(A, \boldsymbol{x}) \Rightarrow \boldsymbol{X}$, or
   (ii) pops the top symbol on the stack, provided that it matches the next input symbol.

Now we pass to a formal description of $\mathscr{A}$. Let $V = \{A_1, A_2, \ldots, A_m\}$, where $S = A_1$. Then $\mathscr{A} = (Q, p_1, \boldsymbol{u}', \rho, \mu)$, where

– $\boldsymbol{u}' = \boldsymbol{u}w_1w_2\cdots w_m$. Intuitively, the first r registers are "floating:" they are intended to simulate the registers of G, and the last m registers are "fixed:" they are intended to keep symbols of Σ which correspond to the grammar variables in V. That is, $f(A_i)$ is stored in the $(r+i)$th register, $i = 1, 2, \ldots, m$. The symbols $f(A_1), f(A_2), \ldots, f(A_m)$ do not appear in the input. The automaton "guesses" them at the beginning of the computation using appropriate reassignments.
– Assume that for each $j = 1, 2, \ldots, m$, the productions of G of the form $(A_j, i) \to \boldsymbol{a}$ are indexed by positive integers from 1 to N_j. Then

$$\begin{aligned} Q &= \{p_1, p_2, \ldots, p_m\} \cup \{t\} \cup \{q_1, q_2, \ldots, q_r\} \\ &\cup \bigcup_{j=1}^{m} \{s_{j,1}, s_{j,2}, \ldots, s_{j,r}\} \cup \bigcup_{j=1}^{m} \{t_{j,1}, t_{j,2}, \ldots, t_{j,N_j}\}. \end{aligned}$$

The intended meaning of the elements of Q is as follows.

The states $p_1, p_2, \ldots, p_m$ are used to reassign the last m registers with symbols not appearing in the input. These symbols correspond to the grammar variables in V.

The state t is used either to pop the top symbol on the stack, provided that it matches the next input symbol, or to pass the control to a sequence of states that simulates a grammar derivation.

The states $\{q_1, q_2, \ldots, q_r\} \cup \{s_{j,1}, s_{j,2}, \ldots, s_{j,r}\}$, $j = 1, 2, \ldots, m$, are used to pop $f(A_j)\boldsymbol{x}$ from the topmost portion of the stack and to store $\boldsymbol{x}$ in the first r registers (and to "remember" A_j, of course). In particular, the states $\{q_k\}_{k=1,2,\ldots,r}$ are used to "refresh" the first r registers with new symbols, and the states $\{s_{j,k}\}_{k=1,2,\ldots,r}$ are used to pop $\boldsymbol{x}$ from the stack into these registers. Finally, the states $\{t_{j,1}, t_{j,2}, \ldots, t_{j,N_j}\}$ are used to simulate productions of G of the form $(A_j, i) \to \boldsymbol{a}$, i.e., to replace $f(A_j)\boldsymbol{x}$ at the top of the stack with $\boldsymbol{X}^f$ such that $(A_j, \boldsymbol{x}) \Rightarrow \boldsymbol{X}$.

– Accordingly, the transition function consists of the following groups of transitions. (We define ρ together with μ.)

1. $\mu(p_1, \varepsilon, r+m) = \{(p_2, 1)\}$, where $\rho(p_1) = r+1$.
   $\mu(p_k, \varepsilon, 1) = \{(p_{k+1}, 1)\}$, where $\rho(p_k) = r+k$, $k = 2, 3, \ldots, m-1$.
   $\mu(p_m, \varepsilon, 1) = \{(t, (r+1)12\cdots r)\}$, where $\rho(p_m) = r+m$.
   These transitions "reassign" the last m registers with $f(A_1), f(A_2), \ldots$ $\ldots, f(A_m)$, replace w_m in the stack with $f(A_1)\boldsymbol{u}$ (recall that $A_1 = S$), and pass the control to the state t.

2(i) $\mu(t, k, k) = \{(t, \varepsilon)\}$, $k = 1, 2, \ldots, r$, and $\mu(t, \varepsilon, r+j) = \{(q_1, r+j)\}$, $j = 1, 2, \ldots, m$, where $\rho(t) = 1$. The transitions $\mu(t, k, k) = \{(t, \varepsilon)\}$, $k = 1, 2, \ldots, r$, are used to pop the top symbol on the stack, if it matches the input symbol (that appears in the kth register). If the input symbol appears in no register, the automaton "guesses" it at the first register using the reassignment. Note that the transitions $\mu(t, k, k) = \{(t, \varepsilon)\}$, $k = 1, 2, \ldots, r$, can be applied only if the content of the last m registers does not appear in the input word. The transitions $\mu(t, \varepsilon, r+j) = \{(q_1, r+j)\}$,

$j = 1, 2, \ldots, m$, are used to pass the control to a sequence of states that simulates a grammar derivation $(A_j, \boldsymbol{x}) \Rightarrow \boldsymbol{X}$.

2(ii) From the state q_1 (with the content of the $(r+j)$th register on the top of the stack, $j = 1, 2, \ldots, m$) the automaton refreshes the first r registers with symbols which do not appear in the input and passes the control to the state $s_{j,1}$:
$\mu(q_k, \varepsilon, r+j) = \{(q_{k+1}, r+j)\}$, where $\rho(q_k) = k$, $k = 1, 2, \ldots, r-1$; and
$\mu(q_r, \varepsilon, r+j) = \{(s_{j,1}, \varepsilon)\}$, where $\rho(q_r) = r$.
From the state $s_{j,1}$ the automaton pops the topmost r symbols from the stack into the first r registers and passes the control to one of the $t_{j,l}$'s which simulate a grammar derivation from A_j:
$\mu(s_{j,k}, \varepsilon, k) = \{(s_{j,k+1}, \varepsilon)\}$, where $\rho(s_{j,k}) = k$, $k = 1, 2, \ldots, r-1$; and
$\mu(s_{j,r}, \varepsilon, r) = \{(t_{j,l}, r+j) : l = 1, 2, \ldots, N_j\}$, where $\rho(s_{j,r}) = r$.
Note that popping of $\boldsymbol{x} = x_1, x_2, \ldots, x_r$ is possible if and only if the kth register is reassigned with x_k, $k = 1, 2, \ldots, r$.
Let $(A_j, i) \to a_1 a_2 \cdots a_n$ be the lth production containing A_j in its left hand side. Then $\mu(t_{j,l}, \varepsilon, r+j) = \{(t, \overline{a}_1 \overline{a}_2 \cdots \overline{a}_n)\}$,[6] where $\rho(t_{j,l}) = i$ and $\overline{a}$ is defined as follows.

$$\overline{a} = \begin{cases} a, & \text{if } a \in \{1, 2, \ldots, r\} \\ (r+h)12\cdots r, & \text{if } a = A_h \in V \end{cases} .$$

Summing up, $\mathscr{A}$ replaces the top portion of the stack $f(A)\boldsymbol{x}$ with $\boldsymbol{X}^f$, where $(A, \boldsymbol{x}) \Rightarrow \boldsymbol{X}$ by the application of the production $(A_j, i) \to a_1 a_2 \cdots a_n$.

The transitions of $\mathscr{A}$ are designed so that the pushdown store during an accepting computation mimics a leftmost derivation of the input word: $\mathscr{A}$ intermittently carries out a step of such a derivation on the stack, by first "refreshing" the first r registers, popping the register content from the stack, and pushing the derived word back into the stack. Between such steps it "strips" away from the top of the stack any non-variable symbols and matches them against symbols in the input word.

Lemmas 2 and 3 below reflect the above intuitive comments.

Lemma 2. *Let* $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma X}$,[7] *where* $\boldsymbol{\sigma} \in \Sigma^*$ *and* $\boldsymbol{X} \in (V \times \Sigma^{r\neq})(\Sigma \cup V \times \Sigma^{r\neq})^* \cup \{\varepsilon\}$, *let* $\boldsymbol{w}' = w'_1 w'_2 \cdots w'_m \in \Sigma^{m\neq}$ *be such that* $[\boldsymbol{w}'] \cap ([\boldsymbol{u}] \cup [\boldsymbol{\sigma}]) = \emptyset$, *and let* $f : V \to \Sigma$ *be defined by* $f(A_j) = w'_j$, $j = 1, 2, \ldots, m$. *Then for some* $\boldsymbol{v} \in \Sigma^{r\neq}$, $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u}) \vdash^* (t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f)$.

Lemma 3. *Let* $\boldsymbol{w}' = w'_1 w'_2 \cdots w'_m \in \Sigma^{m\neq}$ *and let* $f : V \to \Sigma$ *be defined by* $f(A_j) = w'_j$, $j = 1, 2, \ldots, m$. *If for some* $\boldsymbol{v} \in \Sigma^{r\neq}$, $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u}) \vdash^* (t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f)$, *where* $\boldsymbol{\sigma} \in \Sigma^*$ *and* $\boldsymbol{X} \in (\Sigma \cup V \times \Sigma^{r\neq})^*$, *then* $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma X}$.

[6] Note that, by the transition $(t_{j,l}, r+j) \in \mu(s_{j,r}, \varepsilon, r)$, the top stack symbol appears in the $(r+j)$th register.

[7] Here and hereafter the subscript L refers to leftmost derivations.

Since for any $\boldsymbol{w}' \in (\Sigma - [\boldsymbol{u}])^{m}{}_{\neq}$, $(p_1, \boldsymbol{u}w_1w_2 \ldots w_m, \boldsymbol{\sigma}, w_m)$ yields in m steps $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u})$ (and only instantaneous descriptions in such form), Theorem 1 follows from Lemmas 2 and 3 with $\boldsymbol{X} = \varepsilon$. In particular, the inclusion $L(G) \subseteq L(\mathscr{A})$ follows from Lemma 2, and the inclusion $L(\mathscr{A}) \subseteq L(G)$ follows from Lemma 3.

Proof of Lemma 2. The proof is by induction on the length n of a leftmost derivation of $\boldsymbol{\sigma X}$ from $(S, \boldsymbol{u})$. If $n = 0$, then $\boldsymbol{\sigma} = \varepsilon$ and $\boldsymbol{X} = (S, \boldsymbol{u})$. Therefore, for $\boldsymbol{v} = \boldsymbol{u}$, $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u}) \vdash^0 (t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f)$.

Assume that the lemma holds for derivations of length n and prove it for derivations of length $n+1$. Let $(S, \boldsymbol{u}) = \boldsymbol{Y}_0 \Rightarrow_L \boldsymbol{Y}_1 \Rightarrow_L \cdots \Rightarrow_L \boldsymbol{Y}_n \Rightarrow \boldsymbol{Y}_{n+1} = \boldsymbol{\sigma X}$ be a leftmost derivation of $\boldsymbol{\sigma X}$ from $(S, \boldsymbol{u})$. Clearly, $\boldsymbol{Y}_n$ contains a variable. Let $\boldsymbol{Y}_n = \boldsymbol{\sigma}'(A_j, \boldsymbol{x})\boldsymbol{X}'$, where $(A_j, \boldsymbol{x}) \Rightarrow \boldsymbol{\sigma}''\boldsymbol{X}''$, $\boldsymbol{\sigma}', \boldsymbol{\sigma}'' \in \Sigma^*$, $\boldsymbol{\sigma} = \boldsymbol{\sigma}'\boldsymbol{\sigma}''$, and $\boldsymbol{X} = \boldsymbol{X}''\boldsymbol{X}'$.

By the induction hypothesis, for some $\boldsymbol{v} \in \Sigma^{r}{}_{\neq}$,

$$(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}', f(S)\boldsymbol{u}) \vdash^* (t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f) .$$

Let $\boldsymbol{v} = v_1v_2 \ldots v_r$ and let $v_1', v_2', \ldots, v_r'$ be "new" elements of Σ. Then, by the transition $\mu(t, \varepsilon, r + j) = \{(q_1, r + j)\}$ of type 2(i),

$$(t, v_1v_2 \cdots v_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f) \vdash (q_1, v_1v_2 \ldots v_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f) .$$

Next, by the transitions $\mu(q_k, \varepsilon, r + j) = \{(q_{k+1}, r + j)\}$, $k = 1, 2, \ldots, r - 1$, of type 2(ii),

$$\begin{aligned}(q_1, v_1v_2 \cdots v_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f) &\vdash (q_2, v_1'v_2 \cdots v_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f)\\ &\vdash \cdots \vdash (q_r, v_1'v_2' \cdots v_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f) ,\end{aligned}$$

and, by the transition $\mu(q_r, \varepsilon, r + j) = \{(s_{j,1}, \varepsilon)\}$ of type 2(ii),

$$(q_r, v_1'v_2' \cdots v_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{x}\boldsymbol{X}'^f) \vdash (s_{j,1}, v_1'v_2' \cdots v_r'\boldsymbol{w}', \varepsilon, \boldsymbol{x}\boldsymbol{X}'^f) .$$

Let $\boldsymbol{x} = x_1x_2 \ldots x_r$. Then, by the transitions $\mu(s_{j,k}, \varepsilon, k) = \{(s_{j,k+1}, \varepsilon)\}$, $k = 1, 2, \ldots, r - 1$, of type 2(ii),

$$\begin{aligned}(s_{j,1}, v_1'v_2' \cdots v_r'\boldsymbol{w}', \varepsilon, x_1x_2 \cdots x_r\boldsymbol{X}'^f) &\vdash (s_{j,2}, x_1v_2' \cdots v_r'\boldsymbol{w}', \varepsilon, x_2 \cdots x_r\boldsymbol{X}'^f)\\ &\vdash \ldots \vdash (s_{j,r}, x_1x_2 \cdots v_r'\boldsymbol{w}', \varepsilon, \boldsymbol{X}'^f)\end{aligned}$$

and, by the transition $\mu(s_{j,r}, \varepsilon, r) = \{(t_{j,l}, r + j) : l = 1, 2, \ldots, N_j\}$ of type 2(ii),

$$(s_{j,r}, x_1x_2 \cdots v_r'\boldsymbol{w}', \varepsilon, \boldsymbol{X}'^f) \vdash (t_{j,l}, x_1x_2 \cdots x_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{X}'^f) ,$$

where l is the index of the production of the form $(A_j, i) \to \boldsymbol{a}$ used to derive $\boldsymbol{\sigma}''\boldsymbol{X}''$ from $(A_j, \boldsymbol{x})$. Assume that in the derivation of $\boldsymbol{\sigma}''\boldsymbol{X}''$ from $(A_j, \boldsymbol{x})$ the ith register is reassigned with σ. Then, by the transition $\mu(t_{j,l}, \varepsilon, r + j) = \{(t, \bar{a}_1\bar{a}_2 \ldots \bar{a}_n)\}$ of type 2(ii),

$$(t_{j,l}, x_1x_2\cdots x_r\boldsymbol{w}', \varepsilon, f(A_j)\boldsymbol{X}'^f) \vdash (t, x_1x_2\cdots x_{i-1}\sigma x_{i+1}\cdots x_r\boldsymbol{w}', \varepsilon, \boldsymbol{\sigma}''\boldsymbol{X}''^f\boldsymbol{X}'^f)$$
$$= (t, x_1x_2\cdots x_{i-1}\sigma x_{i+1}\cdots x_r\boldsymbol{w}', \varepsilon, \boldsymbol{\sigma}''\boldsymbol{X}^f).$$

Finally, by means of transitions of the first group of type 2(i) (if $\boldsymbol{\sigma}'' \neq \varepsilon$),

$$(t, x_1x_2\cdots x_{i-1}\sigma x_{i+1}\cdots x_r\boldsymbol{w}', \boldsymbol{\sigma}'', \boldsymbol{\sigma}''\boldsymbol{X}^f) \vdash^* (t, xx_2\cdots x_{i-1}\sigma x_{i+1}\cdots x_r\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f),$$

where x is the last symbol in $\boldsymbol{\sigma}''$ that does not appear among $x_2, \ldots, x_{i-1}, \sigma, x_{i+1}, \ldots, x_r$, if $[\boldsymbol{\sigma}''] \not\subseteq \{x_1, x_2, \ldots, x_{i-1}, \sigma, x_{i+1}, \ldots, x_r\}$, and $x = x_1$, otherwise.

Combining the above computation steps of $\mathscr{A}$ together we obtain

$$(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u}) \vdash^* (t, xx_2\cdots x_{i-1}\sigma x_{i+1}\cdots x_r\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f),$$

which completes the proof of the lemma. □

Proof of Lemma 3. The proof is by induction on the number n of appearances of t in the computation of $(t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f)$ from $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u})$. Let $n = 1$, i.e., $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u}) = (t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f)$. Then $\boldsymbol{\sigma} = \varepsilon$ and $\boldsymbol{X} = (S, \boldsymbol{u})$. Therefore, $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma}\boldsymbol{X}$.

Assume that the lemma holds for n and prove it for $n + 1$. Let $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}\sigma, f(S)\boldsymbol{u}) \vdash^* (t, \boldsymbol{v}\boldsymbol{w}', \sigma, \boldsymbol{X}^f) \vdash^* (t, \boldsymbol{v}'\boldsymbol{w}', \varepsilon, \boldsymbol{Y}^f)$, where the computation of $(t, \boldsymbol{v}\boldsymbol{w}', \sigma, \boldsymbol{X}^f)$ contains n appearances of t. Thus, the computation $(t, \boldsymbol{u}\boldsymbol{w}', \boldsymbol{\sigma}, f(S)\boldsymbol{u}) \vdash^* (t, \boldsymbol{v}\boldsymbol{w}', \varepsilon, \boldsymbol{X}^f)$ contains n appearances of t as well. By the induction hypothesis, $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma}\boldsymbol{X}$.

If $\sigma \in \Sigma$, then $\boldsymbol{X} = \sigma\boldsymbol{Y}$. Since $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma}\boldsymbol{X}$, the result follows from the equality $\boldsymbol{\sigma}\boldsymbol{X} = \boldsymbol{\sigma}\sigma\boldsymbol{Y}$.

If $\sigma = \varepsilon$, then $\boldsymbol{X} = (A_j, \boldsymbol{x})\boldsymbol{X}'$ and $\boldsymbol{Y} = \boldsymbol{X}''\boldsymbol{X}'$, where $(A_j, \boldsymbol{x}) \Rightarrow \boldsymbol{X}''$ by means of the lth production. Since $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma}\boldsymbol{X} = \boldsymbol{\sigma}(A_j, \boldsymbol{x})\boldsymbol{X}'$, we have $(S, \boldsymbol{u}) \Rightarrow_L^* \boldsymbol{\sigma}\boldsymbol{Y}$. □

5 Another description of quasi-context-free languages

In this section we introduce a different kind of context-free grammars over infinite alphabets which are allowed to store the same symbol in many registers and in the next section we prove that they can be simulated by the original ones. (Recall that the latter must contain different symbols in different registers.) This result is used for the proof of Theorem 2 in Sect. 7, where we show how infinite-alphabet pushdown automata can be simulated by the new grammars.

Definition 3. *An* infinite-alphabet context-free grammar with multiple reassignment*, or, shortly M-grammar, is a system* $G = (V, \boldsymbol{u}, \Delta, R, S)$*, where*

- V *is a finite set of* variables*.*
- $\boldsymbol{u} = u_1u_2\cdots u_r \in \Sigma^r$ *is the* initial *assignment. Note that in the case of M-grammars the symbols of* $\boldsymbol{u}$ *do not have to be pairwise different.*
- Δ *is a finite non-empty alphabet.*

- $R \subseteq (V \times \boldsymbol{P}_r) \times (\{1,2,\ldots,r\} \cup \Delta \cup (V \times (\{1,2,\ldots,r\} \cup \Delta)^r)^*)$ *is a set of* productions, *where* $\boldsymbol{P}_r$ *is the set of all partitions of* $\{1,2,\ldots,r\}$. *That is, elements of* $\boldsymbol{P}_r$ *are of the form* $\{p_1,p_2,\ldots,p_m\}$, *where* p_i*'s are mutually disjoint non-empty subsets of* $\{1,2,\ldots,r\}$ *such that* $\bigcup_{i=1}^m p_i = \{1,2,\ldots,r\}$. *For* $A \in V$, $\boldsymbol{p} \in \boldsymbol{P}_r$, *and* $\boldsymbol{a} \in (\{1,2,\ldots,r\} \cup \Delta \cup (V \times (\{1,2,\ldots,r\} \cup \Delta)^r)^*)$ *we write the triple* $(A,\boldsymbol{p},\boldsymbol{a})$ *as* $(A,\boldsymbol{p}) \to \boldsymbol{a}$.
- $S \in V$ *is the* start symbol.

Let $\boldsymbol{w} = w_1 w_2 \ldots w_r \in \Sigma^r$ and let $\boldsymbol{p} = \{p_1,p_2,\ldots,p_m\} \in \boldsymbol{P}_r$. We say that $\boldsymbol{p}$ *is associated with* $\boldsymbol{w}$ if the following holds.

For each $i,j = 1,2,\ldots,r$, $w_i = w_j$ if and only if for some $l = 1,2,\ldots,m$, both i and j belong to p_l. Alternatively, for $\sigma \in \Sigma$, let $[\sigma]_{\boldsymbol{w}} = \{i : w_i = \sigma\}$. Then $\boldsymbol{p} = \{[\sigma]_{\boldsymbol{w}} \neq \emptyset : \sigma \in \Sigma\}$.

The unique partition associated with $\boldsymbol{w}$ will be denoted by $\boldsymbol{p}_{\boldsymbol{w}}$.

For a function $f : \Delta \to \Sigma$, $A \in V$, $\boldsymbol{w} = w_1 w_2 \cdots w_r \in \Sigma^r$, and $\boldsymbol{X} = X_1 X_2 \cdots X_n \in (\Sigma \cup (V \times \Sigma^r))^*$, we write $(A,\boldsymbol{w}) \overset{f}{\Rightarrow} \boldsymbol{X}$ if there exist a production $(A,\boldsymbol{p}) \to \boldsymbol{a} \in R$, $\boldsymbol{a} = a_1 a_2 \cdots a_n$, such that the conditions below are satisfied.

- $\boldsymbol{p}$ is associated with $\boldsymbol{w}$.
- If $a_i = j \in \{1,2,\ldots,r\}$, then $X_i = w_j$.
- If $a_i = \delta \in \Delta$, then $X_i = f(\delta)$.
- If $a_i = (B, b_1 b_2 \cdots b_r) \in V \times (\{1,2,\ldots,r\} \cup \Delta)^r$ then $X_i = (B, v_1 v_2 \cdots v_r)$, where v_k, $k = 1,2,\ldots,r$, is defined as follows. If $b_k = j \in \{1,2,\ldots,r\}$, then $v_k = w_j$. If $b_k = \delta \in \Delta$, then $v_k = f(\delta)$.

For two words $\boldsymbol{X}$ and $\boldsymbol{Y}$ over $\Sigma \cup (V \times \Sigma^r)$, we write $\boldsymbol{X} \Rightarrow \boldsymbol{Y}$ if there exist words $\boldsymbol{X}_1$, $\boldsymbol{X}_2$, and $\boldsymbol{X}_3$ over $\Sigma \cup (V \times \Sigma^r)$, and $(A,\boldsymbol{w}) \in V \times \Sigma^r$ 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 for some function $f : \Delta \to \Sigma$, $(A,\boldsymbol{w}) \overset{f}{\Rightarrow} \boldsymbol{X}_3$. As usual, the reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. The language $L(G)$ generated by G is defined by $L(G) = \{\boldsymbol{\sigma} \in \Sigma^* : (S,\boldsymbol{u}) \Rightarrow^* \boldsymbol{\sigma}\}$, and is referred to as an M-language.

Theorem 3. *M-languages are quasi-context-free.*

6 Proof of Theorem 3

Definition 4. *An M-grammar is called* simple *if its finite alphabet Δ contains exactly one symbol, say δ, that appears at most one time in each production, and all the productions containing δ are of the form $(A,\boldsymbol{p}) \to (B, 1 \cdots \delta \cdots r)$.*

Lemma 4. *M-languages are generated by simple M-grammars.*

Proof. Let $G = (V,\boldsymbol{u},\Delta,R,S)$ be an r-register M-grammar, where $\Delta = \{\delta_1,\delta_2,\ldots,\delta_m\}$. We shall construct an $(r+m)$-register simple M-grammar G' such that the $(r+k)$th register of G' simulates δ_k, $k = 1,2,\ldots,m$, in the following manner. At each stage of derivation of G' we first reassign, *step by step*,

each $(r+k)$th register, $k = 1, 2, \ldots, m$, with the "value" of δ_k. After that we reassign the first r registers by referring to the last m registers which correspond to the values of δ_k's.

Formally, consider an $(r+m)$-register M-grammar $G' = (V', \boldsymbol{u}', \{\delta\}, R', S_1)$, that is defined as follows.

- $V' = V \times \{1, 2, \ldots, m+1\}$. The pair (A, k) will be denoted by A_k.
- $\boldsymbol{u}' \in \Sigma^{r+m}$ contains $\boldsymbol{u}$ as a prefix.
- R' consists of the following two types of productions:

1. For each $A \in V$, each $k = 1, 2, \ldots, m$, and each partition $\boldsymbol{p} \in \boldsymbol{P}_{r+m}$, R' contains the production $(A_k, \boldsymbol{p}) \to (A_{k+1}, 12 \cdots r(r+1) \cdots (r+k-1)\delta(r+k+1) \cdots (r+m-1)(r+m))$.

2. For each production $(A, \boldsymbol{p}) \to a_1 a_2 \cdots a_n \in R$, R' contains the production $(A_{m+1}, \boldsymbol{p}') \to a'_1 a'_2 \cdots a'_n$, such that $\boldsymbol{p} = \{p \cap \{1, 2, \ldots, r\} : p \in \boldsymbol{p}'\}$, and a'_i, $i = 1, 2, \ldots, n$, is defined as follows.
   If $a_i \in \{1, 2, \ldots, r\}$, then $a'_i = a_i$.
   If $a_i = \delta_k \in \Delta$, then $a'_i = r + k$.
   If $a_i = (B, b_1 b_2 \cdots b_r)$, then $a'_i = (B_1, c_1 c_2 \cdots c_{r+m})$, where $c_j = j$ for $j > r$, and for $j \leq r$

$$c_j = \begin{cases} b_j, & \text{if } b_j \in \{1, 2, \ldots, r\} \\ r + k, & \text{if } b_j = \delta_k \end{cases} .$$

Let $\boldsymbol{X} = X_1 X_2, \cdots, X_n \in (\Sigma \cup (V \times \Sigma^r))^*$ and let $\boldsymbol{X}' = X'_1 X'_2, \cdots, X'_n \in (\Sigma \cup (V' \times \Sigma^{r+m}))^*$. We say that $\boldsymbol{X}'$ is an *extension* of $\boldsymbol{X}$ if and only if for each $i = 1, 2, \ldots, n$, the following holds. If $X_i \in \Sigma$, then $X'_i = X_i$, and if $X_i = (A, \boldsymbol{w}) \in V \times \Sigma^r$, then $X'_i = (A_1, \boldsymbol{w}\boldsymbol{w}')$ for some $\boldsymbol{w}' \in \Sigma^m$.

Since $\boldsymbol{u}$ is a prefix of $\boldsymbol{u}'$, to prove the lemma it suffices to show that $(A, \boldsymbol{w}) \underset{G}{\Rightarrow}^* \boldsymbol{X}$ if and only if $(A_1, \boldsymbol{w}\boldsymbol{w}') \underset{G'}{\Rightarrow}^* \boldsymbol{X}'$ for some extension $\boldsymbol{X}'$ of $\boldsymbol{X}$.

The latter equivalence easily follows by induction on the length of derivations in G and G': each step $(A, \boldsymbol{w}) \overset{f}{\underset{G}{\Rightarrow}} \boldsymbol{X}$ can be "stretched" to

$$(A_1, \boldsymbol{w}\boldsymbol{w}_0) \overset{f_1}{\underset{G'}{\Rightarrow}} (A_2, \boldsymbol{w}\boldsymbol{w}_1) \overset{f_2}{\underset{G'}{\Rightarrow}} \cdots \overset{f_m}{\underset{G'}{\Rightarrow}} (A_{m+1}, \boldsymbol{w}\boldsymbol{w}_m) \underset{G'}{\Rightarrow} \boldsymbol{X}' ,$$

where $f_k(\delta) = f(\delta_k)$, $k = 1, 2, \ldots, m$, and $\boldsymbol{X}'$ is an extension of $\boldsymbol{X}$. Conversely, each sequence of steps

$$(A_1, \boldsymbol{w}\boldsymbol{w}_0) \overset{f_1}{\underset{G'}{\Rightarrow}} (A_2, \boldsymbol{w}\boldsymbol{w}_1) \overset{f_2}{\underset{G'}{\Rightarrow}} \cdots \overset{f_m}{\underset{G'}{\Rightarrow}} (A_{m+1}, \boldsymbol{w}\boldsymbol{w}_m) \underset{G'}{\Rightarrow} \boldsymbol{X}' ,$$

where $\boldsymbol{X}'$ is an extension of $\boldsymbol{X}$ and $f(\delta_k) = f_k(\delta)$, $k = 1, 2, \ldots, m$ can be "compressed" to $(A, \boldsymbol{w}) \overset{f}{\underset{G}{\Rightarrow}} \boldsymbol{X}$. □

We proceed with the proof of Theorem 3. Let G be an M-grammar. By Lemma 4, we may assume that $G = (V, \boldsymbol{u}, \{\delta\}, R, S)$, $\boldsymbol{u} = u_1 u_2 \cdots u_r$, is simple. We shall construct an infinite-alphabet context-free grammar G' that simulates G. The registers of G with a "multiple" assignment $\boldsymbol{w} = w_1 w_2 \cdots w_r$ will be encoded by an assignment $\boldsymbol{w}' = w'_1 w'_2 \cdots w'_r w'_{r+1} \in \Sigma^{r+1}_{\neq}[\boldsymbol{w}_m] \subseteq [\boldsymbol{w}'_m]$ and an r-dimensional vector $\bar{\boldsymbol{p}} = (p_1, p_2, \ldots, p_r)$ such that $p_k = \{j : w_j = w'_k\}$, $k = 1, 2, \ldots, r$. We shall say that $\bar{\boldsymbol{p}}$ *corresponds* to a partition $\boldsymbol{p}$ if and only if $\boldsymbol{p} = \{p_k \neq \emptyset : k = 1, 2, \ldots, r\}$. Thus, the configuration $(A, \boldsymbol{w})$, $\boldsymbol{w} = w_1 w_2 \cdots w_r$, of G is represented in G' by $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$, $\bar{\boldsymbol{p}} = (p_1, p_2, \ldots, p_r)$ and $\boldsymbol{w}' = w'_1 w'_2 \cdots w'_{r+1}$, such that for each $j \in p_k$, $w_j = w'_k$, $k = 1, 2, \ldots, r$. In particular, $\bar{\boldsymbol{p}}$ corresponds to $\boldsymbol{p}_{\boldsymbol{w}}$ (the partition associated with $\boldsymbol{w}$). We denote the set of all r-dimensional vectors which correspond to the partitions of $\boldsymbol{P}_r$ by $\bar{\boldsymbol{P}}_r$.

Thus, by "looking" at $\bar{\boldsymbol{p}}$ and $\boldsymbol{w}'$, G' "can reproduce" $\boldsymbol{w}$ and, therefore, simulate G. Formally, $G' = (V', \boldsymbol{u}', R', S')$ is defined as follows.

- $V' = V \times \boldsymbol{P}_r$.
- $\boldsymbol{u}' \in \Sigma^{r+1}_{\neq}$, $\boldsymbol{u}' = u'_1 u'_2 \cdots u'_{r+1}$, is an assignment of length $r+1$ such that $[\boldsymbol{u}] \subseteq \{u'_1, u'_2, \ldots, u'_r\}$,
- $S' = (S, (p_1^{\boldsymbol{u}}, p_2^{\boldsymbol{u}}, \ldots, p_r^{\boldsymbol{u}}))$, where $p_k^{\boldsymbol{u}} = \{j : u_j = u'_k\}$.
- R' is the minimal set of productions such that:
  1. R' contains $(S', r+1) \to S'$. Like in Lemma 1, this production is required to reassign the last register with a symbol that does not appear in the derived word.
  2. For each production $(A, \boldsymbol{p}) \to (B, 12 \cdots (k-1)\delta(k+1) \cdots r) \in R$ and for each $\bar{\boldsymbol{p}} = (p_1, p_2, \ldots, p_r)$ that corresponds to $\boldsymbol{p}$, R' contains the following $r+1$ productions:

     $((A, \bar{\boldsymbol{p}}), r+1) \to (B, (p_1^i, p_2^i, \ldots, p_r^i))$, $i = 1, 2, \ldots, r$, where $p_i^i = p_i \cup \{k\}$, and $p_j^i = p_j - \{k\}$, for $j \neq i$. This production corresponds to the case where δ is mapped to a symbol that appears in a register whose index belongs to p_i. Note that if $k \notin p_j$, then $p_j^i = p_j$.

     $((A, \bar{\boldsymbol{p}}), i) \to (B, (p'_1, p'_2, \ldots, p'_r))$, where $p'_i = \{k\}$, $p'_j = p_j - \{k\}$, for $j \neq i$, and i is defined as follows. If $\bar{\boldsymbol{p}}$ has an empty component, then $i = \min\{j : p_j = \emptyset\}$. Otherwise, each component of $\bar{\boldsymbol{p}}$ contains exactly one register index, and we define i by $p_i = \{k\}$. This production corresponds to the case where δ is mapped to a symbol that does not appear in any register. This symbol is placed into the ith register of G'. Note that in this case $(p'_1, p'_2, \ldots, p'_r) = \bar{\boldsymbol{p}}$.
  3. For each production $(A, \boldsymbol{p}) \to a_1 a_2 \cdots a_n \in R$ and for each $\bar{\boldsymbol{p}} = (p_1, p_2, \ldots, p_r)$ that corresponds to $\boldsymbol{p}$, R' contains the production $((A, \bar{\boldsymbol{p}}), r+1) \to a'_1 a'_2 \cdots a'_n$, where a'_i's are defined as follows.

     If $a_i \in \{1, 2, \ldots, r\}$, then $a'_i = k$, where $a_i \in p_k$. This is the case where a_i is substituted (in G) with the symbol that appears in the a_ith register.

If $a_i = (B, b_1 b_2 \cdots b_r)$, then $a_i' = (B, (p_1', p_2', \ldots, p_r'))$, where $p_k' = \{j : b_j \in p_k\}$, $k = 1, 2, \ldots, r$. This is the case where the jth register is reassigned (in G) with the content of the b_jth one (which is the content of the kth register in G').

For the proof of the equality $L(G) = L(G')$, we need the following definition.

Let $\boldsymbol{X} = X_1 X_2, \cdots, X_n \in (\Sigma \cup (V \times \Sigma^r))^*$ and let $\boldsymbol{X}' = X_1' X_2', \cdots, X_n' \in (\Sigma \cup (V' \times \Sigma^{r+1_{\neq}}))^*$ We say that $\boldsymbol{X}'$ is a *reflection* of $\boldsymbol{X}$ if and only if for each $i = 1, 2, \ldots, n$, the following holds. If $X_i \in \Sigma$, then $X_i' = X_i$, and if $X_i = (A, \boldsymbol{w}) \in V \times \Sigma^r$, $\boldsymbol{w} = w_1 w_2 \cdots w_r$, then $X_i' = ((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$, $\bar{\boldsymbol{p}} = (p_1, p_2, \ldots, p_r)$ and $\boldsymbol{w}' = w_1' w_2' \cdots w_{r+1}'$, where for $j \in p_k$, $w_j = w_k'$, $k = 1, 2, \ldots, r$. Note that in the latter case $\bar{\boldsymbol{p}}$ corresponds to $\boldsymbol{p}_{\boldsymbol{w}}$.

Lemma 5. *Let* $(A, \boldsymbol{w}) \underset{G}{\Rightarrow} \boldsymbol{X}$ *and let* $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ *be a reflection of* $(A, \boldsymbol{w})$*, such that the reassignment in the derivation is different from the last symbol of* $\boldsymbol{w}'$*. Then there exists a reflection* $\boldsymbol{X}'$ *of* $\boldsymbol{X}$ *such that* $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}') \underset{G'}{\Rightarrow} \boldsymbol{X}'$.

Lemma 6. *If* $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}') \underset{G'}{\Rightarrow} \boldsymbol{X}'$ *by a production different from* $(S', r+1) \to S'$ *for some reflections* $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ *of* $(A, \boldsymbol{w})$ *and* $\boldsymbol{X}'$ *of* $\boldsymbol{X}$*, then* $(A, \boldsymbol{w}) \underset{G}{\Rightarrow} \boldsymbol{X}$.

The inclusion $L(G) \subseteq L(G')$ follows from Lemma 5, by which each G-derivation

$$(S, \boldsymbol{u}) \underset{G}{\Rightarrow} \boldsymbol{X}_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \boldsymbol{\sigma}$$

can be transformed into a G'-derivation

$$(S', \boldsymbol{u}') \underset{G'}{\Rightarrow} (S', \boldsymbol{u}'') \underset{G'}{\Rightarrow} \boldsymbol{X}_1' \underset{G'}{\Rightarrow} \cdots \underset{G'}{\Rightarrow} \boldsymbol{\sigma},$$

where at the first step the $(r+1)$st register is reassigned with a symbol that does not appear in the G-derivation and remains unchanged in the subsequent steps.

Conversely, the inclusion $L(G') \subseteq L(G)$ follows from Lemma 6, by which each G'-derivation can be transformed into a G-derivation. (Note that, before the transformation, we must delete all steps corresponding to the application of the production $(S', r+1) \to S'$).

Proof of Lemma 5. Let $\boldsymbol{w} = w_1 w_2 \cdots w_r$, $\boldsymbol{w}' = w_1' w_2' \cdots w_{r+1}'$, $\bar{\boldsymbol{p}} = (p_1, p_2, \ldots, p_r)$, and assume that $(A, \boldsymbol{w}) \underset{G}{\Rightarrow} \boldsymbol{X}$ by the application of the production $(A, \boldsymbol{p}) \to \boldsymbol{a}$. Since G is simple, either $\boldsymbol{a}$ is of the form $(B, 12 \cdots (k-1)\delta(k+1) \cdots r)$, or δ does not appear in $\boldsymbol{a}$.

Let $\boldsymbol{a}$ be of the form $(B, 12 \cdots (k-1)\delta(k+1) \cdots r)$. Then $\boldsymbol{X} = (B, \boldsymbol{v})$, where $\boldsymbol{v}$ is obtained from $\boldsymbol{w}$ by replacing w_k with some $\sigma \in \Sigma$. We distinguish between the cases of $\sigma \in [\boldsymbol{w}]$ and $\sigma \notin [\boldsymbol{w}]$.

Let $\sigma \in [\boldsymbol{w}]$. Since $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ is a reflection of $(A, \boldsymbol{w})$, $\sigma = w_i'$, for some $i = 1, 2, \ldots, r$. Therefore by applying to $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ the production $((A, \bar{\boldsymbol{p}}), r+1) \to (B, (p_1^i, p_2^i, \ldots, p_r^i))$, where $p_i^i = p_i \cup \{k\}$, and $p_j^i = p_j - \{k\}$, for $j \neq i$ we obtain $((B, (p_1^i, p_2^i, \ldots, p_r^i)), \boldsymbol{v} w_{r+1}')$ which is a reflection of $(B, \boldsymbol{v})$.

Let $\sigma \notin [\boldsymbol{w}]$. First we consider the case where the symbols of $\boldsymbol{w}$ are pairwise distinct. Then for some $i = 1, 2, \ldots, r$, $p_i = \{k\}$. Since $\sigma \neq w'_{r+1}$, we can apply to $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ the production $((A, \bar{\boldsymbol{p}}), i) \to (B, \bar{\boldsymbol{p}})$ to obtain $((B, \bar{\boldsymbol{p}}), \boldsymbol{v}w'_{r+1})$ which is a reflection of $(B, \boldsymbol{v})$.

Now assume that some symbol appears in $\boldsymbol{w}$ twice or more. Then some p_j must be empty. Let $i = \min\{j : p_j = \emptyset\}$. We can apply to $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ the production $((A, \bar{\boldsymbol{p}}), i) \to (B, (p'_1, p'_2, \ldots, p'_r))$, where $p'_i = \{k\}$, $p'_j = p_j^- \{k\}$, for $j \neq i$, to obtain $((B, \bar{\boldsymbol{p}}), \boldsymbol{v}w'_{r+1})$ which is a reflection of $(B, \boldsymbol{v})$.

Finally, assume that δ does not appear in $\boldsymbol{a}$. By the definition of R', it contains the production $((A, \bar{\boldsymbol{p}}), r+1) \to a'_1 a'_2 \cdots a'_n$, where a'_i's are defined as follows.

- If $a_i \in \{1, 2, \ldots, r\}$, then $a'_i = k$, where $a_i \in p_k$.
- If $a_i = (B, b_1 b_2 \cdots b_r)$, then $a'_i = (B, (p'_1, p'_2, \ldots, p'_r))$, where $p'_k = \{j : b_j \in p_k\}$, $i = 1, 2, \ldots, r$.

Again it readily follows from the definition that the result of the application of this production to $((A, \bar{\boldsymbol{p}}), \boldsymbol{w}')$ is a reflection of $\boldsymbol{X}$. □

The proof of Lemma 6 can be obtained by the "backward" reading of the proof of Lemma 5. We leave it for the reader.

7 From automata to grammars

In this section we show how to simulate infinite-alphabet pushdown automata by M-grammars, which together with Theorem 3 will imply Theorem 2. An immediate corollary to this result is the equivalence of infinite-alphabet context-free grammars and M-grammars. The construction below is similar to that in the proof of [3, Theorem 5.4, pp. 116–199].

Let $\mathscr{A} = (Q, s_0, \boldsymbol{u}, \rho, \mu)$, $\boldsymbol{u} = u_1 u_2 \cdots u_r$, be an r-register infinite-alphabet pushdown automaton. Consider a $(2r+1)$ register M-grammar $G = (V, \boldsymbol{u}u_r\boldsymbol{u}, \Delta, R, S)$ defined below.

$V = Q \times Q \cup \{S\}$, where S is a new symbol. The intuitive meaning of a variable (s, t) is as follows: $((s, t), \boldsymbol{v}\sigma\boldsymbol{w}) \underset{G}{\Rightarrow}^* \boldsymbol{\sigma} \in \Sigma^*$ if and only if $(s, \boldsymbol{v}, \boldsymbol{\sigma}, \sigma) \vdash^* (t, \boldsymbol{w}, \varepsilon, \varepsilon)$. That is, $((s, t), \boldsymbol{v}\sigma\boldsymbol{w})$ derives the portion $\boldsymbol{\sigma}$ of the input word that must be read between a point in time when $\mathscr{A}$ is in the configuration $(s, \boldsymbol{v})$ with σ on the top of the stack and a point in time when $\mathscr{A}$ removes that σ from the stack and enters the configuration $(t, \boldsymbol{w})$. Thus, during a derivation of G the content of the first r registers and the content of the last r registers must be assignments. To ensure this we shall limit the set of partitions allowed in each production. We call a partition $\{p_1, p_2, \ldots, p_m\} \in \boldsymbol{P}_{2r+1}$ *admissible* if its restrictions to $\{1, 2, \ldots, r\}$ and $\{r+2, r+3, \ldots, 2r+1\}$ are $\{\{1\}, \{2\}, \ldots, \{r\}\}$ and $\{\{r+2\}, \{r+3\}, \ldots, \{2r+1\}\}$, respectively.

We do not give an explicit definition of Δ, but just assume that it contains sufficiently many elements, denoted δ or δ_i, to describe all the productions in R that R consists of the following two groups of productions.

1. $(S, \boldsymbol{p}_{\boldsymbol{u}u_r\boldsymbol{u}}) \to ((s_0, q), 12\ldots r(r+1)\delta_1\delta_2\cdots\delta_r)$, for each $q \in Q$. The production in this group say, essentially, that the goal is to pass from the configuration $(s_0, \boldsymbol{u})$ with u_r in the stack to the configuration $(q, f(\delta_1)f(\delta_2)\ldots f(\delta_r))$ with the empty stack.
2. For each $(q,k,i) \in Q \times (\{1,2,\ldots,r\} \cup \{\varepsilon\}) \times \{1,2,\ldots,r\}$, each $(q_1, j_1 j_2 \cdots \cdots j_n) \in \mu(q,k,i)$, $n \geq 0$, each $q_2, q_2, \ldots, q_{n+1} \in Q$, each admissible partition $\boldsymbol{p} \in \boldsymbol{P}_{2r+1}$, and each $\boldsymbol{\delta}_2, \boldsymbol{\delta}_3, \ldots, \boldsymbol{\delta}_n \in \Delta^r$, R contains the production

$$((q, q_{n+1}), \boldsymbol{p}) \to a((q_1, q_2), 1\cdots(\rho(q)-1)b(\rho(q)+1)\cdots r a_1 \boldsymbol{\delta}_2)\cdots$$
$$\cdots((q_\ell, q_{\ell+1}), \boldsymbol{\delta}_\ell a_\ell \boldsymbol{\delta}_{\ell+1})\cdots((q_n, q_{n+1}), \boldsymbol{\delta}_n a_n (r+2)\cdots(2r+1)),$$

where a, b and a_ℓ's, $\ell = 1,2,\ldots,n$, are defined as follows: $a = k \in \{1,2,\ldots,r\} \cup \{\varepsilon\}$, if $\rho(q) \neq k$, and $a = \delta$, otherwise; $b \in \Delta$, if $i \neq \rho(q)$ and $\{i, r+1\} \in \boldsymbol{p}$, and $b = r+1$, otherwise; and $a_\ell = j_\ell$, if $\rho(q) \neq j_\ell$, and $a_\ell = b$, otherwise. According to the above intuitive remarks, this production states that instead of removing the ith register symbol from the stack we have to remove the symbols which appear in the j_1th, j_2th , $\ldots$, j_nth registers (after the reassignment of the $\rho(q)$th register). In particular, if $n = 0$, then the production is $(q, q_{m+1}), \boldsymbol{p}) \to a$.

It immediately follows from the definition of G that

$$(q, \boldsymbol{v}, \sigma, \tau) \vdash (q_1, \boldsymbol{v}', \varepsilon, v_1' v_2' \cdots v_n'),$$

$\sigma \in \Sigma \cup \{\varepsilon\}$, if and only if there exist $q_2, q_3, \ldots, q_{n+1} \in Q$ and $\boldsymbol{w}_2, \boldsymbol{w}_3, \ldots, \boldsymbol{w}_{n+1} \in \Sigma^{r\neq}$ such that

$$((q, q_{n+1}), \boldsymbol{v}\tau\boldsymbol{w}_{n+1}) \Rightarrow \sigma((q_1, q_2), \boldsymbol{v}' v_1' \boldsymbol{w}_2)((q_2, q_3), \boldsymbol{w}_2 v_2' \boldsymbol{w}_3)\cdots$$
$$\cdots((q_{n-1}, q_n), \boldsymbol{w}_{n-1} v_{n-1}' \boldsymbol{w}_n)((q_n, q_{n+1}), \boldsymbol{w}_n v_n' \boldsymbol{w}_{n+1}).$$

Now the proof is exactly as that of [3, Theorem 5.4, pp. 116–119][8] and will be omitted.

Acknowledgement. The authors are very grateful to S.-W. Cheng for his comments on an earlier version of this paper, which dealt with a different model of a pushdown automaton, and to the anonymous referees for their comments, suggestions, and, in particular, for pointing out some flaws in the proof of Theorem 1.

[8] In view of Remark 1 we can consider G as a "context-free grammar with infinitely many variables and terminals," and in view of Remark 2 we can consider $\mathscr{A}$ as a "pushdown automaton with infinitely many states and infinite input and stack alphabets." Then we just write the configuration $((q,p), \boldsymbol{v}\tau\boldsymbol{w})$ of G as $((q, \boldsymbol{v}), \tau, (p, \boldsymbol{w}))$, where $(q, \boldsymbol{v})$ and $(p, \boldsymbol{w})$ are configurations of $\mathscr{A}$. Note that the proof of [3, Theorem 5.4, pp. 116–119] does not use the finiteness conditions on the grammar or the automaton.

References

1. J.-M. Autebert, J. Beauquir, L. Boasson: Langages des alphabets infinis, Discrete Applied Mathematics **2** (1980), 1–20.
2. J.-M. Autebert, J. Beauquir, L. Boasson: Formes de langages et de grammaries, Acta Informatica **17** (1982), 193–213.
3. J.E. Hopcroft, J.D. Ullman: Introduction to automata theory, languages, and computation, Addison Wesley, Reading, MA, 1979.
4. J. Itd: Automates a pile sur des alphabets infinis, Proceedings of the Symposium of Theoretical Aspects of Computer Science, Springer-Verlag, Berlin, 1984, pp . 260–273 (Lecture Notes in Computer Science **166**).
5. M. Kaminski, N. Francez: Finite-memory automata, Theoretical Computer Science A **138** (1994), 329–363.
6. H.R. Lewis, C.H. Papadimitriou: Elements of the theory of computation, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.
7. F. Otto: Classes of regular and context-free languages over countably infinite alphabets, Discrete Applied Mathematics **12** (1985), 41–56.
8. Y. Shemesh, N. Francez: Finite-state unification automata and relational languages, Information and Computation **114** (1994), 192–213.