Juhani Karhumäki | Michal Kunc | Alexander Okhotin

Computational power of two stacks with restricted communication

Turku Centre *for* Computer Science

TUCS Technical Report
No 744, May 2008

Computational power of two stacks with restricted communication

Juhani Karhumäki
Michal Kunc
Alexander Okhotin
Department of Mathematics, University of Turku and
Turku Centre for Computer Science
Turku FIN–20014, Finland
karhumak@utu.fi, kunc@math.muni.cz, alexander.okhotin@utu.fi

TUCS Technical Report
No 744, May 2008

Abstract

Rewriting systems working on words with a center marker are considered. The derivation is done by erasing a prefix or a suffix and then adding a prefix or a suffix. This models a communication of two stacks according to a fixed protocol defined by the choice of rewriting rules. The paper systematically considers different cases of these systems and determines their expressive power. Several cases are identified where very restricted communication surprisingly yields computational universality.

Keywords: Word rewriting, string rewriting, Post systems, multi-pushdown machines, communication, universality

TUCS Laboratory
Discrete Mathematics for Information Technology

1 Introduction

The earliest evidence of computational power of simple rewriting systems seems to be the following fascinating example of Post [13] dating to 1920's. Given a binary word w, apply to it iteratively the following rule: "omit the three-letter prefix and, if its first symbol was 0 (1, respectively), append a suffix 00 (1101, respectively)". There are three possible outcomes: either the process terminates, or goes into a periodic stage, or proceeds without repetitions. As noticed already by Post, it is not easy to determine what is the case for a given w. In fact, even now no algorithm to decide this is known. Based on the above example Post developed his *canonical systems*, which have universal computing power.

In 1960's Büchi [3] and Kratko [8] independently started to consider simpler rewriting systems of a similar kind. They noticed that if Post's rules are applied locally, that is, a word w is rewritten to $y(x^{-1}w)$ under a rule $x \to y$, then the languages obtained are regular. On the other hand, the power of another simpler variant of Post's rewriting was determined only recently. In this rewriting, given an initial word w and a finite or a regular set X, the rule "delete a prefix from X and append any word from X to the end" is iteratively applied. The choice of the word being appended is independent of the word removed, so this was called *uncontrolled one-way rewriting*, and the regularity of the sets generated was established [7].

This process resembles another problem dealing with operations on two ends of a word proposed by Conway [4]. He asked whether for every regular language X the largest language Z with $XZ = ZX$ is regular as well. This problem was recently solved strongly negatively: by a sophisticated construction it was proved that such Z need not be recursively enumerable [9]. This demonstrated that Conway's equation is deeper than it seems, and motivated the study of its approximate sequential variant, called *uncontrolled two-way rewriting*, defined by the rule "delete an element of X from either of the ends of the word, and at the same time append any element of X to the opposite end". It was found to generate a nonregular language [7], but its exact power was left undetermined.

This paper is dedicated to a systematic study of such rewriting systems and their variants. We assume that the words being rewritten contain a center marker that is never touched: this does not reduce the power of the most general case and leads to interesting special cases. Besides two-way rewriting, we consider three augmented cases of one-way rewriting, which are illustrated by diagrams in Figure 1, where # is the center marker. In these subcases local Büchi rewriting steps are also allowed, that is, steps where both deletion and appending are done at the same end. If this is allowed on the right, we call the rewriting *one-way R-rewriting*, where R stands for "receiving". If the local rewriting is allowed only at the left end, then it is called *one-way S-rewriting*, where S stands for "sending". Finally,

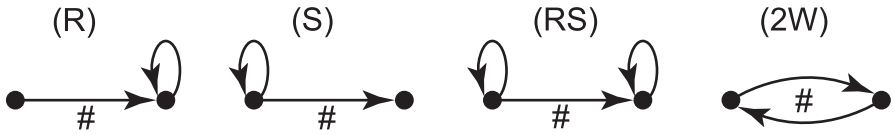


Figure 1: Modes of rewriting.

if it is allowed at both ends, then we call it *one-way RS-rewriting*.

In all of the above cases the rewriting may be *controlled* or *uncontrolled*. In the former case a connection between the word x being erased and the word y being appended is allowed: for example, the set of all pairs (x, y) may be defined by a recognizable or rational relation. In the latter case the words to be deleted and added are chosen independently, that is, the relation between x and y is a Cartesian product of two sets, which we call an uncontrolled relation. Obviously, the computational power of the former tends to be much higher than that of the latter, though in some cases we shall see that uncontrolled rewriting is as powerful as its controlled variant.

We have described our approach in terms of rewriting. However, our systems can be equally interpreted in terms of communication between two (pushdown) stacks. In one-way rewriting we pop from the left stack and simultaneously push onto the right stack, that is, we send a message from left to right. When an uncontrolled relation defines a communication, all messages sent through such a channel are indistinguishable, that is, the fact of sending a message constitutes the entire message. When we pop from and push onto the same stack, this models local processing of data. This interpretation gives a further motivation for our systematic study.

Our presentation is structured as follows. In Section 2 we formally define all variants of our rewriting systems. Then in Section 3 we consider one-way R-rewriting. Here, assuming that the set of initial words is regular, only regular languages can be generated even using a seemingly powerful controlled rewriting. On the other hand, for a context-free initial set the rewriting becomes computationally universal. In the case of one-way S-rewriting studied in Section 4, our result is, in essence, that already quite restricted models generate all context-free languages, but even their significant generalizations do not give anything more. For one-way RS-rewriting, in Section 5 we obtain a greater variety of results: if the one-way rewriting is uncontrolled, then only special cases of context-free languages are generated, while if it is controlled, all recursively enumerable languages can be obtained. Finally, in Section 6 we establish the unexpected computational universality of uncontrolled two-way rewriting.

2 Formal definitions and notation

We consider finite words over a finite nonempty alphabet Σ. For every word $w = a_1 \dots a_n \in \Sigma^*$, with $a_i \in \Sigma$, we denote its length by $|w| = n$ and its reversal by $w^{\mathrm{R}} = a_n \dots a_1$. The unique word of length 0 is denoted ε. Any subset $L \subseteq \Sigma^*$ is called a language. We consider standard operations on languages, such as concantenation $K \cdot L = \{\, uv \mid u \in K, v \in L \,\}$, right-quotient $K \cdot L^{-1} = \{\, x \mid \exists y \in L :\ xy \in K \,\}$, left-quotient $L^{-1} \cdot K = \{\, x \mid \exists y \in L :\ yx \in K \,\}$ and Boolean operations.

Various standard formalisms for specifying languages shall be used throughout this paper. Brief definitions are given below. For further explanations the reader is referred to the textbooks by Salomaa [15] and by Sakarovitch [14].

A *deterministic finite automaton* (DFA) is a quintuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, in which Q is a finite set of states, with the initial state $q_0 \in Q$ and the set of final states $F \subseteq Q$, while $\delta\colon Q \times \Sigma \to Q$ is a transition function. Extend δ to the domain $Q \times \Sigma^*$ as $\delta(q, \varepsilon) = q$ and $\delta(q, wa) = \delta(\delta(q, w), a)$, and define the language recognized by $\mathcal{A}$ as $L(\mathcal{A}) = \{\, w \in \Sigma^* \mid \delta(q_0, w) \in F \,\}$. A language is *regular* if it is recognized by some DFA.

A *context-free grammar* is a quadruple $G = (\Sigma, N, P, S)$, with the set of nonterminal symbols N, the start symbol $S \in N$, and a finite set P of rewriting rules of the form $A \to \alpha$, for $A \in N$ and $\alpha \in (\Sigma \cup N)^*$. We shall refer to the rules of grammars as *productions* and reserve the word "rule" for rewriting systems operating on sides of words, which are the main subject of this paper. The relation $\Longrightarrow$ of generation on $(\Sigma \cup N)^*$ is defined by $\eta A \theta \Longrightarrow \eta \alpha \theta$ for any $A \to \alpha \in P$ and for any $\eta, \theta \in (\Sigma \cup N)^*$. Let $\Longrightarrow^*$ be the reflexive and transitive closure of $\Longrightarrow$. Every word $\alpha \in (\Sigma \cup N)^*$ generated from S in zero or more steps (that is, with $S \Longrightarrow^* \alpha$) is called a *sentential form* of G. The language generated by the grammar is defined as $L(G) = \{\, w \in \Sigma^* \mid S \Longrightarrow^* w \,\}$. Furthermore, we consider the language generated by any word $\alpha \in (\Sigma \cup N)^*$, defined as $L_G(\alpha) = \{\, w \in \Sigma^* \mid \alpha \Longrightarrow^* w \,\}$. If $\alpha \in \Sigma^* N \Sigma^* \cup \Sigma^*$ in every production in P, the grammar is called *linear*. A language is (*linear*) *context-free* if it is generated by some (linear) context-free grammar.

If productions in the definition of a context-free grammar are allowed to be of the form $\alpha \to \beta$, with $\alpha \in N^+$ and $\beta \in (\Sigma \cup N)^*$, and the rest of the definition of generation is repeated verbatim, the resulting device can generate any recursively enumerable set. It is known as Chomsky's *type 0 grammar*.

A *pushdown automaton* (PDA) is defined as a septuple, $\mathcal{B} = (\Sigma, \Gamma, Q, q_0, \delta, F, \gamma_0)$, in which Q, q_0 and F are as in the case of DFAs, Γ is the pushdown alphabet with γ_0 as the initial symbol, and the transition function δ maps $Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\})$ to the set of finite subsets of $Q \times \Gamma^*$. The configurations of the automaton are triples (q, w, x), where $q \in Q$, $w \in \Sigma^*$

and $x \in \Gamma^*$. The relation $\vdash$ of one-step transition on the set of these configurations is defined as $(q, uw, \gamma z) \vdash (q', w, yz)$, for all $(q', y) \in \delta(q, u, \gamma)$. The language generated by the PDA is

$$L(\mathcal{B}) = \{\, w \in \Sigma^* \mid (q_0, w, \gamma_0) \vdash^* (q_F, \varepsilon, \varepsilon) \text{ for some } q_F \in F \,\}.$$

We shall use the following semi-formal notation for commonly used cases of PDA transitions: $\delta(q, u, \varepsilon) \ni (q', \varepsilon)$ will be called "READ u", $\delta(q, \varepsilon, \varepsilon) \ni (q', x)$ is "PUSH x", while $\delta(q, \varepsilon, \gamma) \ni (q', \varepsilon)$ is "POP γ". In some cases we shall restrict, without loss of generality, the domain of δ to be $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$, that is, every transition checks the top pushdown symbol. Pushdown automata recognize exactly the context-free languages.

An important subclass of PDAs are the *counter automata*, which use a single pushdown symbol 1 (besides the initial symbol γ_0) and which are known to be strictly less powerful than the general PDAs. The initial symbol may appear only at the bottom of the stack and is used to test the emptiness of the pushdown. Then it is natural to consider the pushdown as a counter, and to interpret the number of symbols 1 in it as the value of the counter. We shall actually deal only with a subclass of *one-turn counter automata*, in which, in course of a computation on any word, the value of the counter monotonically increases until some point, and then monotonically decreases until the end of the computation. For such automata we can assume that they have no initial pushdown symbol, as zero test is not needed.

Let us now define the rewriting systems we study in this paper. Let Σ be a finite nonempty alphabet and let $\# \notin \Sigma$ be an additional symbol called the *center marker*. Let $I \subseteq \Sigma^*\#\Sigma^*$ be a set of words with a center marker, from which the rewriting starts; this set is called the *initial set*. Let $\overset{\ell\ell}{\to}, \overset{rr}{\to}, \overset{\ell r}{\to}, \overset{r\ell}{\to} \subseteq \Sigma^* \times \Sigma^*$ be four relations that constitute the *rewriting rules*. The corresponding relations of one-step rewriting are defined as follows:

$$\begin{aligned}
xw\#w' &\overset{\ell\ell}{\Longrightarrow} yw\#w' \quad (w, w' \in \Sigma^*, (x, y) \in \overset{\ell\ell}{\to}) \\
w\#w'x &\overset{rr}{\Longrightarrow} w\#w'y \quad (w, w' \in \Sigma^*, (x, y) \in \overset{rr}{\to}) \\
xw\#w' &\overset{\ell r}{\Longrightarrow} w\#w'y \quad (w, w' \in \Sigma^*, (x, y) \in \overset{\ell r}{\to}) \\
w\#w'x &\overset{r\ell}{\Longrightarrow} yw\#w' \quad (w, w' \in \Sigma^*, (x, y) \in \overset{r\ell}{\to})
\end{aligned}$$

The relation of one-step rewriting is defined as the union of these relations:

$$\Longrightarrow \;=\; \left(\overset{\ell\ell}{\Longrightarrow} \cup \overset{rr}{\Longrightarrow} \cup \overset{\ell r}{\Longrightarrow} \cup \overset{r\ell}{\Longrightarrow}\right)$$

The reflexive and transitive closure of $\Longrightarrow$ is denoted by $\Longrightarrow^*$. The language generated by the rewriting system is then defined as

$$\{\, w \mid \exists w_0 \in I :\; w_0 \Longrightarrow^* w \,\}.$$

The main modes of rewriting we consider, illustrated in Figure 1, are formally defined as follows:

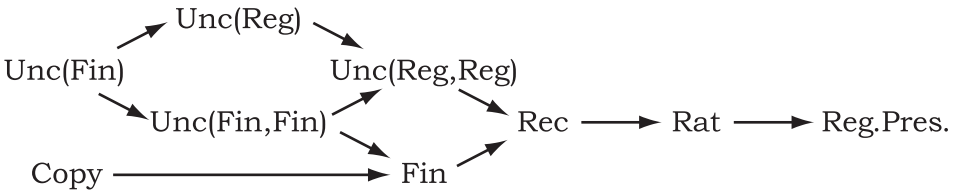


Figure 2: Families of relations used for $\overset{\ell\ell}{\to}$, $\overset{rr}{\to}$, $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$.

- If the relations $\overset{\ell r}{\to}$ and $\overset{rr}{\to}$ are essentially used, while $\overset{\ell\ell}{\to} = \overset{r\ell}{\to} = \varnothing$, we call this *one-way R-rewriting*.
- If $\overset{\ell\ell}{\to}$ and $\overset{\ell r}{\to}$ are used and $\overset{rr}{\to} = \overset{r\ell}{\to} = \varnothing$, this is *one-way S-rewriting*.
- The combined version of (R) and (S), in which $\overset{\ell\ell}{\to}$, $\overset{\ell r}{\to}$ and $\overset{rr}{\to}$ are used and $\overset{r\ell}{\to} = \varnothing$, will be called *one-way RS-rewriting*.
- Finally, if $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$ are used and $\overset{\ell\ell}{\to} = \overset{rr}{\to} = \varnothing$, we call this *two-way rewriting*.

The strong representability results obtained for two-way rewriting in Section 6 make it unnecessary to consider any more powerful cases.

Let us now further classify relations $\overset{\ell\ell}{\to}$, $\overset{rr}{\to}$, $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$. One type of relations we consider are *uncontrolled* relations, which represent rewriting rules in which there is no connection between the word erased and the word written. Such relations are defined by Cartesian products of two languages as follows:

- Let $\mathcal{L}$ and $\mathcal{M}$ be two families of languages. A relation $\to$ is *uncontrolled* using $\mathcal{L}$ and $\mathcal{M}$, denoted $\to \in Unc(\mathcal{L}, \mathcal{M})$, if $\to = X \times Y$ for any languages $X \in \mathcal{L}$ and $Y \in \mathcal{M}$. We shall consider classes of relations $Unc(Reg, Reg)$ and $Unc(Fin, Fin)$, in which the languages X and Y must be regular or finite, respectively.
- The class of uncontrolled relations of a special form $\to = X \times X$, for a single language $X \in \mathcal{L}$, will be denoted by $Unc(\mathcal{L})$. We shall consider $Unc(Reg)$ and $Unc(Fin)$.

We call other relations *controlled* and consider, in particular, the following well-known families:

- *Finite* relations (Fin) are given by lists of pairs of the form $\to = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. These are the commonly used sets of rewriting rules, such as in Thue systems and Chomsky phrase-structure grammars.

  A very simple case of finite relations are the *copy* relations: let $\Sigma' \subseteq \Sigma$ and define $\to = \{\, (a, a) \mid a \in \Sigma' \,\}$. When $\overset{\ell r}{\to} \in Copy$, it moves certain symbols verbatim from one side of the word to the other.

- *Recognizable* relations (Rec) are those for which the language $\{\, x\$y \mid (x,y) \in \rightarrow \}$ is regular, or, equivalently, which can be represented as a finite union of relations in $Unc(Reg, Reg)$.

- *Rational* relations (Rat) are those recognized by finite transducers.

  A (nondeterministic) *finite transducer* is defined as a sextuple $(\Gamma, \Theta, Q, q_0, \delta, F)$, where Γ and Θ are two alphabets; Q is a finite set of states; $q_0 \in Q$ is the initial state; $\delta\colon Q \times \big((\Gamma \times \{\varepsilon\}) \cup (\{\varepsilon\} \times \Theta)\big) \to 2^Q$ maps triples of the form (q, a, ε) and (q, ε, s), with $q \in Q$, $a \in \Gamma$ and $s \in \Theta$, to subsets of the set of states; and $F \subseteq Q$ is the set of final states. The relation defined by a transducer consists of all pairs of words $(u, v) \in \Gamma^* \times \Theta^*$, for which there exists a sequence of states $p_0, p_1, \ldots, p_n$, where $p_i \in Q$, $p_0 = q_0$, $p_n \in F$, and factorizations $u = u_1 \ldots u_n$ and $v = v_1 \ldots v_n$, with $p_i \in \delta(p_{i-1}, u_i, v_i)$ for all i. In the case of our rewriting systems, the relations are $\rightarrow \subseteq \Sigma^* \times \Sigma^*$, and the transducers have $\Gamma = \Theta = \Sigma$.

- The most general family of relations we consider are the *regularity preserving* relations ($Reg.Pres.$), i.e., relations $\rightarrow$, such that for every regular L, the language $\{\, y \mid \exists x \in L :\ (x, y) \in \rightarrow \}$ is regular.

These families of relations form a hierarchy from copy relations and $Unc(Fin)$ up to the most powerful class $Reg.Pres.$; this hierarchy is given in Figure 2.

A class of rewriting systems is defined by the mode of rewriting (R, S, RS or 2W), the families of relations to which the relations $\overset{\ell\ell}{\rightarrow}$, $\overset{\ell r}{\rightarrow}$, $\overset{r\ell}{\rightarrow}$ and $\overset{rr}{\rightarrow}$ may belong and the family of languages from which the initial set may be selected. Every class of rewriting systems defines a family of formal languages. On the other hand, each class of rewriting systems models a certain type of communication between two stacks, and the computational power of this kind of communication is characterized by the aforementioned language family. For example, we shall consider S-rewriting with recognizable $\overset{\ell\ell}{\rightarrow}$, rational $\overset{\ell r}{\rightarrow}$ and context-free I, and show that these rewriting systems define only context-free languages.

Quite a few classes of rewriting systems we study turn out to have universal computational power, that is, some system from such a class generates a language, to which every recursively enumerable language is reducible. This usually means that there is a way to encode the definition of any given Turing machine in the rewriting rules of such a system, so that derivations of words of a certain form (that is, belonging to a certain regular language) simulate computations of the Turing machine, while derivations of the rest of the words are irrelevant.

For some combinations of modes of rewriting and families of relations their computational universality is too obvious. If either of the relations $\overset{\ell\ell}{\rightarrow}$, $\overset{rr}{\rightarrow}$ is rational, then it is easy to produce an r.e.-complete language from

a one-element initial set by using a well-known result that iteration of finite transducers has universal computational power.

This holds already for an *input-deterministic transducer*, also known as a generalized sequential machine or gsm. Transducers of this kind are defined with a transition function specifying, for every state q and for every input symbol $a \in \Gamma$, a unique target state q' and output word $w \in \Theta^*$. Formally, its transition function is $\delta\colon Q \times \Gamma \to Q \times \Theta^*$, and the transducer implements a function $\to$ from Γ^* to Θ^*. In case $\Gamma = \Theta = \Sigma$, one can consider the relation $\to^* \subseteq \Sigma^* \times \Sigma^*$, which is the reflexive and transitive closure of $\to$.

Folklore Theorem. *There exists an alphabet Σ, a word $w_0 \in \Sigma^*$ and a function $\to$ from Σ^* to Σ^* computed by an input-deterministic finite transducer, for which the language $\{ w \in \Sigma^* \mid w_0 \to^* w \}$ is r.e.-complete.*

This can be established, for instance, as follows. Let M be a Turing machine starting on an empty tape and doing one infinite computation, in which it produces all elements of a certain r.e.-complete set one by one. Let the configuration of M in state q, with the tape containing uav and with the head over a be represented by a string $uqav$ over a suitable alphabet Σ. Let w_0 be the word representing the initial configuration of M. Then an input-deterministic finite transducer can be constructed, which computes the next configuration of M out of the previous one. Therefore, the set $\{ w \in \Sigma^* \mid w_0 \to^* w \}$ represents all reachable configurations of M, and the given r.e.-complete set is clearly reducible to it.

In view of the above folklore theorem, the most general relations we are going to consider in this paper are regularity-preserving relations $\xrightarrow{\ell r}$ and $\xrightarrow{r\ell}$ (in particular, rational ones) and recognizable relations $\xrightarrow{\ell\ell}$ and $\xrightarrow{rr}$.

Let us also mention another folklore theorem on the computational completeness of two-pushdown automata. Because of this, controlled 2W-rewriting is trivially computationally universal, and so only the uncontrolled case is of interest.

3 Receiving

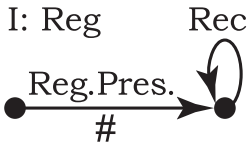
In this section we consider *one-way R-rewriting*, which uses the relations $\xrightarrow{\ell r}$ and $\xrightarrow{rr}$. We can show that the relation $\Longrightarrow^*$ of such rewriting systems preserves regularity even in the most general case, and thus the power of this rewriting is determined by the initial set. Let us organize the study of R-rewriting on the basis of whether I is regular or not.

3.1 The case of regular I

If I is regular, then $\xrightarrow{rr}$ may be any recognizable relation, while the relation $\xrightarrow{\ell r}$ needs to be only regularity-preserving, and the resulting rewriting system

is always guaranteed to generate a regular language.

Theorem 1. *The language $L \subseteq \Sigma^*\#\Sigma^*$ generated by a rewriting system consisting of a regularity-preserving relation $\xrightarrow{\ell r}$, a recognizable relation $\xrightarrow{rr}$ and a regular initial set I is always regular. If in addition images of regular languages under $\xrightarrow{\ell r}$ can be algorithmically computed, then L is algorithmically computable too.*



For a given alphabet Σ, we shall use its disjoint copy $\widetilde{\Sigma} = \{\, \widetilde{a} \mid a \in \Sigma \,\}$, where the letter $\widetilde{a}$ will represent deletion of a. For any word $w = a_1 \dots a_n \in \Sigma^*$, where $n \in \mathbb{N}_0$ and $a_1, \dots, a_n \in \Sigma$, denote $\widetilde{w} = \widetilde{a}_n \dots \widetilde{a}_1$, and extend this notation to languages by the rule $\widetilde{K} = \{\, \widetilde{w} \mid w \in K \,\}$.

The proof consists of two parts. First, we consider *computation histories* of our rewriting, in which, to the right from #, all letters which occurred there during the rewriting are preserved and only symbolically deleted by appending the corresponding "negative" symbols from $\widetilde{\Sigma}$. We shall demonstrate that the language of all such computation histories, defined as

$$L_0 = \big\{ u\#v_0\rho_0 z_1\rho_1 \dots z_n\rho_n \;\big|\; n \geqslant 0, \rho_i \in \{\, \widetilde{x}y \mid (x,y) \in \xrightarrow{rr} \,\}^*, \; \exists w_1, \dots, w_n : \; w_1 \dots w_n u\#v_0 \in I, (w_i, z_i) \in \xrightarrow{\ell r} \big\}, \tag{1}$$

is regular.

Second, we consider the set of reduction rules $\{\, a\widetilde{a} \to \varepsilon \mid a \in \Sigma \,\}$. A word $\alpha \in (\Sigma \cup \widetilde{\Sigma})^*$ is said to be *reducible* to $\beta \in (\Sigma \cup \widetilde{\Sigma})^*$ if it can be transformed to β by zero or more such reductions. For any language K over $\Sigma \cup \widetilde{\Sigma} \cup \{\#\}$, denote by $\mathrm{red}(K)$ the set of all words over $\Sigma \cup \{\#\}$ to which some word from K can be reduced. Then we apply a known result on such reductions to show that the transformation from computation histories to actual words derived by the rewriting system preserves regularity.

Before getting to the main track of the proof, let us establish a useful auxiliary statement on reduction rules of this kind.

Lemma 1. *For each $\alpha \in (\Sigma \cup \widetilde{\Sigma})^*$ and $x, u \in \Sigma^*$, if $\alpha\widetilde{x}$ is reducible to u, then α is reducible to ux.*

Proof. Let us first show for $a \in \Sigma$ that if $\alpha\widetilde{a}$ is reducible to u, then α is reducible to ua. The symbol $\widetilde{a}$ gets cancelled at some point, that is, $\alpha\widetilde{a}$ is reduced to $\alpha' a\widetilde{a}$, then to α' and finally to u. By the same sequence of reduction steps, α can be reduced to $\alpha' a$ and then to ua.

The main statement is proved by induction on $|x|$. The basis trivially holds. For the induction step, let $y = ax$ and suppose $\alpha\widetilde{y} = \alpha\widetilde{x}\widetilde{a}$ is reducible to u. By the above claim, then $\alpha\widetilde{x}$ is reducible to ua. By the induction hypothesis, this implies that α is reducible to $uax = uy$. □

Our first claim is the following:

Lemma 2. *A word $u\#v \in \Sigma^*\#\Sigma^*$ is derivable in the R-rewriting system if and only if there exists $\alpha \in L_0$ reducible to $u\#v$.*

Proof. First we assume that $u\#v$ is derivable in zero or more steps, and show, by induction on the length of the derivation of $u\#v$, that there exists $\alpha \in L_0$ reducible to $u\#v$.

Basis. If the derivation is of length 0, then $u\#v \in I$ and hence $u\#v \in L_0$.

Induction step. Suppose $u\#v$ is derivable in one or more steps from some $u_0\#v_0 \in I$, and consider the last step in the derivation. If a left-to-right rewriting is performed, that is,

$$u_0\#v_0 \Longrightarrow \ldots \Longrightarrow wu\#v' \Longrightarrow u\#v'z \quad (\text{where } (w,z) \in \xrightarrow{\ell r}),$$

then, by the induction hypothesis, there exists $\alpha = wu\#v_0\rho_0 z_1\rho_1 \ldots z_n\rho_n \in L_0$ reducible to $wu\#v'$. The same sequence of reductions reduces $\alpha' = u\#v_0\rho_0 z_1\rho_1 \ldots z_n\rho_n z$ to $u\#v'z$. Also, since α is of the form (1), α' is of the same form, with $w_{n+1} = w$, $z_{n+1} = z$ and $\rho_{n+1} = \varepsilon$. Therefore, $\alpha' \in L_0$, which proves this case.

Now suppose the last step in the derivation is a local rewriting on the right:

$$u_0\#v_0 \Longrightarrow \ldots \Longrightarrow u\#v'x \Longrightarrow u\#v'y \quad (\text{where } (x,y) \in \xrightarrow{rr}).$$

By the induction hypothesis, there is $\alpha = u\#v_0\rho_0 z_1\rho_1 \ldots z_n\rho_n \in L_0$ reducible to $u\#v'x$. Using the same sequence of reductions, $\alpha' = u\#v_0\rho_0 z_1\rho_1 \ldots z_n\rho_n\widetilde{x}y$ is reduced to $u\#v'x\widetilde{x}y$, which can be further reduced to $u\#v'y$. This α' is also of the form (1), with $\rho'_n = \rho_n\widetilde{x}y$, which means $\alpha' \in L_0$, completing the proof of this part.

Let us now establish the converse implication. Consider $\alpha = u\#v_0\rho_0 z_1\rho_1 \ldots z_{n-1}\rho_{n-1}z_n\rho_n \in L_0$, where, as in (1), there exist $w_1, \ldots, w_n \in \Sigma^*$, such that $(w_i, z_i) \in \xrightarrow{\ell r}$ for all i and $w_1 \ldots w_n u\#v_0 \in I$. Suppose α is reducible to a word $u\#v \in \Sigma^*\#\Sigma^*$; it has to be proved that $u\#v$ is derivable. The proof is an induction on $n + |\rho_0 z_1\rho_1 \ldots z_{n-1}\rho_{n-1}z_n\rho_n|$.

Basis: $n = 0$, $\rho_0 = \varepsilon$. Then $u\#v = u\#v_0 \in I$, derivable in zero steps.

Induction step, case $\rho_n = \varepsilon$. Because $z_n \in \Sigma^*$, the letters of z_n cannot be deleted when reducing α, and so $v = v'z_n$ for a certain word $v' \in \Sigma^*$ obtained by reduction from $v_0\rho_0 z_1\rho_1 \ldots z_{n-1}\rho_{n-1}$. Let $\alpha' = w_n u\#v_0\rho_0 z_1\rho_1 \ldots z_{n-1}\rho_{n-1}$, and note that $\alpha' \in L_0$. The same sequence of reductions that transforms α to $u\#v'z_n$ can be used to transform α' to $w_n u\#v'$. Then, by the induction hypothesis, $w_n u\#v'$ is derivable, and $u\#v'z_n = u\#v$ can be derived from it using the pair $(w_n, z_n) \in \xrightarrow{\ell r}$.

Induction step, case $\rho_n = \rho'_n\widetilde{x}y$. Let $\alpha = \alpha'\widetilde{x}y$, where $\alpha' = u\#v_0\rho_0 z_1\rho_1 \ldots z_{n-1}\rho_{n-1}z_n\rho'_n \in L_0$ and $(x,y) \in \xrightarrow{rr}$. Then $v = v'y$, where $v' \in \Sigma^*$ is obtained by reduction from $v_0\rho_0 z_1\rho_1 \ldots z_{n-1}\rho_{n-1}z_n\rho'_n\widetilde{x}$. Using the

same steps of reduction, $\alpha'\widetilde{x}$ is reducible to $u\#v'$. By Lemma 1, this implies that α' is reducible to $u\#v'x$. By the induction hypothesis, $u\#v'x$ is hence derivable, and using the rule $(x, y) \in \xrightarrow{rr}$, the word $u\#v$ can be derived next. □

Next, we show regularity of the language of computation histories.

Lemma 3. *The language L_0 is regular. A deterministic finite automaton recognizing it can be effectively constructed, provided that the images of regular languages under $\xrightarrow{\ell r}$ are computable.*

Proof. Since I is regular, it is a finite union of languages of the form $K\#M$, where K and M are regular languages over Σ, so we can assume that $I = K\#M$. Let $(\Sigma, Q, q_0, \delta, F)$ be a DFA recognizing K. For $p, q \in Q$, we denote by $K_{p,q}$ the language $\{\, w \in \Sigma^* \mid \delta(p, w) = q \,\}$ and by $K_{p,F}$ the language $\{\, w \in \Sigma^* \mid \delta(p, w) \in F \,\}$. Consider the alphabet $\Gamma = (Q \times Q) \cup \{r\}$ and define a regular language N over Γ as follows:

$$N = r^*(\{q_0\} \times Q)\Gamma^* \setminus \bigcup_{\substack{p,q \in Q \\ p \neq q}} \Gamma^*(Q \times \{p\})r^*(\{q\} \times Q)\Gamma^*$$

Each occurrence of the letter r in this language represents one rewriting step of $\xrightarrow{rr}$, and each letter (p, q) stands for reading a word belonging to $K_{p,q}$ from the left and appending an appropriate word to the right according to $\xrightarrow{\ell r}$. Further, we define a regular substitution φ from Γ^* to $(\Sigma \cup \widetilde{\Sigma})^*$ by the rules $\varphi((p, q)) = \xrightarrow{\ell r}(K_{p,q})$, for $p, q \in Q$, and $\varphi(r) = \{\, \widetilde{x}y \mid x \xrightarrow{rr} y \,\}$.

Let us show that the following regular language

$$L_1 = K\#M\varphi(r^*) \cup \bigcup_{q \in Q} K_{q,F}\#M\varphi(N \cap \Gamma^*(Q \times \{q\})r^*) \subseteq (\Sigma \cup \widetilde{\Sigma})^*\#(\Sigma \cup \widetilde{\Sigma})^*$$

is equal to L_0.

Let us take any word $w = u\#v_0\rho_0z_1\rho_1 \dots z_n\rho_n$ from L_0 and consider words $w_1, \dots, w_n$ satisfying $w_1 \dots w_n u \in K$ and $(w_i, z_i) \in \xrightarrow{\ell r}$. Define $q_i = \delta(q_0, w_1 \dots w_i)$, for $i = 1, \dots, n$. Then w belongs to the language

$$K_{q_n,F}\#M\varphi(r^*(q_0, q_1)r^*(q_1, q_2)r^* \dots (q_{n-1}, q_n)r^*)$$

and consequently also to L_1. Conversely, every element of N belongs to the language $r^*(q_0, q_1)r^*(q_1, q_2)r^* \dots (q_{n-1}, q_n)r^*$ for some states $q_1, \dots, q_n \in Q$, where $n \geq 0$. Therefore each word from L_1 lies in a language of the form

$$K_{q_n,F}\#M\varphi(r)^*z_1\varphi(r)^*z_2\varphi(r)^* \dots z_n\varphi(r)^*,$$

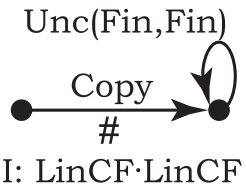
where for each $i \in \{1, \dots, n\}$ there exists $w_i \in K_{q_{i-1},q_i}$ such that $(w_i, z_i) \in \xrightarrow{\ell r}$. Because every word from the language $K_{q_0,q_1}K_{q_1,q_2} \dots K_{q_{n-1},q_n}K_{q_n,F}$ belongs to K, we can immediately see that all words from L_1 actually belong to L_0. □

Proof of Theorem 1. According to Lemma 2, the language L generated by the rewriting system equals $\mathrm{red}(L_0)$. By Lemma 3, the language L_0 is regular. This implies that $\mathrm{red}(L_0)$ is regular, due to the known results on reduction in free groups dating back to Benois [2], see Sakarovitch [14, Ch. II, Sec. 6], Hofbauer and Waldmann [5], as well as the authors [7]. We have thus proved that the language L is regular. □

3.2 The case of nonregular I

If we consider nonregular sets of initial words, then, even for very simple relations $\xrightarrow{\ell r}$, R-rewriting systems become computationally universal.

Theorem 2. *For every recursively enumerable language L_0 over an alphabet Σ there exist an alphabet Σ' and an R-rewriting system over Σ' given by a copy relation $\xrightarrow{\ell r}$, a finite uncontrolled relation $\xrightarrow{rr}$ and a language I that is a concatenation of two linear context-free languages, such that the language generated by the rewriting system equals $\#L_0$ modulo intersection with $\#\Sigma^*$.*



Proof. Using the method of Baker and Book [1], one can construct an alphabet $\Gamma \supseteq \Sigma$ and linear context-free languages K and M over Γ, such that $L_0 = MK^{-1}$. Define $\Sigma' = \Gamma \cup \widetilde{\Gamma}$, where $\widetilde{\Gamma} = \{\, \widetilde{a} \mid a \in \Gamma \,\}$, and keep the tilde notation from the proof of Theorem 1. Let $\xrightarrow{\ell r} = \{\, (\widetilde{a}, \widetilde{a}) \mid a \in \Gamma \,\}$ be a copy relation on $\widetilde{\Gamma}$; consider the set $X = \{\, a\widetilde{a} \mid a \in \Gamma \,\} \cup \{\varepsilon\}$ and define $\xrightarrow{rr} = X \times X$. We take the initial set $I = \widetilde{K}\#M$, which is a concatenation of two linear context-free languages. We are going to prove that the language L generated by this rewriting system satisfies $L \cap \#\Sigma^* = \#MK^{-1}$. To show that all words in $\#MK^{-1}$ can be generated, one can use induction with respect to the length of a word $u \in \Sigma^*$ to prove that if some word $\widetilde{u}\#vu$ is derivable, then the word $\#v$ is derivable too. The induction step can be performed as follows:

$$\widetilde{ua}\#vua = \widetilde{a}\widetilde{u}\#vua \overset{\ell r}{\Longrightarrow} \widetilde{u}\#vua\widetilde{a} \overset{rr}{\Longrightarrow} \widetilde{u}\#vu$$

Conversely, it is easy to see that all words $u\#v$ generated by our system satisfy $\mathrm{red}(\{vu\}) \subseteq MK^{-1}$, and therefore $L \cap \#\Sigma^* \subseteq \#MK^{-1} = \#L_0$. □

Corollary 1. *There exist a finite relation $\xrightarrow{\ell r}$, a finite uncontrolled relation $\xrightarrow{rr}$ and a language I recognized by a three-turn pushdown automaton such that the language generated by the rewriting system is r.e.-complete.*

If $\xrightarrow{\ell r}$ is uncontrolled or the initial set is linear context-free, the exact power of such rewriting remains unknown.

4 Sending

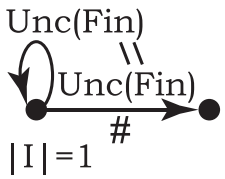
This section is devoted to *one-way S-rewriting* systems, in which we are allowed to delete and add words at the beginning of the word or delete a word at the beginning and append some word to the end. According to our two stacks analogy, the first stack can do internal processing, as well as send data to the second stack. Since the second stack only receives data and cannot do any internal processing, it can be understood as the output. Then our rewriting system behaves as a special case of a pushdown automaton without internal states which starts working in configurations given by the set of initial words I.

S-rewriting also has an explanation in terms of R-rewriting. Observe that the derivability relations $\Longrightarrow^*$ of S-rewriting systems are in principle just inverses of those of R-rewriting systems. More precisely, for a given R-rewriting system we can construct an S-rewriting system by taking $x \xrightarrow{\ell r} y$ whenever $y^{\mathrm{R}} \xrightarrow{\ell r} x^{\mathrm{R}}$ in the original system, and $x \xrightarrow{\ell\ell} y$ whenever $y^{\mathrm{R}} \xrightarrow{rr} x^{\mathrm{R}}$. Then it is easy to see that $w_0 \Longrightarrow^* w$ in the S-rewriting system if and only if $w^{\mathrm{R}} \Longrightarrow^* {w_0}^{\mathrm{R}}$ in the R-rewriting system.

4.1 The case of uncontrolled $\xrightarrow{\ell r}$

Let us first suppose that the sending relation $\xrightarrow{\ell r}$ is uncontrolled. The following example gives a weakest system of this kind, in which the processing relation $\xrightarrow{\ell\ell}$ is also uncontrolled: however, a nonregular language can still be produced.

Example 1. *Let* $\Sigma = \{a, b\}$ *and let* $\xrightarrow{\ell\ell} = \xrightarrow{\ell r} = \{a, aab\} \times \{a, aab\}$, *i.e.,* $\xrightarrow{\ell\ell}, \xrightarrow{\ell r} \in Unc(Fin)$ *and the same two-element set is used for both relations, which is denoted in the diagram on the right by the equality sign. Let* $I = \{ab\#\}$. *Then the set* L *of derivable words is nonregular.*



Let us show that L is nonregular. Notice that every word in L has the same number of occurrences of a and of b, since the initial word has this property and each step of rewriting preserves it. On the other hand, every word $ab^n\#a^{n-1}$, for $n \geqslant 1$, can be inductively derived from $ab\#$ as follows:

$$\underline{a}b^n\#a^{n-1} \xRightarrow{\ell\ell} \underline{a}ab \cdot b^n\#a^{n-1} \xRightarrow{\ell r} abb^n\#a^{n-1} \cdot a = ab^{n+1}\#a^n.$$

Every word of the form $ab^n\#a^{n-1}$ derives $b^n\#a^n$ by a single application of $\xRightarrow{\ell r}$, and therefore $L \cap b^*\#a^* = \{\, b^n\#a^n \mid n \geqslant 1 \,\}$, which proves the nonregularity of L.

Though the language generated in Example 1 is nonregular, it is linear context-free. It turns out that even if the relation $\overset{\ell\ell}{\to}$ is controlled, the generated languages are still only linear context-free. This will be proved later in Theorem 5.

If the initial set I can be linear context-free, we can generate non-linear-context-free languages even without using $\overset{\ell\ell}{\to}$.

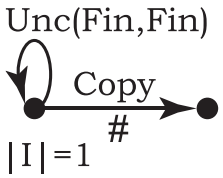
Example 2. *Let $\Sigma = \{a, b\}$, let $I = \{\, a^n b^n \# \mid n \geqslant 0 \,\}$, let $\overset{\ell r}{\to} = \{(a, a)\}$ (i.e., $\overset{\ell r}{\to} \in Unc(Fin) \cap Copy$) and let $\overset{\ell\ell}{\to} = \varnothing$. Then the rewriting system generates the language $\{\, a^i b^n \# a^{n-i} \mid 0 \leqslant i \leqslant n \,\}$, which is a known non-linear context-free language.*

Let us turn to the second case of a controlled sending relation.

4.2 The case of controlled $\overset{\ell r}{\to}$

Here even in the case of an uncontrolled $\overset{\ell\ell}{\to}$ every context-free language can be generated.

Theorem 3. *For every context-free language $L_0 \subseteq \Sigma^*$ there exists an S-rewriting system over an alphabet $\Gamma \supseteq \Sigma$ given by relations $\overset{\ell\ell}{\to} \in Unc(Fin, Fin)$ and $\overset{\ell r}{\to} \in Copy$, and a singleton initial set I, which generates $\#L_0$ modulo intersection with $\#\Gamma^*$. Given a context-free grammar for L_0, this rewriting system can be effectively constructed.*



Let us first define the construction. Assume the language is given by a context-free grammar $G = (\Sigma, N, P, S)$ in Chomsky normal form with possible ε-productions, that is, every production in P is of the form $A \to BC$ or $A \to w$, with $A, B, C \in N$, $w \in \Sigma^*$. Let $\overset{G/lm}{\Longrightarrow}$ be the relation of one-step derivability in G by rewriting the leftmost nonterminal, and let $\overset{G/lm}{\Longrightarrow}{}^*$ be its reflexive and transitive closure. Consider the alphabet $\Gamma = \Sigma \cup N \cup \widehat{N} \cup \widetilde{N}$, where $\widehat{N} = \{\, \widehat{A} \mid A \in N \,\}$ and $\widetilde{N} = \{\, \widetilde{A} \mid A \in N \,\}$, and construct the following two finite sets:

$$X = N \cup \{\, \widetilde{A}\widehat{A} \mid A \in N \,\} \tag{2a}$$

$$Y = \{\, B\widehat{B}C\widehat{C}\widetilde{A} \mid A \to BC \in P \,\} \cup \{\, w\widetilde{A} \mid A \to w \in P \,\} \cup \{\varepsilon\} \tag{2b}$$

Define $I = \{S\widehat{S}\#\}$, $\overset{\ell\ell}{\to} = X \times Y$ and let $a \overset{\ell r}{\to} a$ for all $a \in \Sigma$.

The proof of correctness of this construction begins with the following simulation of generation in S by the rewriting system:

Claim 3.1. *If $S \overset{G/lm}{\Longrightarrow}{}^* uA_1 \dots A_n$, where $u \in \Sigma^*$ and $A_1, \dots, A_n \in N$, then there exist $\theta_1, \dots, \theta_n \in \{\, \widetilde{D}\widehat{D} \mid D \in N \,\}^*$, such that the word*

$$A_1\widehat{A}_1\theta_1 A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n \# u \tag{3}$$

is derivable in the rewriting system.

Proof. The statement is proved by induction on the length of the generation of $uA_1 \dots A_n$ in G.

Basis. S is generated by G in 0 steps, and $S\widehat{S}\# \in I$.

Induction step, production $A_1 \to BC$**.** Let $S \overset{G/lm}{\Longrightarrow}{}^* uA_1A_2 \dots A_n$ and let $uA_1A_2 \dots A_n \overset{G/lm}{\Longrightarrow} uBCA_2 \dots A_n$ by a production $A_1 \to BC \in P$. By the induction hypothesis, a word

$$A_1\widehat{A}_1\theta_1 A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n\#u,$$

for some $\theta_1, \dots, \theta_n \in \{\, \widetilde{D}\widehat{D} \mid D \in N \,\}^*$, is derivable in the rewriting system. Since $A_1 \to BC \in P$, by construction $A_1 \in X$ and $B\widehat{B}C\widehat{C}\widetilde{A}_1 \in Y$, hence we derive

$$B\widehat{B}C\widehat{C}\underbrace{\widetilde{A}_1\widehat{A}_1\theta_1}_{\theta'_1} A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n\#u \tag{4}$$

by $\overset{\ell\ell}{\to}$. Since the factor $\widetilde{A}_1\widehat{A}_1\theta_1$ is in $\{\, \widetilde{D}\widehat{D} \mid D \in N \,\}^*$, it can be taken as θ'_1, which shows that (4) is of the form (3) corresponding to $uBCA_2 \dots A_n$.

Induction step, production $A_1 \to w$**.** Let $S \overset{G/lm}{\Longrightarrow}{}^* uA_1A_2 \dots A_n$ and then $uA_1A_2 \dots A_n \overset{G/lm}{\Longrightarrow} uwA_2 \dots A_n$ by $A_1 \to w \in P$. Again, by the induction hypothesis, a word

$$A_1\widehat{A}_1\theta_1 A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n\#u,$$

for some $\theta_1, \dots, \theta_n \in \{\, \widetilde{D}\widehat{D} \mid D \in N \,\}^*$, is derivable in the rewriting system. Because $A_1 \in X$ and $w\widetilde{A}_1 \in Y$, by $\overset{\ell\ell}{\to}$ we derive

$$w\widetilde{A}_1\widehat{A}_1\theta_1 A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n\#u.$$

By the copying rule $\overset{\ell r}{\to}$ applied $|w|$ times we obtain

$$\widetilde{A}_1\widehat{A}_1\theta_1 A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n\#uw.$$

It remains to use $|\theta_1|/2 + 1$ times the relation $\overset{\ell\ell}{\to}$ (via $\widetilde{D}\widehat{D} \in X$ and $\varepsilon \in Y$) to erase $\widetilde{A}_1\widehat{A}_1\theta_1$, obtaining the required word

$$A_2\widehat{A}_2\theta_2 \dots A_n\widehat{A}_n\theta_n\#uw.$$

This proves Claim 3.1. □

In order to show that only words generated by G can be produced by our rewriting, let us denote by $\mathrm{d}(\alpha)$ the word obtained from $\alpha \in (\Gamma \cup \{\#\})^*$ by deleting all occurrences of elements of $\widehat{N} \cup \widetilde{N}$. The following claim states that every word α derivable in our rewriting system either corresponds to a sentential form of G, or it cannot be further rewritten to get a word belonging to the language $\#\Gamma^*$, and so it has no impact on the intersection of the generated language with $\#\Gamma^*$.

Claim 3.2. *Let $\alpha \in (\Gamma \cup \{\#\})^*$ be any word derivable in the rewriting system such that some word belonging to $\#\Gamma^*$ can be derived from α. Then α belongs to the language*

$$M = \Sigma^* \{\, A\widehat{A}, \widetilde{A}\widehat{A} \mid A \in N \,\}^* \#\Sigma^*.$$

In addition, if we denote the words to the left and to the right from # in $\mathrm{d}(\alpha)$ by $vA_1 \dots A_n$ and u, respectively, where $u, v \in \Sigma^*$ and $A_1, \dots, A_n \in N$, i.e. $\mathrm{d}(\alpha) = vA_1 \dots A_n \# u$, then $uvA_1 \dots A_n$ can be generated in G.

Proof. We verify both statements of this claim simultaneously by induction on the length of the derivation of α.

Basis. The initial word $\alpha = S\widehat{S}\#$ clearly satisfies the claim.

Induction step. Assume we have a word $\alpha = v\theta\#u \in M$, where $u, v \in \Sigma^*$, $\theta \in \{\, A\widehat{A}, \widetilde{A}\widehat{A} \mid A \in N \,\}^*$ and $\mathrm{d}(uv\theta)$ can be generated in G. We take any word β obtained from α by one step of the rewriting such that some word from $\#\Gamma^*$ is derivable from β. We have to show that β belongs to M and that the word over $\Sigma \cup N$ constructed from $\mathrm{d}(\beta)$ can be generated in G. This trivially holds when β is obtained from α by applying a rule from $\xrightarrow{\ell r}$. If some rule from $\xrightarrow{\ell\ell}$ is applied to α, then certainly $v = \varepsilon$. Now we have to distinguish two cases according to the first letter of θ, and consider all possible rules which can be applied.

First assume that $\alpha = A\widehat{A}\eta\#u$, where $A \in N$ and $\eta \in \{\, A\widehat{A}, \widetilde{A}\widehat{A} \mid A \in N \,\}^*$. If A is rewritten to $B\widehat{B}C\widehat{C}\widetilde{A}$ for a certain $A \to BC \in P$ or to $w\widetilde{A}$ for $A \to w \in P$, then β is of the required form. If A is replaced by a word ending with $\widetilde{D}$, where $D \in N$, $D \neq A$, then β contains a factor $\widetilde{D}\widehat{A}$ to the left from #, which can never be removed, and therefore no element of $\#\Gamma^*$ can be derived from β. Otherwise, the rule $A \xrightarrow{\ell\ell} \varepsilon$ is used, producing the word $\beta = \widehat{A}\eta\#u$, to which no rule of the system can be applied, and so elements of $\#\Gamma^*$ are not derivable from β.

Now assume that $\alpha = \widetilde{A}\widehat{A}\eta\#u$. Rewriting $\widetilde{A}\widehat{A}$ to the empty word produces β satisfying the claim because $\mathrm{d}(\beta) = \mathrm{d}(\alpha)$. Otherwise, $\widetilde{A}\widehat{A}$ is rewritten to a word ending with $\widetilde{D}$ for a certain $D \in N$. Therefore there is an occurrence of $\widetilde{D}$ in β to the left of #, which is not immediately followed by $\widehat{D}$. Since this occurrence of $\widetilde{D}$ cannot be removed by our rewriting rules, no word starting with # is derivable from β. This concludes the proof of Claim 3.2. □

Returning to the proof of Theorem 3, it can now be established that a word $\alpha \in \#\Gamma^*$ is derivable in the rewriting system if and only if $\alpha = \#w \in \#\Sigma^*$ and $w \in L(G)$. Indeed, Claim 3.2 is applicable to any derivable word $\alpha \in \#\Gamma^*$, giving that $\alpha = \#w \in \#\Sigma^*$ for some w generated in G. Conversely, if $w \in L(G)$, then there is a leftmost derivation $S \overset{G/lm}{\Longrightarrow}{}^* w$, and, using Claim 3.1 with $n = 0$, we conclude that the word $\#w$ is generated in the rewriting system. Q. E. D.

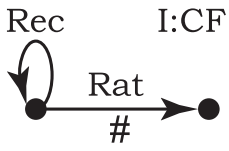
As a by-product we obtain a peculiar normal form for pushdown automata.

Corollary 2. *Every context-free language over an alphabet Σ is recognized by a nondeterministic pushdown automaton with pushdown alphabet $\Gamma \supseteq \Sigma$, with two states, q_0 and q_1 (of which q_0 is the only initial state and the only accepting state), and with the following transitions:*

- *(for every $a \in \Sigma$) from q_0 to q_0, reading an input symbol a and popping the corresponding pushdown symbol a from the stack;*
- *(for every word $x_i \in \Gamma^*$ from a finite list) from q_0 to q_1, popping x_i and not touching the input;*
- *(for every word $y_j \in \Gamma^*$ from a finite list) from q_1 to q_0, pushing y_j and not touching the input.*

If the least controlled S-rewriting yields all context-free languages, what could be its expressive power in a fully controlled case? It will now be proved that, in fact, still nothing but the context-free languages can be generated.

Theorem 4. *Let $\overset{\ell\ell}{\to}$ be a recognizable relation, let $\overset{\ell r}{\to}$ be a rational relation, let I be a context-free language. Then the language generated by this rewriting system is context-free. Given a finite automaton for $\overset{\ell\ell}{\to}$, a finite transducer for $\overset{\ell r}{\to}$ and a context-free grammar for I, a context-free grammar for the generated language can be effectively constructed.*



The proof is based upon two results, which are interesting on their own. One of them is the known closure of the context-free languages under the cyclic shift operation.

Lemma 4 (Oshiba, 1972 [11]; Maslov, 1973 [10]; Hopcroft and Ullman, 1979 [6, solution to Ex. 6.4c])**.** *For every context-free language L, the language* $\textsc{shift}(L) = \{\, uv \mid vu \in L \,\}$ *is context-free. Given a context-free grammar for L, a context-free grammar for* $\textsc{shift}(L)$ *can be effectively constructed.*

Another key property is that pushdown automata with initial stack contents defined by a context-free language still recognize only context-free languages.

Lemma 5. *Let Σ and Γ be two alphabets, and let L_0 be a context-free language over Γ. Let $\mathcal{A} = (\Sigma, \Gamma, Q, q_0, \delta, F)$ be a nondeterministic pushdown automaton without the initial pushdown symbol, where $\delta \colon Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$. Define*

$$L(\mathcal{A}) = \{\, w \mid \exists x \in L_0 \; \exists q_F \in F : \; (q_0, w, x) \vdash^* (q_F, \varepsilon, \varepsilon) \,\}.$$

Then $L(\mathcal{A})$ is context-free and a standard PDA recognizing this language can be effectively constructed.

This can be regarded as a system of two cooperating pushdown automata, in which one PDA supplies the initial contents of the pushdown to the other. We prove that such a superposition is no more powerful than a single PDA.

Another interpretation of this result is the following. Let us say that each PDA $\mathcal{A}$ defines a relation $R_{\mathcal{A}} \subseteq \Gamma^* \times \Sigma^*$ between its pushdown and its input. This relation is defined as $R_{\mathcal{A}}(x, w)$ if and only if $(q_0, w, x) \vdash^* (q_F, \varepsilon, \varepsilon)$ for some $q_F \in F$, that is, $\mathcal{A}$ accepts w from state q_0 and with the initial pushdown contents x. Then our lemma states that every such relation preserves context-freeness.

One more interpretation is that the word given in the pushdown is a "certificate" of the membership of the input word, and we prove that an access to a context-free set of certificates does not give a PDA any extra computational power.

Proof of Lemma 5. It is convenient to assume that L_0 is given by a context-free grammar $G = (\Gamma, N, P, S)$. Define $\Theta = \Gamma \cup N$.

Construct a new PDA (in the ordinary sense) $\mathcal{B} = (\Sigma, \Theta, Q, q_0, \delta', F, S)$ with the same states as $\mathcal{A}$, with a pushdown alphabet Θ, with $S \in \Theta$ as the initial pushdown symbol and with transitions defined as follows:

$$\delta'(q, u, s) = \delta(q, u, s) \quad (\text{for all } q \in Q,\ u \in \Sigma \cup \{\varepsilon\} \text{ and } s \in \Gamma) \tag{5a}$$

$$\delta'(q, \varepsilon, A) = \{\, (q, \alpha) \mid A \to \alpha \in P \,\} \quad (\text{for all } q \in Q \text{ and } A \in N) \tag{5b}$$

$$\delta'(q, a, A) = \varnothing \quad (\text{for all } q \in Q,\ a \in \Sigma \text{ and } A \in N) \tag{5c}$$

The idea is that symbols of an initial pushdown word are generated upon demand. In the beginning, a leftmost derivation in G from S is simulated at the top of the pushdown until any terminal symbol from Γ is generated. This first symbol of the initial pushdown word is used to operate $\mathcal{A}$. Once the simulated $\mathcal{A}$ empties its pushdown and requires the next symbol of the initial pushdown word, a nonterminal from N appears at the top of the pushdown, and a leftmost derivation in G is simulated until another symbol from Γ is generated. For the computation to be accepting, the simulated derivation in G should eventually eliminate all symbols from N in the pushdown, and conclude with a pure computation of $\mathcal{A}$.

It is claimed that a configuration (q, w, β) of $\mathcal{B}$, in which $q \in Q$, $w \in \Sigma^*$ and $\beta \in \Theta^*$, leads to acceptance if and only if there exists $y \in L_G(\beta)$, such that the configuration (q, w, y) of $\mathcal{A}$ leads to acceptance.

$\Leftrightarrow$ Induction on the length of an accepting computation of $\mathcal{B}$ starting from (q, w, β).

Basis. If (q, w, β) is an accepting configuration of $\mathcal{B}$, then $q \in F$, $w = \varepsilon$ and $\beta = \varepsilon$. Taking $y = \varepsilon$, we show that (q, w, y) is an accepting configuration of $\mathcal{A}$.

Induction step. Consider the first step in the given accepting computation of $\mathcal{B}$. Suppose it is an application of a production from P, that

is,

$$(q, w, A\beta') \overset{\mathcal{B}}{\vdash} (q, w, \alpha\beta') \overset{\mathcal{B}}{\vdash} \ldots \overset{\mathcal{B}}{\vdash} \text{Acc.}$$

By the induction hypothesis, there exists $y \in L_G(\alpha\beta')$, such that (q, w, y) leads $\mathcal{A}$ to acceptance. Since $L_G(\alpha) \subseteq L_G(A)$, $y \in L_G(\alpha\beta')$ implies $y \in L_G(A\beta')$, and the required conditions are met.

Consider the other case, when the given computation of $\mathcal{B}$ starts with simulating one step of $\mathcal{A}$:

$$(q, uv, s\beta') \overset{\mathcal{B}}{\vdash} (q', v, z\beta') \overset{\mathcal{B}}{\vdash} \ldots \overset{\mathcal{B}}{\vdash} \text{Acc.}$$

By the induction hypothesis, there exists $y \in L_G(z\beta')$, such that (q', v, y) leads $\mathcal{A}$ to acceptance. Note that $y = zy'$, where $y' \in L_G(\beta')$, and construct the computation of $\mathcal{A}$, which starts as $(q, uv, sy') \overset{\mathcal{A}}{\vdash} (q', v, zy') = (q', v, y)$ and then proceeds to acceptance, completing the proof of this implication.

$\Leftarrow$ Induction on the pair $(\ell_{\mathcal{A}}, \ell_G)$, where $\ell_{\mathcal{A}}$ is the length of accepting computation of $\mathcal{A}$ from (q, w, y), while ℓ_G is the length of generation of y from β in G.

Basis. If $y = \beta$ and (q, w, y) is an accepting configuration, then $w = \varepsilon$ and $y = \varepsilon$, and (q, w, β) is an accepting configuration of $\mathcal{B}$ as well.

In the induction step, the word β is always nonempty, because $\beta = \varepsilon$ implies $y = \varepsilon$, which obviously implies $\ell_G = 0$, and in addition, since $\mathcal{A}$ cannot perform any transitions with the empty pushdown, $\ell_{\mathcal{A}} = 0$. We shall consider two cases, depending on what the first symbol of β is.

Induction step, case $\beta = A\beta'$, for $A \in N$. Let (q, w, y) be a configuration that leads $\mathcal{A}$ to acceptance and let $y \in L_G(A\beta')$. Consider the first step in the leftmost generation of y in G: $A\beta' \overset{G}{\Longrightarrow} \alpha\beta' \overset{G}{\Longrightarrow}{}^* y$. This generation can be simulated by $\mathcal{B}$ using (5b): $(q, w, A\beta') \vdash (q, w, \alpha\beta')$. Since the word $y \in \Gamma^*$ is derived from $\alpha\beta'$ in fewer steps than y from β, the induction hypothesis can be applied to $(q, w, \alpha\beta')$, showing that $(q, w, A\beta')$ leads $\mathcal{B}$ to acceptance.

Induction step, case $\beta = s\beta'$, for $s \in \Gamma$. In this case we have $y = sy'$ for a certain $y \in L_G(\beta')$. The first step in the accepting computation of $\mathcal{A}$ on (q, w, y) is $(q, uw', sy') \vdash (q', w', zy')$, where $w = uw'$, $u \in \Sigma \cup \{\varepsilon\}$, $w' \in \Sigma^*$ and $(q', z) \in \delta(q, u, s)$. This step can be directly repeated by $\mathcal{B}$ using (5a): $(q, uw', s\beta') \vdash (q', w', z\beta')$. By the induction hypothesis applied to $(q', w', z\beta')$, there exists an accepting computation of $\mathcal{B}$ from $(q', w', z\beta')$, which yields the required accepting computation from $(q, w, s\beta')$.

It follows that $\mathcal{B}$ accepts an input word $w \in \Sigma^*$ if and only if $\mathcal{A}$ accepts w for some initial pushdown word $x \in L_0$. □

Let us now simulate the derivation in an S-rewriting system by a computation of a certain pushdown automaton over a common input and pushdown alphabet $\Sigma \cup \{\$\}$, where $\$ \notin \Sigma$ is a new symbol. This PDA, denoted by $\mathcal{A}$,

is set to verify whether a word $x\#y$ can derive another word $u\#v$. Both words are supplied to $\mathcal{A}$, one as an input, the other as the initial pushdown contents; in addition, the components of each word are exchanged, that is, $u\#v$ is given as an input word $v\$u$, while $x\#y$ is given as a pushdown word $y\$x$.

The constructed PDA $\mathcal{A}$, illustrated in Figure 3, uses three principal states, q_0, q_1 and q_2, as well as some auxiliary states to simulate finite automata for $\xrightarrow{\ell\ell}$ and $\xrightarrow{\ell r}$. In the initial state q_0, the PDA reads input symbols, matching them to the symbols popped from the pushdown. The equality of the symbols is checked: the PDA is set to read a word y and at the same time to pop the same word y from the pushdown. This is done until the symbol \$ is encountered in the pushdown.

The main processing is done in q_1. Let $\xrightarrow{\ell\ell} = \bigcup_{i=1}^{n} X_i \times Y_i$, where X_i, Y_i are regular languages. Every time a rule from $\xrightarrow{\ell\ell}$ is to be applied, $\mathcal{A}$ nondeterministically chooses a number i and proceeds with popping any word $x \in X_i$ from the pushdown. The first symbol of x is at the top of the pushdown and should be popped first. Therefore, x is removed from the pushdown by simulating a finite automaton for X_i, with its input symbols drawn from the pushdown. Once the simulated automaton accepts, $\mathcal{A}$ should place any word $y \in Y_i$ onto the pushdown, the first symbol at the top. For that reason, the symbols of y should be pushed in the reverse order, which is done by simulating a finite automaton for $(Y_i)^R$. Every time this automaton requests an input symbol, such a symbol is pushed to the pushdown. Once the automaton for $(Y_i)^R$ accepts, $\mathcal{A}$ returns to state q_1.

To apply a rule from $\xrightarrow{\ell r}$, the PDA simulates a finite transducer implementing this relation. Every time the transducer requests an input symbol, the symbol is popped from the pushdown; every time the transducer prints a symbol, it is read from the input. Once the transducer accepts, the control returns to q_1.

Eventually the symbol \$ will be encountered in the input, and then $\mathcal{A}$ goes to the accepting state q_2, in which it reads and pops the same symbols exactly as it did in q_0. To reach acceptance, the remainder of the input has to coincide with the pushdown contents.

Let us formally define this PDA. Let the DFA for X_i be $(\Sigma, R_i, r_{i0}, \delta_i, F_i)$ and let the DFA for Y_i^R be $(\Sigma, R'_i, r'_{i0}, \delta'_i, F'_i)$. Let the nondeterministic finite transducer for $\xrightarrow{\ell r}$ be $(\Sigma, \Sigma, P, p_0, \widehat{\delta}, \widehat{F})$, in which $\widehat{\delta}\colon P \times \big((\{\varepsilon\} \times \Sigma) \cup (\Sigma \times \{\varepsilon\})\big) \to 2^P$. Then the PDA $\mathcal{A}$ has the set of states $\{q_0, q_1, q_2\} \cup P \cup \bigcup_{i=1}^{n} (R_i \cup R'_i)$, of which q_0 is the initial state and q_2 is the only finite state, and its

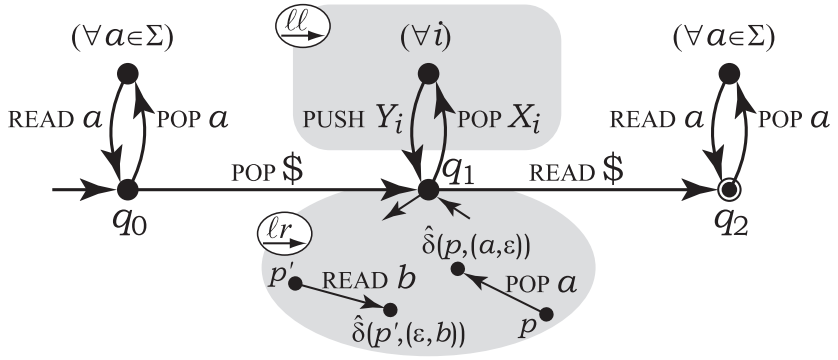


Figure 3: PDA $\mathcal{A}$ in the proof of Theorem 4.

transitions are defined as follows:

$$
\begin{aligned}
&(q_0, a, a) \to (q_0, \varepsilon) \quad (\text{for all } a \in \Sigma)\\
&(q_0, \varepsilon, \$) \to (q_1, \varepsilon)\\
&(q_1, \varepsilon, \varepsilon) \to (r_{i0}, \varepsilon) \quad (\text{for all } i = 1, \ldots, n)\\
&(r_i, \varepsilon, a) \to (\delta_i(r_i, a), \varepsilon) \quad (\text{for all } i = 1, \ldots, n,\ r_i \in R_i,\ a \in \Sigma)\\
&(r_i, \varepsilon, \varepsilon) \to (r'_{i0}, \varepsilon) \quad (\text{for all } i = 1, \ldots, n,\ r_i \in F_i)\\
&(r'_i, \varepsilon, \varepsilon) \to (\delta'_i(r'_i, a), a) \quad (\text{for all } i = 1, \ldots, n,\ r'_i \in R'_i,\ a \in \Sigma)\\
&(r'_i, \varepsilon, \varepsilon) \to (q_1, \varepsilon) \quad (\text{for all } i = 1, \ldots, n,\ r'_i \in F'_i)\\
&(q_1, \varepsilon, \varepsilon) \to (p_0, \varepsilon)\\
&(p, \varepsilon, a) \to (q, \varepsilon) \quad (\text{for all } p \in P,\ a \in \Sigma,\ q \in \widehat{\delta}(p, (a, \varepsilon)))\\
&(p, b, \varepsilon) \to (q, \varepsilon) \quad (\text{for all } p \in P,\ b \in \Sigma,\ q \in \widehat{\delta}(p, (\varepsilon, b)))\\
&(p, \varepsilon, \varepsilon) \to (q_1, \varepsilon) \quad (\text{for all } p \in \widehat{F})\\
&(q_1, \$, \varepsilon) \to (q_2, \varepsilon)\\
&(q_2, a, a) \to (q_2, \varepsilon) \quad (\text{for all } a \in \Sigma)
\end{aligned}
$$

The correctness of this construction is stated in the following lemma:

Lemma 6. *Consider a one-way S-rewriting system over an alphabet Σ as defined in Theorem 4, and construct PDA $\mathcal{A}$ according to $\overset{\ell\ell}{\to}$ and $\overset{\ell r}{\to}$, as described above. Then for every $x, y, u, v \in \Sigma^*$:*

I. $\mathcal{A}$ can reach an accepting configuration from a configuration $(q_1, v\$u, x)$ if and only if $x\# \Longrightarrow^* u\#v$;

II. $\mathcal{A}$ can reach an accepting configuration from a configuration $(q_0, v\$u, y\$x)$ if and only if $x\#y \Longrightarrow^* u\#v$.

Proof. Let us prove the first part of the lemma.

$\Leftarrow$ Suppose $\mathcal{A}$ has an accepting computation starting from $(q_1, v\$u, x)$; it has to be proved that $x\# \Longrightarrow^* u\#v$. The proof is by an induction on the number of visits to state q_1 in this accepting computation of $\mathcal{A}$.

Basis: no visits. Suppose the computation starts with a transition to q_2, in which \$ is read. Then $v = \varepsilon$, and for the rest of the computation to be accepting, u must be equal to x. Then $x\#$ derives $u\#$ in zero steps.

Induction step, first visit via X_i and Y_i. Let the computation start in $(q_1, v\$u, x)$ with implementing a transition by $\stackrel{\ell\ell}{\rightarrow}$. Let the pair (X_i, Y_i) be chosen, let $\widetilde{x} \in X_i$ be the word popped from the stack and let $\widetilde{y} \in Y_i$ be the word subsequently pushed to the stack. Then $x = \widetilde{x}x'$, and the configuration at the next visit to q_1 is $(q_1, v\$u, \widetilde{y}x')$.

The latter configuration, by assumption, leads to acceptance. Then, by the induction hypothesis, $\widetilde{y}x'\# \Longrightarrow^* u\#v$. Since $(\widetilde{x}, \widetilde{y}) \in \stackrel{\ell\ell}{\rightarrow}$, we can construct a derivation

$$\widetilde{x}x'\# \stackrel{\ell\ell}{\Longrightarrow} \widetilde{y}x'\# \Longrightarrow \ldots \Longrightarrow u\#v,$$

which proves this case.

Induction step, first visit via $\stackrel{\ell r}{\rightarrow}$. Now suppose the computation, having started in $(q_1, v\$u, x)$, first implements a transition by $\stackrel{\ell r}{\rightarrow}$. Then, for a certain pair $(\widetilde{x}, \widetilde{v}) \in \stackrel{\ell r}{\rightarrow}$ recognized by the given finite transducer, PDA $\mathcal{A}$ pops $\widetilde{x}$, reads $\widetilde{v}$ and gets back to q_1 in the configuration $(q_1, v'\$u, x')$, where $v = \widetilde{v}v'$ and $x = \widetilde{x}x'$.

Since this configuration leads to acceptance, $x'\# \Longrightarrow^* u\#v'$ by the induction hypothesis. Using the rule $(\widetilde{x}, \widetilde{v}) \in \stackrel{\ell r}{\rightarrow}$, and then repeating the same steps as in the first derivation, we obtain

$$\widetilde{x}x'\# \stackrel{\ell r}{\Longrightarrow} x'\#\widetilde{v} \Longrightarrow \ldots \Longrightarrow u\#\widetilde{v}v',$$

which completes the proof of the forward implication.

$\Rightarrow$ Let us prove the converse, that is, if $x\# \Longrightarrow^* u\#v$, then the configuration $(q_1, v\$u, x)$ leads $\mathcal{A}$ to acceptance. We use induction on the length of the derivation.

Basis: 0 steps. If $x = u$ and $v = \varepsilon$, the computation starting from $(q_1, \$u, x)$ proceeds to q_2 by reading \$ and then verifies the equality of x and u, accepting in the end.

Induction step, first step using $\stackrel{\ell\ell}{\rightarrow}$. Suppose the derivation starts as follows:

$$\widetilde{x}x'\# \stackrel{\ell\ell}{\Longrightarrow} \widetilde{y}x'\# \Longrightarrow \ldots \Longrightarrow u\#v,$$

where $(\widetilde{x}, \widetilde{y}) \in \stackrel{\ell\ell}{\rightarrow}$. By the induction hypothesis applied to the rest of the derivation, the configuration $(q_1, v\$u, \widetilde{y}x')$ leads the PDA to acceptance.

Then an accepting computation of the PDA starting from $(q_1, v\$u, \widetilde{x}x')$ can be constructed as follows. Let $(\widetilde{x}, \widetilde{y}) \in X_i \times Y_i$. From q_1, the new computation proceeds into the "POP X_i–PUSH Y_i" cycle of $\mathcal{A}$, in which $\widetilde{x}$ is

first popped from the pushdown, and then $\widetilde{y}$ is pushed. No symbols of the input are read. The resulting configuration $(q_1, v\$u, \widetilde{y}x')$, as we know, leads to acceptance.

Induction step, first step using $\xrightarrow{\ell r}$. Consider the case of the derivation starting as follows:

$$\widetilde{x}x'\# \overset{\ell r}{\Longrightarrow} x'\#\widetilde{v} \Longrightarrow \ldots \Longrightarrow u\#v,$$

for some $(\widetilde{x}, \widetilde{v}) \in \xrightarrow{\ell r}$. Since the symbols sent to the right part remain there, $v = \widetilde{v}v'$. By repeating the same steps, we can construct the derivation

$$x'\# \Longrightarrow \ldots \Longrightarrow u\#v',$$

to which the induction hypothesis is applicable, that is, $\mathcal{A}$ accepts from the configuration $(q_1, v'\$u, x')$.

Now let the PDA start from $(q_1, \widetilde{v}v'\$u, \widetilde{x}x')$. We start the new computation by simulating the finite transducer, to the effect that $\widetilde{x}$ is popped from the pushdown, while $\widetilde{v}$ is read from the input. In the end the configuration $(q_1, v'\$u, x')$ is obtained, from which acceptance can be reached. This completes the proof of the first part of the lemma.

The second part can be easily inferred from the first part. It is easy to see that one can reach acceptance from $(q_0, v\$u, y\$x)$ if and only if $v = yz$ and the configuration $(q_1, z\$u, x)$ can lead to an accepting configuration. Similarly, since in S-rewriting the symbols on the right are only stacked and never modified, $x\#y \Longrightarrow^* u\#v$ holds if and only if $v = yz$ and $x\# \Longrightarrow^* u\#z$. By the first part of the lemma, these two statements are equivalent. □

Now Lemmata 4–6 can be combined to obtain a succinct proof of our theorem on the power of S-rewriting.

Proof of Theorem 4. Consider the context-free initial set I of the S-rewriting system. Let $\$ \notin \Sigma$ and define

$$I' = \textsc{shift}(I\$) \cdot \{\#\}^{-1} = \{\, v_0\$u_0 \mid u_0\#v_0 \in I \,\}$$

By Lemma 4 and by the well-known closure of the context-free languages under quotient with regular languages, I' is context-free.

Construct PDA $\mathcal{A}$ as described before Lemma 6 and let I' be the language of its initial pushdown words. By definition, $v\$u \in L(\mathcal{A})$ if and only if

> there exists $v_0\$u_0 \in I'$, such that $(q_0, v\$u, v_0\$u_0)$ leads to acceptance.

According to the second part of Lemma 6, this is equivalent to the following:

> there exists $v_0\$u_0 \in I'$, such that $u_0\#v_0 \Longrightarrow^* u\#v$,

which means $u\#v \in L$, where L is the language generated by the given S-rewriting system. Therefore, $L(\mathcal{A}) = \{\, v\$u \mid u\#v \in L \,\}$.

Now, by Lemma 5, the language $L(\mathcal{A})$ is context-free and we can construct PDA $\mathcal{B}$ with a single initial pushdown symbol, such that $L(\mathcal{B}) = L(\mathcal{A})$. It remains to apply the cyclic shift to obtain $\textsc{shift}(L(\mathcal{B})\#) \cdot \{\$\}^{-1} = L$, and this language is context-free by the same closure properties as above. □

5 Receiving and sending

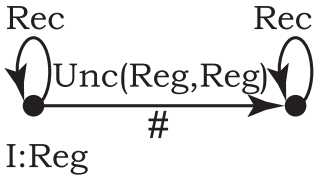
We shall now consider *one-way RS-rewriting*, which is the strongest type of one-way rewriting that uses the relations $\overset{\ell\ell}{\to}$ (local processing of the first stack), $\overset{\ell r}{\to}$ (one-way communication from the first stack to the second) and $\overset{rr}{\to}$ (local processing of the second stack).

As in the case of S-rewriting, the expressive power is mainly determined by whether $\overset{\ell r}{\to}$ is controlled or not. Let us consider two cases.

5.1 The case of uncontrolled $\overset{\ell r}{\to}$

In Example 1 we have already seen that uncontrolled S-rewriting is sufficient to generate nonregular languages. An uncontrolled relation $\overset{\ell r}{\to}$ was used there to communicate the *number* of symbols generated on the left to the right side, resulting in a most simple nonregular language. The most obvious computational model for replicating this behaviour is a counter automaton. We shall now see that the one-turn counter automata are sufficient to simulate any RS-rewriting system, as long as $\overset{\ell r}{\to}$ is uncontrolled.

Theorem 5. *Let $\overset{\ell r}{\to} \in Unc(Reg, Reg)$ be uncontrolled, let $\overset{\ell\ell}{\to}$ and $\overset{rr}{\to}$ be recognizable relations, let I be regular. Then the generated language can be recognized by a one-turn counter automaton, and, given finite automata for $\overset{\ell r}{\to}$, $\overset{\ell\ell}{\to}$, $\overset{rr}{\to}$ and I, this PDA can be effectively constructed.*



The proof generally follows the scheme used for Theorem 1: first we define the language of computation histories, then use reductions to obtain the language generated by the rewriting system. However, the use of one-turn counter automata instead of DFAs in this context is novel, so we can no longer rely upon any well-known results.

Let Σ be the original alphabet and define two copies: $\overrightarrow{\Sigma} = \{\, \overrightarrow{a} \mid a \in \Sigma \,\}$ and $\overleftarrow{\Sigma} = \{\, \overleftarrow{a} \mid a \in \Sigma \,\}$. For every word $w = a_1 \dots a_\ell \in \Sigma^*$, with $\ell \geqslant 0$, denote $\overrightarrow{w} = \overrightarrow{a}_\ell \dots \overrightarrow{a}_1$ and $\overleftarrow{w} = \overleftarrow{a}_\ell \dots \overleftarrow{a}_1$. Extend this notation to languages as $\overrightarrow{L} = \{\, \overrightarrow{w} \mid w \in L \,\}$ and $\overleftarrow{L} = \{\, \overleftarrow{w} \mid w \in L \,\}$ for every $L \subseteq \Sigma^*$. Consider the alphabet $\Sigma_3 = \Sigma \cup \overrightarrow{\Sigma} \cup \overleftarrow{\Sigma}$.

Let $L_\ell = \{\, y\overrightarrow{x} \mid x \xrightarrow{\ell\ell} y \,\}$, $L_r = \{\, \overleftarrow{x}\, y \mid x \xrightarrow{rr} y \,\}$ and $\xrightarrow{\ell r} = Z \times W$, and define the language of computation histories $L_0 \subseteq \Sigma_3^* \# \Sigma_3^*$ as follows:

$$L_0 = \big\{ \alpha_n \overrightarrow{z_n} \ldots \alpha_1 \overrightarrow{z_1} \alpha_0 u_0 \# v_0 \beta_0 w_1 \beta_1 \ldots w_n \beta_n \;\big|\; n \geqslant 0, u_0 \# v_0 \in I, \alpha_i \in L_\ell^*, \beta_i \in L_r^*, z_i \in Z, w_i \in W \big\} \tag{6}$$

Consider the following reduction rules on $(\Sigma_3 \cup \{\#\})^*$: $\overrightarrow{a}\, a \to \varepsilon$ and $a \overleftarrow{a} \to \varepsilon$, for all $a \in \Sigma$. A word $\alpha \in (\Sigma_3^* \cup \{\#\})$ is said to be *reducible* to $\beta \in (\Sigma_3 \cup \{\#\})^*$ if it can be transformed to β by zero or more such reductions. For every $L \subseteq (\Sigma \cup \overrightarrow{\Sigma})^* \# (\Sigma \cup \overleftarrow{\Sigma})^*$, denote by $\mathrm{red}(L)$ the set of words in $\Sigma^* \# \Sigma^*$ obtained by reducing words in L.

Lemma 7. *A word $u\#v \in \Sigma^*\#\Sigma^*$ is derivable in the system if and only if there exists $\eta\#\theta \in L_0$ reducible to $u\#v$.*

Proof. Let $u\#v$ be derivable in zero or more steps, and let us show that some $\eta\#\theta \in L_0$ is reducible to $u\#v$. The proof is an induction on the length of the derivation of $u\#v$.

Basis: derivation of length 0. Then $u\#v = u_0\#v_0 \in I$ and hence $u\#v \in L_0$.

Induction step. Consider the last step in the derivation of $u\#v$ from some $u_0\#v_0 \in I$. Suppose it is a left-to-right rewriting:

$$u_0\#v_0 \Longrightarrow \ldots \Longrightarrow zu\#v' \overset{\ell r}{\Longrightarrow} u\#v'w \quad (\text{where } z \xrightarrow{\ell r} w).$$

By the induction hypothesis, there exists $\eta\#\theta \in L_0$ reducible to $zu\#v'$. Then $\overrightarrow{z}\eta\#\theta w$ is in L_0, and it is reducible to $\overrightarrow{z}zu\#v'w$ using the same sequence of reductions. The latter can be reduced to $u\#v'w$.

If the last step in the derivation of $u\#v$ from $u_0\#v_0$ is a local rewriting on the right, then

$$u_0\#v_0 \Longrightarrow \ldots \Longrightarrow u\#v'x \overset{rr}{\Longrightarrow} u\#v'y \quad (\text{where } x \xrightarrow{rr} y).$$

Using the induction hypothesis, we obtain $\eta\#\theta \in L_0$, which can be reduced to $u\#v'x$. The same sequence of reductions reduces $\eta\#\theta\overleftarrow{x}\, y \in L_0$ to $u\#v'x\overleftarrow{x}\, y$, which is in turn reducible to $u\#v'y$.

The case when the last step in the derivation of $u\#v$ from $u_0\#v_0$ is a local rewriting on the left is proved symmetrically.

Next, we establish the converse implication. Let

$$\eta\#\theta = \alpha_n \overrightarrow{z_n} \ldots \alpha_1 \overrightarrow{z_1} \alpha_0 u_0 \# v_0 \beta_0 w_1 \beta_1 \ldots w_n \beta_n \in L_0$$

be reducible to a word $u\#v \in \Sigma^*\#\Sigma^*$. The derivability of this word in the rewriting system has to be shown. The proof is by induction on the length of $\eta\#\theta$.

Basis: $\eta\#\theta = u_0\#v_0$**.** Then $\eta\#\theta \in I$, and $u\#v = u_0\#v_0$ is derivable in zero steps.

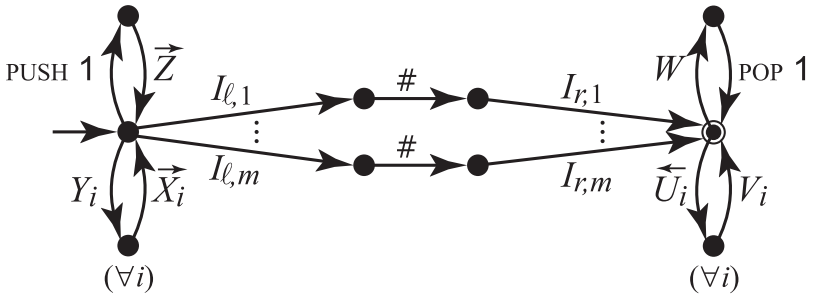


Figure 4: A one-turn counter automaton recognizing computation histories.

Induction step, case $\alpha_n = \beta_n = \varepsilon$, $n > 0$. Let $\eta\#\theta = \overrightarrow{z_n}\eta'\#\theta' w_n$ be reducible to $u\#v'w_n$. Using the same sequence of reductions, $\overrightarrow{z_n}\eta'\#\theta'$ is reduced to $u\#v'$. By Lemma 1, $\eta'\#\theta' \in L_0$ can be reduced to $z_n u\#v'$. Applying the induction hypothesis to the latter reduction, we obtain that $z_n u\#v'$ is derivable in the rewriting system. Then it is sufficient to apply the rule $(z_n, w_n) \in \overset{\ell r}{\to}$ to derive $u\#v'w_n$.

Induction step, case $\beta_n = \beta'_n \overleftarrow{x}\, y$. If $\eta\#\theta = \eta\#\theta'\overleftarrow{x}\, y \in L_0$, where $x \overset{rr}{\to} y$, is reducible to $u\#v'y$, then $\eta\#\theta'\overleftarrow{x}$ is reducible to $u\#v'$, and, by Lemma 1, $\eta\#\theta' \in L_0$ is reducible to $u\#v'x$. Therefore, $u\#v'x$ is derivable by the induction hypothesis, and the rule $x \overset{rr}{\to} y$ can be used to derive $u\#v'y$.

Induction step, case $\alpha_n = y\overrightarrow{x}\alpha'_n$. Proved symmetrically. □

In contrast to Theorem 1, here the language of computation histories is, in general, not regular. However, it can be recognized by a pushdown automaton from a simple subclass.

Since I is regular and $\overset{\ell\ell}{\to}, \overset{rr}{\to}$ are recognizable, they can be represented as $I = \bigcup_i I_{\ell,i}\#I_{r,i}$, $\overset{\ell\ell}{\to} = \bigcup_i X_i \times Y_i$ and $\overset{rr}{\to} = \bigcup_i U_i \times V_i$, where all the languages are regular. Consider a counter automaton $\mathcal{A}$ (that is, a pushdown automaton with the pushdown alphabet $\Gamma = \{1\}$) over the input alphabet $\Sigma_3 \cup \{\#\}$, and with transitions defined according to Figure 4. The arcs labelled with regular languages specify subautomata that simulate DFAs for these languages without modifying the counter.

Lemma 8. *The counter automaton $\mathcal{A}$ defined in Figure 4 generates the language L_0. Additionally, $\mathcal{A}$ makes one turn of the counter in each computation, and this turn takes place exactly over the center marker.*

Proof. By (6), every word $\alpha_n\overrightarrow{z_n}\ldots\alpha_1\overrightarrow{z_1}\alpha_0 u_0\#v_0\beta_0 w_1\beta_1\ldots w_n\beta_n \in L_0$ can be accepted by $\mathcal{A}$ as follows. The left part $\alpha_n\overrightarrow{z_n}\ldots\alpha_1\overrightarrow{z_1}\alpha_0$ is read while in the initial state: every $\alpha_t = y_{i_1}\overrightarrow{x_{i_1}}\ldots y_{i_k}\overrightarrow{x_{i_k}}$ is read by traversing the appropriate paths $Y_{i_j}\overrightarrow{X_{i_j}}$ for every $j \in \{1,\ldots,k\}$, while each $\overrightarrow{z_j}$ is read along with adding 1 to the counter. After this part is read, the value of the counter is n. Since the middle part $u_0\#v_0 \in I$ is in $I_{\ell,s}\#I_{r,s}$ with $1 \leqslant s \leqslant m$ by assumption, it is read along the s-th path from the initial state to the accepting state. The

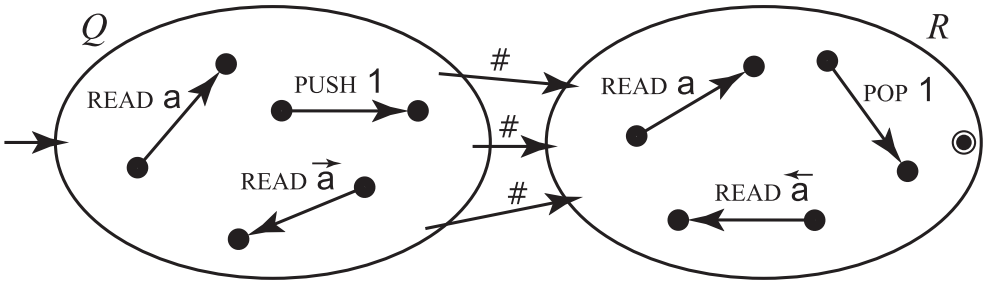


Figure 5: The form of a one-turn counter automaton $\mathcal{A}$ in Lemma 9.

right part $\beta_0 w_1 \beta_1 \dots w_n \beta_n$ is read while in the accepting state, symmetrically to the left part; for every w_j the counter is decremented by 1. Exactly n decrementations are done, and when the input is exhausted, the counter equals zero, which leads to acceptance.

Conversely, if $\mathcal{A}$ accepts some word $\eta\#\theta$, then the computation consists of $n \geqslant 0$ visits to the PUSH 1—$\overrightarrow{Z}$ component, interleaved with any number of visits to any of the Y_i—$\overrightarrow{X_i}$ components, then it follows one of the paths $I_{\ell,s}\#I_{r,s}$, and finally makes n visits to the W—POP 1 component, interleaved with any number of visits to the $\overleftarrow{U_i}$—V_i components. If the number of visits to PUSH 1—$\overrightarrow{Z}$ and to W—POP 1 does not match, the counter will not be equal to zero in the end. This requires the word to be exactly of the form (6). □

The next step is to apply reductions to the language of computation histories (6). It is well-known that such reductions preserve regularity, see the proof of Theorem 1. It turns out that for PDAs of the restricted kind used in Lemma 8, these reductions can also be effectively implemented.

Lemma 9. *Let $\mathcal{A}$ be a one-turn counter automaton over $\Sigma_3 \cup \{\#\}$, such that $L(\mathcal{A}) \subseteq (\Sigma \cup \overrightarrow{\Sigma})^*\#(\Sigma \cup \overleftarrow{\Sigma})^*$, and for every computation of $\mathcal{A}$ on an input $u\#v$, the turn of the counter takes place over the center marker. Then* $\mathrm{red}(L(\mathcal{A}))$ *is recognized by a one-turn counter automaton $\mathcal{B}$, which, given $\mathcal{A}$, can be effectively constructed.*

Proof. Let us assume that transitions of $\mathcal{A}$ are of three forms: by reading a certain input symbol without touching the counter (READ a), by incrementing the counter (PUSH 1) and by decrementing the counter (POP 1). Every accepting computation of $\mathcal{A}$ contains exactly one transition by #, so it can be assumed that $\mathcal{A}$ is of the form given in Figure 5, where the left part contains transitions by symbols in $\Sigma \cup \overrightarrow{\Sigma}$ and increments the counter, while the right part makes transitions by symbols in $\Sigma \cup \overleftarrow{\Sigma}$ and decrements the counter.

It is known that the language of reduced words $\mathrm{red}(L(\mathcal{A}))$ can be represented in terms of an inverse substitution of the set D of words reducible to ε into $L(\mathcal{A})$, see e.g. Benois [2], Pin and Sakarovitch [12] or the authors [7].

The set D is a variant of the Dyck language, defined by a context-free grammar with the following productions: $S \to \overrightarrow{a} S a$, $S \to a S \overleftarrow{a}$, $S \to SS$ and $S \to \varepsilon$. Precisely, we have

$$\begin{aligned}\mathrm{red}(L(\mathcal{A})) = \{u_1 \dots u_m \# v_1 \dots v_n \mid\ & u_1, \dots, u_m, v_1, \dots, v_n \in \Sigma^*, \text{ and} \\ & \text{there exist } x_0, x_1, \dots, x_m, y_0, y_1, \dots, y_n \in D, \text{ such that} \\ & x_0 u_1 x_1 \dots u_m x_m \# y_0 v_1 y_1 \dots v_n y_n \in L(\mathcal{A})\},\end{aligned}$$

and we aim at constructing a new one-turn counter automaton $\mathcal{B}$ over the alphabet $\Sigma \cup \{\#\}$, such that the computation of $\mathcal{B}$ on the input $u_1 \dots u_m \# v_1 \dots v_n$ simulates the computation of $\mathcal{A}$ on the appropriate input $x_0 u_1 x_1 \dots u_m x_m \# y_0 v_1 y_1 \dots v_n y_n$.

In our simulation, the value of $\mathcal{B}$'s counter after reading $u_1 \dots u_m$ shall be equal to the value of $\mathcal{A}$'s counter after reading $x_0 u_1 x_1 \dots u_m x_m$, and the right part shall be handled symmetrically. When the automaton $\mathcal{A}$ reads a block u_i and increments the counter by some value, $\mathcal{B}$ repeats exactly the same actions. On the other hand, when $\mathcal{A}$ reads a block x_i and increments the counter by some value, the automaton $\mathcal{B}$ shall not consume any input symbols, but it shall increment the counter by the same value. To achieve the latter, $\mathcal{B}$ has to be equipped with an additional detour block between every two states, which we now describe.

Let Q be the set of states in the left part of the automaton $\mathcal{A}$, as shown in Figure 5. For every pair of states $q, q' \in Q$, we consider the computations of $\mathcal{A}$ starting from q, ending with q', consuming any word from D and incrementing the counter by k. Let $K_{q,q'}$ be the set of all such numbers k. The key to our proof is the following fact:

Claim 5.1. *For every $q, q' \in Q$, the set $K_{q,q'}$ is ultimately periodic, and an automaton for this set can be algorithmically computed.*

To prove this claim, let us construct a pushdown automaton $\mathcal{C}_{q,q'}$ recognizing this set, with numbers given to it in unary notation. All these automata are obtained from a single base automaton by fixing its initial and accepting states.

This base automaton $\mathcal{C}$ has the input alphabet $\{1\}$, the set of states Q, and its transitions are obtained from $\mathcal{A}$'s transitions by the following relabelling:

- Every time $\mathcal{A}$ reads a symbol $\overrightarrow{a} \in \overrightarrow{\Sigma}$, $\mathcal{C}$ pushes a onto the pushdown.
- Every time $\mathcal{A}$ reads a symbol $a \in \Sigma$, $\mathcal{C}$ pops a from the pushdown.
- Every time $\mathcal{A}$ increments the counter, $\mathcal{C}$ reads 1 from the input.

Recalling that $\mathcal{A}$ and $\mathcal{C}$ have the same states and their transitions differ only in labels, there is a natural one-to-one correspondence between sequences of transitions of $\mathcal{A}$ and of $\mathcal{C}$. Every sequence of transitions of $\mathcal{A}$ within Q

yields a well-formed computation. On the other hand, not every computation of $\mathcal{C}$ is valid, since some transitions lead to popping a symbol which is not at the top of the pushdown. However, as long as $\mathcal{A}$ is reading a word from D, the corresponding transitions of $\mathcal{C}$ form a valid computation, because $\mathcal{C}$ operates its stack in the same way as the standard PDA accepting the Dyck language. This leads to the following correspondence between computations of $\mathcal{C}$ starting and ending with an empty pushdown and computations of $\mathcal{A}$ reading a word from D within Q:

- A computation of $\mathcal{C}$ starting with an empty pushdown ends with an empty pushdown if and only if the corresponding computation of $\mathcal{A}$ reads a word from D (since the PUSH and POP in $\mathcal{C}$ simulate the well-known recognizer for the Dyck language).

- A computation of $\mathcal{C}$ reads 1^k if and only if the corresponding computation of $\mathcal{A}$ increments the counter by k.

Claim 5.1.1. *For every $q, q' \in Q$, and $k \geqslant 0$, there is a computation of the automaton $\mathcal{C}$ starting from $(q, 1^k, \varepsilon)$ and ending with $(q', \varepsilon, \varepsilon)$ if and only if there exist $w \in D$ and a computation of the automaton $\mathcal{A}$ starting from (q, w, ε) and ending with $(q', \varepsilon, 1^k)$.*

If $\mathcal{C}$ goes from $(q, 1^k, \varepsilon)$ to $(q', \varepsilon, \varepsilon)$, then the corresponding computation of $\mathcal{A}$ goes from (q, w, ε) to $(q', \varepsilon, 1^\ell)$ for certain w and ℓ; by the above properties, $w \in D$ and $\ell = k$. Conversely, when $\mathcal{A}$ goes from (q, w, ε) to $(q', \varepsilon, 1^k)$, for some $w \in D$ and $k \geqslant 0$, the corresponding computation of $\mathcal{C}$ goes from $(q, 1^\ell, \varepsilon)$ to (q', ε, x), for some $\ell \geqslant 0$ and for some pushdown word x. By the above properties, $x = \varepsilon$ and $\ell = k$, and Claim 5.1.1 is proved.

For every $q, q' \in Q$, define an automaton $\mathcal{C}_{q,q'}$ as $\mathcal{C}$ with the initial state q and a unique accepting state q'. The following statement immediately follows from Claim 5.1.1:

Claim 5.1.2. *For every $q, q' \in Q$, and $k \geqslant 0$, $\mathcal{C}_{q,q'}$ accepts 1^k if and only if $k \in K_{q,q'}$;*

The latter claim establishes that $K_{q,q'}$ is recognized by a certain pushdown automaton, and hence is context-free. Then, as every context-free language over a unary alphabet, it must be regular, which proves Claim 5.1.

Let us apply the same procedure to the right part of the automaton $\mathcal{A}$, see Figure 5. Let R be the set of states in that part, and for every pair of states $r, r' \in R$ we consider those computations of $\mathcal{A}$ that start from r, end in r', consume any word from D and decrement the counter by k. Let $K_{r,r'}$ be the set of all such numbers k. Then the following statement is proved by exactly the same method as Claim 5.1:

Claim 5.2. *For every $r, r' \in R$, the set $K_{r,r'}$ is ultimately periodic, and an automaton for this set can be constructed algorithmically.*

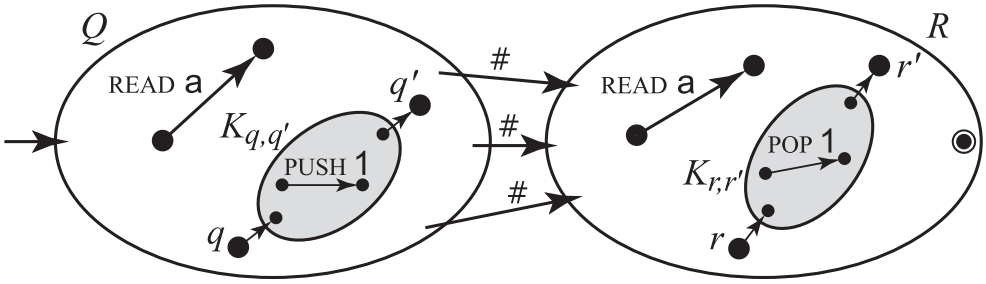


Figure 6: Transitions and detour blocks in the counter automaton $\mathcal{B}$.

Using the regularity of these sets, we can define the new counter automaton $\mathcal{B}$ as follows. The automaton $\mathcal{B}$ has the same states as $\mathcal{A}$, plus new states needed to simulate each $K_{q,q'}$ and $K_{r,r'}$. The initial and final states of $\mathcal{B}$ are the same as in $\mathcal{A}$. It has the following transitions:

- When $\mathcal{A}$ has a transition by $a \in \Sigma$ or # between any two of its states, $\mathcal{B}$ has the same transition between the same states.
- For every pair of states $q, q' \in Q$, $\mathcal{B}$ has an ε-transition from q to a subautomaton defined as follows. Let the DFA for $K_{q,q'}$ have the set of states $P_{q,q'}$ and the transition function $\delta_{q,q'}$. Then $\mathcal{B}$ has all the states from $P_{q,q'}$, its transition from q goes to the initial state in this subautomaton, and for every $p \in P_{q,q'}$ the automaton $\mathcal{B}$ has a transition from p to $\delta_{q,q'}(p, 1)$ labelled "PUSH 1". For every final state p_F in this subautomaton, $\mathcal{B}$ has an ε-transition from p_F to q'.
- Similarly, for every pair of states $r, r' \in R$, $\mathcal{B}$ contains a subautomaton implementing $K_{r,r'}$, in which every transition is labelled "POP 1".

The general form of $\mathcal{B}$ is illustrated in Figure 6.

Let us now prove that $\mathcal{B}$ recognizes the language $\mathrm{red}(L(\mathcal{A}))$. First, we shall see that $\mathcal{B}$ correctly simulates $\mathcal{A}$ on the right parts of words.

Claim 5.3. *The automaton $\mathcal{B}$ accepts a word v from state $r \in R$ with pushdown contents 1^k if and only if there exists $y \in \Sigma_3^*$, such that $v = \mathrm{red}(y)$ and $\mathcal{A}$ accepts y from state r with pushdown contents 1^k.*

$\Leftarrow$ Induction on the length of an accepting computation of $\mathcal{B}$.

Basis. If $(r, v, 1^k)$ is an accepting configuration of $\mathcal{B}$ then $v = \varepsilon$, $k = 0$ and taking $y = \varepsilon$ we obtain that $(r, y, 1^k)$ is an accepting configuration of $\mathcal{A}$.

Induction step, first step READ a**.** Suppose $\mathcal{B}$ starts its computation by reading an input symbol $a \in \Sigma$. Then $\mathcal{B}$ goes from $(r, av', 1^k)$ to $(r', v', 1^k)$ and from there to acceptance. By the induction hypothesis, there is y', such that $v' = \mathrm{red}(y')$ and $\mathcal{A}$ accepts from $(r', y', 1^k)$. Taking $y = ay'$, we obtain the following computation:

$$(r, ay', 1^k) \overset{\mathcal{A}}{\vdash} (r', y', 1^k) \overset{\mathcal{A}^*}{\vdash} \mathrm{Acc},$$

and this case is proved.

Induction step, starting from a detour through $K_{r,r'}$. Let the first step done by $\mathcal{B}$ in this computation be an ε-transition to one of the detour blocks. Thus $\mathcal{B}$ makes a transition from $(r, v, 1^k)$ to $(r', v, 1^\ell)$, for certain $r' \in R$ and $k-\ell \in K_{r,r'}$, and the latter configuration leads to acceptance. By the induction hypothesis, there exists y, such that $v \in \mathrm{red}(y)$ and $(r', y, 1^\ell)$ leads $\mathcal{A}$ to acceptance. On the other hand, by definition, $k - \ell \in K_{r,r'}$ means that there exists a word $z \in D$, such that $\mathcal{A}$ can go from $(r, zy, 1^k)$ to $(r', y, 1^\ell)$. Since $\mathrm{red}(zy) = \mathrm{red}(y) = v$, the given computation meets the requirements.

$\Leftarrow$ Let $\mathcal{A}$ accept starting from $(r, y, 1^k)$ and let $y = y_0 v_1 y_1 \dots v_n y_n$, with $y_i \in D$, $v_i \in \Sigma^+$. We need to prove that $\mathcal{B}$ accepts starting from $(r, v_1 \dots v_n, 1^k)$. The argument proceeds by induction on the length of a computation of $\mathcal{A}$.

Basis. If $\mathcal{A}$'s configuration $(r, y, 1^k)$ is accepting, then $n = 0$, $y = \varepsilon$, $v_1 \dots v_n = \varepsilon$ and $k = 0$. Consequently, $(r, v_1 \dots v_n, 1^k)$ is an accepting configuration of $\mathcal{B}$.

Induction step, case of reading from v_1 at the first step. Suppose $y_0 = \varepsilon$ and the first step in the accepting computation of $\mathcal{A}$ is $(r, a v_1' y_1 \dots v_n y_n, 1^k)$ to $(r, v_1' y_1 \dots v_n y_n, 1^k)$, from where $\mathcal{A}$ proceeds to acceptance. By the induction hypothesis, $\mathcal{B}$ accepts from $(r, v_1' v_2 \dots v_n, 1^k)$. Since $\mathcal{B}$ can go from $(r, a v_1' v_2 \dots v_n, 1^k)$ to $(r, v_1' v_2 \dots v_n, 1^k)$, the required accepting computation has been constructed.

Induction step, case of the first step not touching v_1. If v_1 is not touched at the first step of the computation, consider the last configuration in the computation of $\mathcal{A}$ before it starts reading v_1. Let it be $(r', v_1 y_1 \dots v_n y_n, 1^\ell)$. Reaching it requires at least one step, and hence the induction hypothesis can be applied to show that $\mathcal{B}$ accepts from $(r', v_1 \dots v_n, 1^\ell)$. While moving from configuration $(r, y_0 v_1 y_1 \dots v_n y_n, 1^k)$ to configuration $(r', v_1 y_1 \dots v_n y_n, 1^\ell)$, $\mathcal{A}$ has read $y_0 \in D$, so, by the definition of $K_{r,r'}$, $k - \ell \in K_{r,r'}$. Therefore, $\mathcal{B}$, having started in $(r, v_1 \dots v_n, 1^k)$, can proceed through the $K_{r,r'}$ detour, popping $1^{k-\ell}$ on the way, and thus reaching $(r', v_1 \dots v_n, 1^\ell)$, from where it can accept. This completes the proof of Claim 5.3.

The behaviour of $\mathcal{B}$ on left parts of words is stated in the following claim:

Claim 5.4. *$\mathcal{B}$ accepts a word $u\#v$ from state $q \in Q$ with pushdown contents 1^k if and only if there exists $x\#y \in \Sigma_3^*\#\Sigma_3^*$, such that $u\#v = \mathrm{red}(x\#y)$ and $\mathcal{A}$ accepts $x\#y$ from state q with pushdown contents 1^k.*

Claim 5.4 is proved in the same way as Claim 5.3, with the only difference that the basis of induction for Claim 5.4 is the statement of Claim 5.3.

Substituting the initial state of A for q and letting $k = 0$ in Claim 5.4, we obtain that $u\#v \in L(\mathcal{B})$ if and only if there exists $x\#y \in L(\mathcal{A})$, such that $u\#v = \mathrm{red}(x\#y)$, which completes the proof of the lemma. $\square$

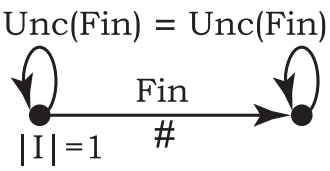
On the basis of these three lemmata, Theorem 5 can be proved along the same lines as Theorem 1, though in a context of nonregularity.

Proof of Theorem 5. Let $L \subseteq \Sigma^*\#\Sigma^*$ be the language generated by the rewriting system. According to Lemma 7, $L = \mathrm{red}(L_0)$. By Lemma 8, the language L_0 is recognized by a one-turn counter automaton. Due to Lemma 9, this implies that $\mathrm{red}(L_0)$ is recognized by a one-turn counter automaton as well. □

5.2 The case of controlled $\overset{\ell r}{\to}$

As we have seen, having an uncontrolled relation $\overset{\ell r}{\to}$ limits the expressive power of RS-rewriting to a subset of the context-free languages. Now let $\overset{\ell r}{\to}$ be a controlled finite relation. We shall see that even if $\overset{\ell\ell}{\to}$ and $\overset{rr}{\to}$ are uncontrolled, the resulting system is still computationally universal.

Theorem 6. *For every recursively enumerable language $L \subseteq \Sigma^*$ there exists a one-way RS-rewriting system formed by relations $\overset{\ell r}{\to} \in Fin$ and $\overset{\ell\ell}{\to} = \overset{rr}{\to} \in Unc(Fin)$ and a singleton initial set I, which generates the language $L\#$ modulo $\Sigma^*\#$. Given a type 0 grammar for L, such a system can be effectively constructed.*



Let $G = (\Sigma, N, P, S)$ be a type 0 grammar for L, let $V = \Sigma \cup N$. Consider a new alphabet $V \cup \overline{V}$, where $\overline{V} = \{\, \overline{s} \mid s \in V \,\}$. Let $\overline{s_1 \dots s_n} = \overline{s}_1 \dots \overline{s}_n$ for all $n \geqslant 0$, $s_i \in V$. Define the RS-rewriting system as follows:

$$
\begin{aligned}
I &= \{S\#\} \\
\overset{\ell\ell}{\to} = \overset{rr}{\to} &= X \times X, \quad \text{where } X = \{\, \overline{a}a \mid a \in V \,\} \cup \{\varepsilon\} \\
\overset{\ell r}{\to} &= \{\, (a, \overline{a}), (\overline{a}, a) \mid a \in V \,\} \cup \{\, (u, \overline{v}) \mid u \to v \in P \,\}
\end{aligned}
$$

The intended behaviour of the system is to simulate G using words of a particular form, which have only symbols without bars in the left stack and only symbols with bars in the right stack. Symbols with inappropriate bars may appear for a moment on either side during a communication of the stacks, but they are supposed to be immediately removed using a rule of the form $\overline{a}a \to \varepsilon \in \overset{\ell\ell}{\to} = \overset{rr}{\to}$.

A word of the above form $\alpha\#\overline{\beta}$, with $\alpha, \beta \in V^*$, represents the sentential form $\beta\alpha$ of the grammar. Productions are encoded in the relation $\overset{\ell r}{\to}$, that is, the factor being rewritten is always prepared on the top of the left stack, and always migrates to the right stack during a rewriting. In order to do rewriting in arbitrary parts of a sentential form, it is necessary to scroll it in both directions. A symbol can migrate from the left stack to the right stack

by a direct use of the relation $\xrightarrow{\ell r}$. However, even though $\xrightarrow{r\ell} = \varnothing$, that is, no messages may be directly sent from the right stack to the left stack, such a reverse communication channel can be implemented as well.

Claim 6.1 (Moving symbols)**.** *Let $a \in V$ and $u, v \in V^*$. Then*

(“$\rightarrow$”) *$au\#\overline{v}$ derives $u\#\overline{va}$;*

(“$\leftarrow$”) *$u\#\overline{va}$ derives $au\#\overline{v}$.*

Proof. Part (“$\rightarrow$”) follows by a single application of $\xrightarrow{\ell r}$. Part (“$\leftarrow$”) defines a reverse communication channel using the following three-step protocol:

$$u\#\overline{va} \xRightarrow{\ell\ell} \overline{a}au\#\overline{va} \xRightarrow{\ell r} au\#\overline{va}a \xRightarrow{rr} au\#\overline{v}$$

First a symbol $a \in V$ is guessed at the left and a pair $\overline{a}a$ is created. Then one of these symbols is transferred to the right and cancelled there with the existing a. Thus a has effectively been moved from the right to the left. □

As we shall soon see, if a wrong symbol is guessed in the above sequence, or the protocol is violated in any other way, then a word of the form $w\#$ can no longer be derived.

Claim 6.2. *If $w \in V^*$ is generated by G from S, then $w\#$ is derivable in the constructed rewriting system.*

Proof. Induction on the length of the generation of w.

Basis. S is generated by G in 0 steps, and $S\# \in I$.

Induction step. Let $xuy \xRightarrow{G} xvy$ by a production $u \to v$. By the induction hypothesis, a word $xuy\#$ is derivable in the rewriting system. Using Claim 6.1(“$\rightarrow$”) $|x|$ times we can derive $uy\#\overline{x}$. Since $u \to v \in P$, $u \xrightarrow{\ell r} \overline{v}$ by construction, and we obtain $y\#\overline{xv}$. Finally, by Claim 6.1(“$\leftarrow$”) $|xv|$ times, the word $xvy\#$ is derivable, which proves Claim 6.2. □

Let us show that only words derivable in G can be produced by our rewriting. Denote by $\mathrm{d}(x)$ the word obtained from $x \in (V \cup \overline{V})^*$ by deleting all occurrences of factors of the form $\overline{a}a$ for $a \in V$. In the following we adopt the convention that $\overline{\overline{x}} = x$ for any word $x \in V^*$.

Claim 6.3. *For every word $\alpha = y\#x$ derivable in the rewriting system, from which a word belonging to $V^*\#$ can be derived, the word $\overline{\mathrm{d}(x)}\,\mathrm{d}(y)$ can be generated in G.*

Proof. We proceed by induction on the length of the derivation. The initial word $S\#$ clearly satisfies the claim. To prove the induction step, take any word $\alpha = y\#x$, where $x, y \in (V \cup \overline{V})^*$ are such that $\overline{\mathrm{d}(x)}\,\mathrm{d}(y)$ is derivable in G. Let $\beta = y'\#x'$ be a word obtained from α by one step of the rewriting

and assume that some word from $V^*\#$ is derivable from β. If a rule from $\overset{\ell\ell}{\to}$ or from $\overset{rr}{\to}$ was applied, then $\mathrm{d}(x') = \mathrm{d}(x)$ and $\mathrm{d}(y') = \mathrm{d}(y)$, and so β satisfies the claim.

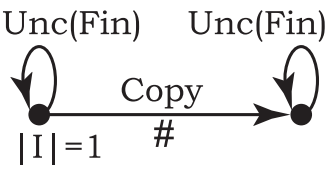
If β was produced by applying a rule $u \overset{\ell r}{\to} \overline{v}$, where either $u \to v \in P$ or $u = v \in V$, then $x' = x\overline{v}$ and $y = uy'$. This implies that $\mathrm{d}(x') = \mathrm{d}(x)\overline{v}$ and $\mathrm{d}(y) = u\,\mathrm{d}(y')$ hold. Therefore we have $\overline{\mathrm{d}(x')}\,\mathrm{d}(y') = \overline{\mathrm{d}(x)}v\,\mathrm{d}(y')$ and $\overline{\mathrm{d}(x)}\,\mathrm{d}(y) = \overline{\mathrm{d}(x)}u\,\mathrm{d}(y')$, which shows that $\overline{\mathrm{d}(x')}\,\mathrm{d}(y')$ can be derived in G.

Finally, assume that the rule applied is of the form $\overline{a} \overset{\ell r}{\to} a$ for $a \in V$, in other words, we have $x' = xa$ and $y = \overline{a}y'$. Because $\mathrm{d}(y)$ contains no letters from $\overline{V}$, the initial letter of y' has to be a, so $y' = az$ for some $z \in (V \cup \overline{V})^*$ satisfying $\mathrm{d}(z) = \mathrm{d}(y)$. By our assumption on β, the final occurrence of a in x' can be eventually removed by the rewriting. Because this can be achieved only using rules of $\overset{rr}{\to}$, the letter preceding a in x' must be $\overline{a}$, which shows that $x = w\overline{a}$ for some $w \in (V \cup \overline{V})^*$ such that $\mathrm{d}(w) = \mathrm{d}(x')$. Altogether, we obtain $\overline{\mathrm{d}(x')}\,\mathrm{d}(y') = \overline{\mathrm{d}(w)}a\,\mathrm{d}(z) = \overline{\mathrm{d}(w)\overline{a}}\,\mathrm{d}(z) = \overline{\mathrm{d}(x)}\,\mathrm{d}(y)$ as required. The claim is proved. □

Returning to the proof of Theorem 6, for every word $w\# \in \Sigma^*\#$ derivable in the rewriting system, the word w can be generated using G by Claim 6.3. Conversely, if $w \in \Sigma^*$ is generated by G, then, by Claim 6.2, $w\#$ is derivable in the constructed rewriting system. Q. E. D.

The relation $\overset{\ell r}{\to}$ in the above construction is almost a copy relation. If the bars in the right sides of words are inverted, $\overset{\ell r}{\to}$ becomes a copy relation, while the relations $\overset{\ell\ell}{\to}$ and $\overset{rr}{\to}$ become different. This leads to the following variant of Theorem 6:

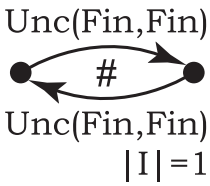
Proposition 1. *For every recursively enumerable language $L \subseteq \Sigma^*$ there exists a one-way RS-rewriting system formed by relations $\overset{\ell r}{\to} \in Copy$ and $\overset{\ell\ell}{\to}, \overset{rr}{\to} \in Unc(Fin)$ and a singleton initial set I, which generates the language $L\#$ modulo $\Sigma^*\#$. Given a type 0 grammar for L, such a system can be effectively constructed.*



6 Two-way communication

The rewriting systems considered so far allowed one-way communication only, with messages sent from the left stack to the right stack. Let us now consider two-way rewriting, in which both relations $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$ are used. It will be shown that this rewriting is computationally universal already in its weakest, least controlled form.

Theorem 7. *For every recursively enumerable language $L \subseteq \Sigma^+$ there exists an alphabet $\Gamma \supseteq \Sigma$, finite uncontrolled relations $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$ and a word $w_0 \in \Gamma^*\#\Gamma^*$, such that the language generated by the two-way rewriting system from w_0 equals $\lambda(L)\#$ modulo intersection with $\lambda(\Sigma^+)\#$, for a suitable injective morphism λ.*



The general idea behind our construction is clear. The two stacks contain a sentential form of a Chomsky type 0 grammar, which is redistributed between the stacks before every rewriting step. The symbols and productions are communicated between the stacks in unary notation: in order to send an object number n, $f(n)$ empty messages are sent. The set of uncontrolled rules is constructed in such a way that both parties must faithfully follow a certain rigid protocol, and if they ever divert from it, they will never be able to get back on the right track.

This sounds easy in theory, but if one recalls that the stacks do not even have local states, the existence of such a rewriting system will appear highly unlikely. However, there exists quite a sophisticated solution.

Let $G = (\Sigma, N, P, S)$ be a type 0 grammar generating L, let $V = \Sigma \cup N$ be its full alphabet and assume that every production in P is of the form $u \to v$ with $u, v \in V^+$. We consider the following extended alphabet:

$$\Gamma = V \cup \{\pounds, \$, ¢, a, b, c, d, e, g, \bar{g}, \tilde{g}, h, \hbar\}.$$

Let us denote $n = |V \cup P| + 3$ and fix an arbitrary bijection $\varphi\colon V \cup P \to \{3, \ldots, n-1\}$. We define morphisms $\lambda, \rho\colon V^+ \to \Gamma^+$ by setting $\lambda(A) = h\hbar A¢^n c$ and $\rho(A) = c¢^n A\hbar h$ for every $A \in V$. The morphism λ will be used to encode symbols from V in the left stack, while ρ symmetrically encodes symbols in the right stack.

Let $\overset{\ell r}{\to} = L_- \times R_+$ and $\overset{r\ell}{\to} = R_- \times L_+$, where

$$\begin{aligned}
L_- &= \{\$, ¢^n ch\hbar, g\bar{g}, \tilde{g}\pounds, \pounds^2 ab\} \cup \{\, A¢^{\varphi(A)}, h\hbar A¢^{\varphi(A)} \mid A \in V \,\} \\
&\quad \cup \{\varepsilon, h\hbar\}^{-1}\{\, \lambda(u)(¢^{n-\varphi(u\to v)}c)^{-1} \mid u \to v \in P \,\}, \\
R_+ &= \{\pounds, ¢, g, h, a\pounds^2\tilde{g}\bar{g}\$, h\hbar b, c¢^n de\hbar\$\} \cup \{\, c¢^n A\hbar\$^{\varphi(A)} \mid A \in V \,\} \\
&\quad \cup \{\, (\rho(v)h^{-1})\$^{\varphi(u\to v)} \mid u \to v \in P \,\}, \\
R_- &= \{\$, \hbar hc¢^n, \bar{g}g, \pounds\tilde{g}, ba\pounds^2, e\hbar h, ¢^2 d\} \cup \{\, ¢^{\varphi(A)}A, ¢^{\varphi(A)}A\hbar h \mid A \in V \,\}, \\
L_+ &= \{\pounds, ¢, g, h, \$\bar{g}\tilde{g}\pounds^2 a, b\hbar h, \$^2\hbar S¢^n c\} \cup \{\, \$^{\varphi(A)}\hbar A¢^n c \mid A \in V \,\}.
\end{aligned}$$

As before, the relations of left-to-right and right-to-left one-step derivation induced by these rules are denoted by $\overset{\ell r}{\Longrightarrow}$ and $\overset{r\ell}{\Longrightarrow}$, respectively, and $\Longrightarrow = \overset{\ell r}{\Longrightarrow} \cup \overset{r\ell}{\Longrightarrow}$. The initial set is defined as $I = \{g\bar{g}\tilde{g}\pounds^2 ab\hbar\#\hbar h\hbar\}$.

A sentential form $z \in V^+$ of the grammar will be distributed between two stacks as a word $\lambda(y)(h\hbar)^m\#(\hbar h)^k\rho(x)$, for various factorizations $z = xy$

and for various $k, m \geqslant 0$. The rewriting will redistribute the sentential form, alternately shifting it symbol by symbol from the left stack to the right stack until it is entirely in the right stack, then shifting it back from the right stack to the left stack until no symbols of the sentential form are left in the right stack, and so forth. During the left-to-right movement, the rewriting may apply productions $u \to v \in P$: in this case the word encoding u is removed from the left stack and $\rho(v)$ is added to the top of the right stack. Simply moving a symbol $A \in V$ from one stack to the other can thus be regarded as applying a production $A \to A$. During the right-to-left movement, symbols are only moved verbatim, that is, the whole sentential form is returned back to the left.

Let us now briefly describe how our rewriting rules implement correct shifting of parts of a sentential form between the stacks. Each basic piece of a sentential form to be moved from left to right is represented by two words in L_- and a single corresponding word in R_+. For each symbol $A \in V$ the set L_- contains two words $A¢^{\varphi(A)}$ and $h\hbar A¢^{\varphi(A)}$ used for moving A to the other side; the corresponding word in R_+ is $c¢^n A\hbar\$^{\varphi(A)}$. Similarly, for each production $u \to v \in P$ there are two words $\{\varepsilon, h\hbar\}^{-1}\lambda(u)(¢^{n-\varphi(u\to v)}c)^{-1}$ in L_- representing its left-hand side, and the corresponding word $(\rho(v)h^{-1})\$^{\varphi(u\to v)} \in R_+$ encodes its right-hand side. It is intended that whenever one of these words from L_- is removed from the left, the corresponding word in R_+ must be appended to the right.

This correspondence is ensured by keeping a certain number of ¢s on the left, which encode the word from L_- that has been deleted there, and by adding a number of \$s on the right as a part of the appended word for subsequent verification. The resulting blocks of ¢s and \$s encode the number of the applied production or the number of the moved symbol in unary notation. This information can be communicated between the stacks by sending \$s to the left, changing them for ¢s on the way. If the number of new ¢s matches the number of ¢s left in the left stack by the rewriting, all these ¢s can be deleted together, and this information is communicated to the right by appending a letter h. If all \$s were previously sent to the left, this letter h adjoins the letter $\hbar$ originating from the appended word of R_+ where it occurs right before the block of \$s. This certifies that the word removed from the left and the word added to the right correspond to the same symbol or production of the grammar. The newly assembled pair $\hbar h$ allows to start shifting another piece of a sentential form.

Symmetrically, the sets R_- and L_+ contain words corresponding to symbols to be moved from right to left. The only difference is that productions are applied only when moving material from left to right. Apart from this, productions of the grammar and rules for moving symbols between the two stacks are encoded in the very same way.

To prove the theorem, it is sufficient to verify the following equivalence:

Main Claim. *A word $w \in V^*$ is a sentential form of G if and only if the*

word $\lambda(w)\#$ is derivable in our rewriting system.

To prove that every word $\lambda(w)\#$, with $w \in V^*$ generated by G, can be derived, we simulate each application of a production $u \to v$ of G to a sentential form w by rewriting the corresponding word $\lambda(w)\#$. First, we shift the entire sentential form w from the left stack to the right one, modifying the factor $\lambda(u)$ to $\rho(v)$ on the way, and then we move it back to the left to get the word $\lambda(w')\#$, where w' is the resulting sentential form.

Each time the sentential form is entirely shifted to one of the stacks and the direction of shifting is reversed, one pair $h\hbar$ or $\hbar h$ is consumed at the bottom of the emptied stack. Therefore, before starting the actual simulation we have to generate a sufficient amount of these pairs at the bottom of each stack. This production of such pairs, which we call "fuel", is performed quite similarly to moving parts of a sentential form between stacks; the letters g and $\bar{g}$ play the role of h and $\hbar$, and £s are used instead of ¢s.

Claim 7.1 (Fuel generation)**.** *For every $m \in \mathbb{N}$, the word $\lambda(S)(h\hbar)^m\#(\hbar h)^m$ can be derived in the rewriting system.*

Proof. Let us first show by induction on m how to derive the word

$$g\bar{g}\tilde{g}£^2ab\hbar(h\hbar)^{m-1}\#(\hbar h)^m\hbar$$

for every $m \in \mathbb{N}$. For $m = 1$ this is just the initial word. Assume we have already derived $g\bar{g}\tilde{g}£^2ab\hbar(h\hbar)^{m-1}\#(\hbar h)^m\hbar$. Then we can add another pair $h\hbar$ and $\hbar h$ at each side by the following derivation:

$$\begin{aligned}
&\underline{g\bar{g}}\,\underline{\tilde{g}£}\,£ab\hbar(h\hbar)^{m-1}\#(\hbar h)^m\hbar \\
&\overset{\ell r}{\Longrightarrow}{}^2 \quad £ab\hbar(h\hbar)^{m-1}\#(\hbar h)^m\hbar(h\hbar b)(a£^2\tilde{g}\bar{g}\,\underline{\$}) \\
&\overset{r\ell}{\Longrightarrow} \underline{(£)£ab}\,\hbar(h\hbar)^{m-1}\#(\hbar h)^{m+1}\hbar ba£^2\tilde{g}\bar{g} \\
&\overset{\ell r}{\Longrightarrow} \hbar(h\hbar)^{m-1}\#(\hbar h)^{m+1}\hbar ba£\underline{£\tilde{g}}\,\underline{\bar{g}(g)} \\
&\overset{r\ell}{\Longrightarrow}{}^2 \quad (\underline{\$}\,\bar{g}\tilde{g}£^2a)(b\hbar h)\hbar(h\hbar)^{m-1}\#(\hbar h)^{m+1}\hbar ba£ \\
&\overset{\ell r}{\Longrightarrow} \bar{g}\tilde{g}£^2ab\hbar(h\hbar)^m\#(\hbar h)^{m+1}\hbar\,\underline{ba£(£)} \\
&\overset{r\ell}{\Longrightarrow} (g)\bar{g}\tilde{g}£^2ab\hbar(h\hbar)^m\#(\hbar h)^{m+1}\hbar
\end{aligned}$$

Once a sufficient number of pairs has been produced, we can derive

$\lambda(S)(h\hbar)^m\#(\hbar h)^m$ as follows:

$$\begin{aligned}
&\underline{g\bar{g}}\,\underline{\tilde{g}\pounds}\pounds ab\hbar(h\hbar)^{m-1}\#(\hbar h)^m\hbar\\
\overset{\ell r}{\Longrightarrow}{}^2\quad &\pounds ab\hbar(h\hbar)^{m-1}\#(\hbar h)^m\hbar(h)(c\text{¢}^n de\hbar\,\underline{\$})\\
\overset{r\ell}{\Longrightarrow}\ &\underline{(\pounds)\pounds ab}\,\hbar(h\hbar)^{m-1}\#(\hbar h)^{m+1}c\text{¢}^n de\hbar\\
\overset{\ell r}{\Longrightarrow}\ &\hbar(h\hbar)^{m-1}\#(\hbar h)^{m+1}c\text{¢}^{n-2}\underline{\text{¢}^2 d}\,\underline{e\hbar(h)}\\
\overset{r\ell}{\Longrightarrow}{}^2\quad &(\underline{\$^2}\,\hbar S\text{¢}^n c)(h)\hbar(h\hbar)^{m-1}\#(\hbar h)^{m+1}c\text{¢}^{n-2}\\
\overset{\ell r}{\Longrightarrow}{}^2\quad &\hbar S\text{¢}^n c(h\hbar)^m\#(\hbar h)^m\underline{\hbar hc\text{¢}^{n-2}(\text{¢}^2)}\\
\overset{r\ell}{\Longrightarrow}\ &(h)\hbar S\text{¢}^n c(h\hbar)^m\#(\hbar h)^m\\
&=\lambda(S)(h\hbar)^m\#(\hbar h)^m
\end{aligned}$$

□

This provides the initial words for simulating the generation of sentential forms in G. The next two claims represent elementary manipulations of the encoded sentential form by the rewriting system.

Claim 7.2 (Moving a symbol)**.** *Let $x, y \in V^*$ and $A \in V$. Then for every $m \in \mathbb{N}$:*

$$\begin{aligned}
\lambda(Ay)(h\hbar)^m\#(\hbar h)^m &\Longrightarrow^* (h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(A),\\
(h\hbar)^{-1}\lambda(Ay)(h\hbar)^m\#(\hbar h)^m\rho(x) &\Longrightarrow^* (h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(xA).
\end{aligned}$$

Similarly in the other direction, for every $m \in \mathbb{N}_0$:

$$\begin{aligned}
(h\hbar)^m\#(\hbar h)^{m+1}\rho(xA) &\Longrightarrow^* \lambda(A)(h\hbar)^m\#(\hbar h)^{m+1}\rho(x)(\hbar h)^{-1},\\
\lambda(y)(h\hbar)^m\#(\hbar h)^{m+1}\rho(xA)(\hbar h)^{-1} &\Longrightarrow^* \lambda(Ay)(h\hbar)^m\#(\hbar h)^{m+1}\rho(x)(\hbar h)^{-1}.
\end{aligned}$$

Proof. In the first part of the claim, the initial words start with $h\hbar A\text{¢}^n c$ and $A\text{¢}^n c$, respectively, and since L_- contains two words for dealing with A, we can set $x = \varepsilon$ in the first case and treat both cases uniformly:

$$\begin{aligned}
&\underline{\{\varepsilon, h\hbar\}A\text{¢}^{\varphi(A)}}\text{¢}^{n-\varphi(A)}c\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)\\
&\qquad\overset{\ell r}{\Longrightarrow}\ \text{¢}^{n-\varphi(A)}c\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)(c\text{¢}^n A\hbar\,\underline{\$^{\varphi(A)}})\\
\overset{r\ell}{\Longrightarrow}{}^{\varphi(A)}\ &\underline{(\text{¢}^{\varphi(A)})\text{¢}^{n-\varphi(A)}ch\hbar}(h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)c\text{¢}^n A\hbar\\
&\qquad\overset{\ell r}{\Longrightarrow}\ (h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)c\text{¢}^n A\hbar(h)\\
&\qquad=(h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(xA)
\end{aligned}$$

The second part of the claim is proved symmetrically. □

Claim 7.3 (Applying a production)**.** *For every $x, y \in V^*$, $m \in \mathbb{N}$ and $u \to v \in P$,*

$$\begin{aligned}
\lambda(uy)(h\hbar)^m\#(\hbar h)^m &\Longrightarrow^* (h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(v),\\
(h\hbar)^{-1}\lambda(uy)(h\hbar)^m\#(\hbar h)^m\rho(x) &\Longrightarrow^* (h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(xv).
\end{aligned}$$

Proof. We prove this claim similarly to the previous one, again setting $x = \varepsilon$ in the first case:

$$\begin{aligned}
&\{\varepsilon, h\hbar\}^{-1}\lambda(u)\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)\\
=\ &\underline{\{\varepsilon, h\hbar\}^{-1}\lambda(u)(¢^{n-\varphi(u\to v)}c)^{-1}}¢^{n-\varphi(u\to v)}c\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)\\
&\overset{\ell r}{\Longrightarrow}\ ¢^{n-\varphi(u\to v)}c\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(x)(\rho(v)h^{-1}\underline{\$^{\varphi(u\to v)}})\\
\overset{r\ell}{\Longrightarrow}{}^{\varphi(u\to v)}\ &\underline{(¢^{\varphi(u\to v)})¢^{n-\varphi(u\to v)}ch\hbar}(h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(xv)h^{-1}\\
&\overset{\ell r}{\Longrightarrow}\ (h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(xv)h^{-1}(h)\\
&=(h\hbar)^{-1}\lambda(y)(h\hbar)^m\#(\hbar h)^m\rho(xv)
\end{aligned}$$

□

Using these elementary operations on sentential forms, the generation of a word in G can be simulated by the rewriting system as follows:

Claim 7.4 (Simulating generation)**.** *For every $m \in \mathbb{N}_0$ and every word $w \in V^*$ generated by G, the word $\lambda(w)(h\hbar)^m\#(\hbar h)^m$ can be derived in the rewriting system.*

Proof. We proceed by induction on the length of the derivation of w. For the initial symbol S this was already proved in Claim 7.1 except for the case $m = 0$, which can be obtained from the case of $m = 1$ using Claim 7.2 twice:

$$\lambda(S)h\hbar\#\hbar h \Longrightarrow^* \#\hbar h\rho(S) \Longrightarrow^* \lambda(S)\#$$

Let $m \in \mathbb{N}_0$, $u \to v \in P$ and $x, y \in V^*$, and consider the generation step of G in which xuy generates xvy. By the induction hypothesis we know that the word $\lambda(xuy)(h\hbar)^{m+1}\#(\hbar h)^{m+1}$ is derivable in our rewriting system. Then if $x \neq \varepsilon$ we use $|x|$ times Claim 7.2 to derive the word $(h\hbar)^{-1}\lambda(uy)(h\hbar)^{m+1}\#(\hbar h)^{m+1}\rho(x)$. This means that for any x we can derive $(h\hbar)^{-1}\lambda(y)(h\hbar)^{m+1}\#(\hbar h)^{m+1}\rho(xv)$ by Claim 7.3. Then using $|y|$ times Claim 7.2 we get $(h\hbar)^m\#(\hbar h)^{m+1}\rho(xvy)$. Thus a unit of fuel has been consumed at the left stack, and the rewriting proceeds by moving all symbols from right to left. This is achieved by applying the second part of Claim 7.2 $|xvy|$ times to derive $\lambda(xvy)(h\hbar)^m\#(\hbar h)^m$. In the end, a unit of fuel is consumed at the right stack, and the rewriting system is ready to process the sentential form from left to right again. □

Taking $m = 0$ in Claim 7.4, we obtain that the word $\lambda(w)\#$ can be derived, which concludes the proof of the direct implication of Main Claim.

To prove the converse of Main Claim, we have to consider words derived by our rewriting system which belong to one of the following languages:

$$\begin{aligned}
L_1 &= \lambda(V^*)(h\hbar)^*\#(\hbar h)^*\rho(V^*)(\hbar h)^{-1}\\
L_2 &= (h\hbar)^{-1}\lambda(V^*)(h\hbar)^*\#(\hbar h)^*\rho(V^*)\\
L_3 &= g\bar{g}\tilde{g}£^2ab\hbar(h\hbar)^*\#(\hbar h)^*\hbar
\end{aligned}$$

We shall show that each word of this form derivable in the rewriting system corresponds to a sentential form of G.

Note that all derivations constructed in Claims 7.1–7.4 generally go through words in $L_1 \cup L_2 \cup L_3$, and between every two such words there is only a bounded number of intermediate words. We shall show that if a derivation proceeds in an essentially different way than as given in the above Claims, then it inevitably leads to a word from which no word belonging to any of these three languages can ever be derived.

Many "wrong" derivation steps can be identified immediately by some prohibited combinations of letters.

Claim 7.5 (Prohibited pairs)**.** *Let $\alpha \in \Gamma^*\#\Gamma^*$ be a word, from which some element of $L_1 \cup L_2$ can be derived. Then every occurrence of g, h, a or c to the left of $\#$ is followed by $\bar{g}$, $\hbar$, b or h, respectively. Similarly, every occurrence of g, h, a or c to the right of $\#$ is preceded by $\bar{g}$, $\hbar$, b or h, respectively.*

Proof. Denote $\Gamma \cup \{\#\}$ by Ξ. It is easy to verify that for every $\alpha \in L_1 \cup L_2$ the statement of the claim holds. Then it is sufficient to show that a forbidden factor from the set

$$g(\Xi \setminus \{\bar{g}\}) \cup h(\Xi \setminus \{\hbar\}) \cup a(\Xi \setminus \{b\}) \cup c(\Xi \setminus \{h\})$$

to the left of $\#$ cannot be removed by the rewriting, and similarly, a factor from the set

$$(\Xi \setminus \{\bar{g}\})g \cup (\Xi \setminus \{\hbar\})h \cup (\Xi \setminus \{b\})a \cup (\Xi \setminus \{h\})c$$

cannot be removed on the right. This is true, because every occurrence of g, h, a and c in a word from L_- is immediately followed by $\bar{g}$, $\hbar$, b and h, respectively. For factors on the right, the same argument applies to R_-. □

Claim 7.6 (Prohibited prefixes and suffixes)**.** *Let $\alpha \in \Gamma^*\#\Gamma^*$ be a word, from which some element of $L_1 \cup L_2$ can be derived. If the first letter of α is £, then the second letter is £ or a, and if the first letter is ¢, then the second one is ¢ or c. Similarly, if the last letter is £, then the second last is £ or a, and if the last letter is ¢, then the second last is ¢ or c.*

Proof. If the initial letter of α is £ or ¢, then, by Claim 7.5, until anything is removed at the left, only the words £, ¢ $\in L_+$ may be added there (any other word in L_+ would form a prohibited pair with each of these letters). Because no word in $L_1 \cup L_2$ starts with £ or with ¢, these letters are eventually removed. This can only be achieved using one of the words $¢^n ch\hbar$, $£^2 ab \in L_-$. Therefore, the second letter of α is either a or £ if the initial one is £, and c or ¢ if the initial one is ¢. The second part of the claim can be treated in exactly the same way. □

The above two technical conditions will now be used to compile the list of potentially applicable rules for different top letters in the stacks. This list will be used many times in the following arguments to justify that some word must be rewritten using a rule from a short list given by this claim.

Claim 7.7 (Rules allowed in a context)**.** *Let a rule $\mu \to \nu \in \overset{\ell r}{\to}$ ($\mu \to \nu \in \overset{r\ell}{\to}$, respectively) be applied to a word $\alpha s \in \Gamma^+\#\Gamma^*$ ($s\alpha \in \Gamma^*\#\Gamma^+$, respectively), with $s \in \Gamma \cup \{\#\}$, and assume that it is possible to derive some word in $L_1 \cup L_2$ from the resulting word. Then*

- *s does not belong to $V \cup \{\#, \$, d, e, g, \tilde{g}\}$.*
- *If $s \in \{£, a\}$, then $\nu = £$.*
- *If $s \in \{¢, c\}$, then $\nu = ¢$.*
- *If $s = b$, then $\nu = a£^2\tilde{g}\bar{g}\$ \in R_+$ ($\nu = \$\bar{g}\tilde{g}£^2a \in L_+$, respectively).*
- *If $s = \bar{g}$, then $\nu = g$.*
- *If $s = h$, then ν is one of the words from R_+ starting with c (one of the words from L_+ ending with c, respectively), that is, $\nu = c¢^nde\hbar\$ \in R_+$, $\nu = c¢^nA\hbar\$^{\varphi(A)} \in R_+$ with $A \in V$, or $\nu = (\rho(v)h^{-1})\$^{\varphi(u\to v)} \in R_+$ with $u \to v \in P$ in the case of appending to the right, and $\nu = \$^2\hbar S¢^nc \in L_+$ or $\nu = \$^{\varphi(A)}\hbar A¢^nc \in L_+$ with $A \in V$ in the case of appending to the left.*
- *If $s = \hbar$, then $\nu = h \in R_+$ or $\nu = h\hbar b \in R_+$ ($\nu = h \in L_+$ or $\nu = b\hbar h \in L_+$, respectively).*

Proof. To prove this claim, one has to consider all possible combinations of top letters and elements of R_+ (L_+, respectively) that are *not* listed in the statement, and in each case a contradiction with Claim 7.5 or with Claim 7.6 is obtained. The combinations listed in the statement are exactly those that do not yield an immediate contradiction with these claims.

For instance, for $s = £$ and for $\nu = h\hbar b \in R_+$, the resulting word will be $\mu^{-1}\alpha£h\hbar b$, which contains an occurrence of h to the right of $\#$ that is not preceded by $\hbar$, contradicting Claim 7.5. If $s = £$ and $\nu = ¢ \in R_+$, the resulting word is $\mu^{-1}\alpha£¢$, which contradicts Claim 7.6. In this way each element of R_+ is considered, and for all except $£$ a contradition is obtained.

All other cases are handled in exactly the same way. □

The next property of our rewriting system we prove ensures that the information about a symbol being moved or a production being applied is always correctly communicated to the other side. This is done by comparing the length of the block of \$s at one end of the word to the length of the block of ¢s or £s at the other end.

Claim 7.8 (Soundness of data transfer)**.** *Let $m, k \in \mathbb{N}$ and $\alpha \in \Gamma^* \# \Gamma^*$ be arbitrary, and let $s \in \{\bar{g}, \hbar\}$. Then some word from $L_1 \cup L_2$ can be derived from the word $¢^m c\alpha s\$^k$ ($\$^k s\alpha c¢^m$, respectively) only if $m + k = n$, and every derivation leading to a word from $L_1 \cup L_2$ starts by applying k times the rule $(\$, ¢) \in \overset{r\ell}{\to}$ ($(\$, ¢) \in \overset{\ell r}{\to}$, respectively).*

Furthermore, some word from $L_1 \cup L_2$ can be derived from the word $£^m a\alpha s\$^k$ ($\$^k s\alpha a£^m$, respectively) only if $m = k = 1$, and every derivation leading to a word from $L_1 \cup L_2$ starts by applying the rule $(\$, £) \in \overset{r\ell}{\to}$ ($(\$, £) \in \overset{\ell r}{\to}$, respectively).

Proof. We deal only with the case of the word $¢^m c\alpha s\$^k$; the other cases can be treated similarly. Consider any derivation starting from $¢^m c\alpha s\$^k$ and ending at any word from $L_1 \cup L_2$. By Claim 7.7, the relation $\overset{\ell r}{\to}$ is not applicable at the first step, hence the derivation starts with the application of a rule $\mu \to \nu \in \overset{r\ell}{\to}$, which must have $\mu = \$$. On the other hand, Claim 7.7 asserts that $\nu = ¢$, and the word is rewritten to $¢^{m+1} c\alpha s\$^{k-1}$. The same rule must be applied until there is no \$ at the end. The resulting word $¢^{m+k} c\alpha s$ ends with s, which is not the last letter of any element of R_-, so some rule $\mu' \to \nu' \in \overset{\ell r}{\to}$ must follow. This rule must have $\mu' = ¢^n ch\hbar$, since no other words from L_- apply, and therefore $m + k = n$. □

Now, with all technical results established, we can prove how exactly the rewriting *must* proceed in order to derive any word in $L_1 \cup L_2$.

It turns out that for every derivable word which is not in $L_1 \cup L_2 \cup L_3$ the next rule to apply is determined uniquely: if a different rule is used, the subsequent derivation can no longer arrive at any word in $L_1 \cup L_2$. Branching of a derivation is only possible at words belonging to $L_1 \cup L_2 \cup L_3$. For words in $L_1 \cup L_2$, the choice is whether to move a symbol to the other stack or to apply a production of G, as well as which production to apply. For words in L_3, the choice is whether to generate two more units of fuel or to proceed to a word from L_1 encoding the start symbol of G.

Claim 7.9 (Soundness of the simulation)**.** *Let $x_0, y_0 \in V^*$ and let*

$$\alpha \in \underbrace{\lambda(y_0)(h\hbar)^* \#(\hbar h)^* \rho(x_0)(\hbar h)^{-1}}_{\subseteq L_1} \cup \underbrace{(h\hbar)^{-1} \lambda(y_0)(h\hbar)^* \#(\hbar h)^* \rho(x_0)}_{\subseteq L_2}$$

be a word derivable in the rewriting system. Then $x_0 y_0$ can be generated in G.

Proof. The claim will be proved by induction on the length of the derivation of α. The induction assumption states that the claim holds for every word from $L_1 \cup L_2$ occurring in this derivation before α. Let β be the last word in this derivation that belongs to $L_1 \cup L_2 \cup L_3$. Such a word β clearly exists since the initial word belongs to L_3. The argument proceeds by considering possible subsequent words in this derivation: since from each of them one

can derive $\alpha \in L_1 \cup L_2$, Claims 7.7 and 7.8 generally apply to each rewriting step, and they impose some restrictions on the applicable rules.

The below case study shows that, in fact, once the rewriting rule applied to β is fixed, only one rule will be applicable at each subsequent step until α is reached. In this way we determine how α was obtained from β. If $\beta \in L_1 \cup L_2$, then it corresponds to a sentential form of G, and the derivation from β to α represents a transformation of this sentential form: either by moving a symbol from V from one stack to the other, or by applying a production of G (chosen by the rule applied to β). If $\beta \in L_3$, then it was obtained during fuel generation, and α is derived from β either by generating one more unit of fuel in each stack, or by closing the fuel generation phase and producing an encoded sentential form S.

Case I. Let us first assume that β belongs to L_1, that is,

$$\beta = \lambda(y)(h\hbar)^m \#(\hbar h)^k \rho(x)(\hbar h)^{-1}$$

for some $x, y \in V^*$ and $m, k \in \mathbb{N}_0$. By the induction hypothesis the word xy can be generated by G. We have to distinguish two cases.

Case I(a). If $x \neq \varepsilon$, let $x = x'A$ for some $A \in V$. Then

$$\beta = \lambda(y)(h\hbar)^m \#(\hbar h)^k \rho(x')c\text{¢}^n A, \tag{7}$$

and, by Claim 7.7, the first step of the derivation uses $\xrightarrow{r\ell}$. The word removed on the right is $\text{¢}^{\varphi(A)}A$ and, since the first letter of β is h or $\#$, Claim 7.7 implies that only words $\$^2\hbar S\text{¢}^n c$ and $\$^{\varphi(B)}\hbar B\text{¢}^n c$ with $B \in V$ can be added on the left. In the former case,

$$(7) \implies \$^2\hbar S\text{¢}^n c\lambda(y)(h\hbar)^m \#(\hbar h)^k \rho(x')c\text{¢}^{n-\varphi(A)}. \quad \lightning$$

By Claim 7.8, no word from $L_1 \cup L_2$ can be derived from the resulting word, since the sum of the number of \$s at the beginning and the number of ¢s at the end is strictly less than n. So this case is impossible and $\$^{\varphi(B)}\hbar B\text{¢}^n c$ must be added to the left. By the same Claim 7.8 applied to the resulting word, $\varphi(B) + n - \varphi(A) = n$, which, due to the bijectivity of φ, implies that $A = B$, and the word added to the left is actually $\$^{\varphi(A)}\hbar A\text{¢}^n c$, that is,

$$(7) \implies \$^{\varphi(A)}\hbar A\text{¢}^n c\lambda(y)(h\hbar)^m \#(\hbar h)^k \rho(x')c\text{¢}^{n-\varphi(A)}. \tag{8}$$

Claim 7.8 further asserts that the rewriting must continue by applying $\varphi(A)$ times the rule $(\$, \text{¢}) \in \xrightarrow{\ell r}$:

$$(8) \implies^{\varphi(A)} h^{-1}\lambda(Ay)(h\hbar)^m \#(\hbar h)^k \rho(x')c\text{¢}^n. \tag{9}$$

Because the word obtained in (9) begins with $\hbar$, no prefix of it belongs to L_-, and hence the rule applied after (9) must belong to $\xrightarrow{r\ell}$. The only word which

can be deleted on the right is $\hbar hc¢^n$, and by Claim 7.7 only h or $b\hbar h$ can be added to the left.

If $b\hbar h$ is used, the derivation proceeds as

$$(9) \overset{r\ell}{\Longrightarrow} b\hbar\lambda(Ay)(h\hbar)^m\#(\hbar h)^k\rho(x')(\hbar h)^{-1}. \tag{9'}$$

As the initial letter of the resulting word is b, from which no words in L_- begin, no rules from $\overset{\ell r}{\to}$ are applicable. Hence the next step uses $\overset{r\ell}{\to}$. The word deleted on the right must be of the form $¢^{\varphi(B)}B$, for some $B \in V$. By Claim 7.7, the word added to the beginning must be $\$\bar{g}\tilde{g}£^2a$:

$$(9') \overset{r\ell}{\Longrightarrow} \$\bar{g}\tilde{g}£^2ab\hbar\lambda(Ay)(h\hbar)^m\#(\hbar h)^k\rho(x'\cdot B^{-1})c¢^{n-\varphi(B)}. \quad \lightning$$

In the resulting word, the sum of the number of \$s at the beginning and ¢s at the end is less than n, which contradicts Claim 7.8.

This means that the letter h is added to (9):

$$(9) \overset{r\ell}{\Longrightarrow} \lambda(Ay)(h\hbar)^m\#(\hbar h)^k\rho(x')(\hbar h)^{-1},$$

where the word obtained lies in L_1. By the choice of β, this word must be α. This shows that $x_0y_0 = x'Ay = xy$, and so x_0y_0 can be generated by G by the induction hypothesis.

Case I(b). If $x = \varepsilon$, then

$$\beta = \lambda(y)(h\hbar)^m\#(\hbar h)^{k-1}. \tag{10}$$

In this case β has no suffix from R_- and so a rule from $\overset{\ell r}{\to}$ must be applied. The word removed on the left can be either $\lambda(A)(¢^{n-\varphi(A)}c)^{-1}$ with $A \in V$, in which case let $y = Ay'$, or $\lambda(u)(¢^{n-\varphi(u\to v)}c)^{-1}$ with $u \to v \in P$, in which case let $y = uy''$. Since β ends with h or $\#$, by Claim 7.7, the only words from R_+ which can be appended to β are $c¢^nde\hbar\$$, $(\rho(B)h^{-1})\$^{\varphi(B)}$ with $B \in V$, or $(\rho(v')h^{-1})\$^{\varphi(u'\to v')}$ with $u' \to v' \in P$. By Claim 7.8, the sum of the number of \$s at the end of the resulting word and the number of ¢s at its beginning is n, which limits the possible combinations of removals and appends to the following: (1) if $\lambda(A)(¢^{n-\varphi(A)}c)^{-1}$ was removed, then only $(\rho(A)h^{-1})\$^{\varphi(A)}$ can be appended; (2) if $\lambda(u)(¢^{n-\varphi(u\to v)}c)^{-1}$ was removed, then only $(\rho(v)h^{-1})\$^{\varphi(u\to v)}$ can be appended. Hence the next step is either

$$(10) \overset{\ell r}{\Longrightarrow} ¢^{n-\varphi(A)}c\lambda(y')(h\hbar)^m\#(\hbar h)^{k-1}(\rho(A)h^{-1})\$^{\varphi(A)} \quad \text{or} \tag{11a}$$

$$(10) \overset{\ell r}{\Longrightarrow} ¢^{n-\varphi(u\to v)}c\lambda(y'')(h\hbar)^m\#(\hbar h)^{k-1}(\rho(v)h^{-1})\$^{\varphi(u\to v)}. \tag{11b}$$

In addition, Claim 7.8 states that these words have to be further rewritten by removing all \$s from the right and adding the same number of ¢s to the left. The word obtained after these modifications is either

$$(11a) \overset{r\ell}{\Longrightarrow}{}^{\varphi(A)} ¢^nc\lambda(y')(h\hbar)^m\#(\hbar h)^{k-1}\rho(A)h^{-1} \quad \text{or} \tag{12a}$$

$$(11b) \overset{r\ell}{\Longrightarrow}{}^{\varphi(u\to v)} ¢^nc\lambda(y'')(h\hbar)^m\#(\hbar h)^{k-1}\rho(v)h^{-1}. \tag{12b}$$

Because in both cases the resulting word ends with $\hbar$, it has no suffix from R_-, and so a rule from $\xrightarrow{\ell r}$ must be applied at the next step. The word removed from the beginning is clearly $\text{¢}^n ch\hbar$, and Claim 7.7 ensures that the only words from R_+ that can be appended are h and $h\hbar b$.

Appending $h\hbar b$ results in

$$\text{(12a)} \xRightarrow{\ell r} (h\hbar)^{-1}\lambda(y')(h\hbar)^m\#(\hbar h)^{k-1}\rho(A)\hbar b \quad \lightning \quad \text{or in}$$
$$\text{(12b)} \xRightarrow{\ell r} (h\hbar)^{-1}\lambda(y'')(h\hbar)^m\#(\hbar h)^{k-1}\rho(v)\hbar b, \quad \lightning$$

respectively. In both cases no rule from $\xrightarrow{r\ell}$ is applicable (since none of the words in R_- ends with b), and only the word $a\pounds^2\tilde{g}\bar{g}\$$ can be added to the right by Claim 7.7. On the left, only one of the words $\lambda(B)(\text{¢}^{n-\varphi(B)}c)^{-1}$ with $B \in V$, or $\lambda(u')(\text{¢}^{n-\varphi(u'\to v')}c)^{-1}$ with $u' \to v' \in P$, can be deleted. This would contradict Claim 7.8, since the sum of the number of \$s at the end and ¢s at the beginning would be less than n.

Therefore, after the derivation step (12), h must be appended at the next step, and the derivation proceeds as follows:

$$\text{(12a)} \xRightarrow{\ell r} (h\hbar)^{-1}\lambda(y')(h\hbar)^m\#(\hbar h)^{k-1}\rho(A) \quad \text{or}$$
$$\text{(12b)} \xRightarrow{\ell r} (h\hbar)^{-1}\lambda(y'')(h\hbar)^m\#(\hbar h)^{k-1}\rho(v).$$

Because in both cases the word obtained belongs to L_2, by the definition of β, we have just derived α. The corresponding word x_0y_0 is either $Ay' = xy$ or vy'', where $xy = uy''$ and G generates vy'' from uy''. In both cases this shows that x_0y_0 can be generated by G.

Case II. The case of $\beta \in L_2$ is symmetric. More precisely, in the case of L_1, pieces of a sentential form can be moved to the right only when $x = \varepsilon$, while in the case of L_2, they can be moved to the left only when $y = \varepsilon$. So the only difference is in which situations productions of G can be applied: for $\beta \in L_1$ this happens only when the whole sentential form is in the left stack, whereas for $\beta \in L_2$ applicability of productions is not restricted. This, however, has no impact on the required arguments, since rules for productions can be treated in exactly the same way as rules for moving symbols between stacks.

Case III. It remains to deal with $\beta \in L_3$. So assume that

$$\beta = g\bar{g}\tilde{g}\pounds^2 ab\hbar(h\hbar)^m\#(\hbar h)^k\hbar, \tag{13}$$

where $m, k \in \mathbb{N}_0$. Since no words in R_- end with $\hbar$, the rule applied to β must belong to $\xrightarrow{\ell r}$, and on the left $g\bar{g} \in L_-$ must be removed. By Claim 7.7, either h or $h\hbar b$ can be added on the right.

Case III(a). Suppose the word $h\hbar b$ is appended at the first step:

$$(13) \xRightarrow{\ell r} \tilde{g}\pounds^2 ab\hbar(h\hbar)^m\#(\hbar h)^{k+1}\hbar b. \tag{14}$$

Since the resulting word ends with b, it has no suffix from R_-, so a rule from $\xrightarrow{\ell r}$ is used next. The only prefix that can be removed is $\tilde{g}\pounds \in L_-$. By Claim 7.7, the ending b allows only one word to be appended, namely $a\pounds^2\tilde{g}\bar{g}\$$, so the next step is

$$(14) \implies \pounds ab\hbar(h\hbar)^m\#(\hbar h)^{k+1}\hbar ba\pounds^2\tilde{g}\bar{g}\$. \tag{15}$$

The second part of Claim 7.8 shows that the rule $(\$, \pounds) \in \xrightarrow{r\ell}$ must be applied next:

$$(15) \implies \pounds^2 ab\hbar(h\hbar)^m\#(\hbar h)^{k+1}\hbar ba\pounds^2\tilde{g}\bar{g}. \tag{16}$$

This word has no suffix from R_-, so no deletion at the end is possible. Therefore, a rule from $\xrightarrow{\ell r}$ is applied: $\pounds^2 ab \in L_-$ is the only prefix that can be removed, and by Claim 7.7 only $g \in R_+$ can be appended. This leads to

$$(16) \implies \hbar(h\hbar)^m\#(\hbar h)^{k+1}\hbar ba\pounds^2\tilde{g}\bar{g}g. \tag{17}$$

As before, the suffix $\bar{g}g \in R_-$ is then deleted, and either $h \in L_+$ or $b\hbar h \in L_+$ can be added to the left.

Addition of h produces

$$(17) \implies (h\hbar)^{m+1}\#(\hbar h)^{k+1}\hbar ba\pounds^2\tilde{g}, \quad \lightning$$

and no prefix of the resulting word is in L_-. Hence, $\xrightarrow{r\ell}$ must be used, and $\pounds\tilde{g} \in R_-$ is the only suffix that can be removed. By Claim 7.7, the word added on the left is either $\$^2\hbar S\text{¢}^n c \in L_+$ or $\$^{\varphi(A)}\hbar A\text{¢}^n c \in L_+$ with $A \in V$. In each case we get a word in $\$^\ell\hbar\Gamma^*\#\Gamma^*a\pounds$ with $\ell \geq 2$, which contradicts Claim 7.8.

Therefore, $b\hbar h$ is added to the word obtained in (17), deriving

$$(17) \implies b\hbar(h\hbar)^{m+1}\#(\hbar h)^{k+1}\hbar ba\pounds^2\tilde{g}. \tag{18}$$

Using arguments symmetric to the previous ones (cf. the three-step transition from (14) to (17)), this word must be rewritten as follows:

$$\begin{aligned}(18) &\implies \$\bar{g}\tilde{g}\pounds^2 ab\hbar(h\hbar)^{m+1}\#(\hbar h)^{k+1}\hbar ba\pounds \\ &\implies \bar{g}\tilde{g}\pounds^2 ab\hbar(h\hbar)^{m+1}\#(\hbar h)^{k+1}\hbar ba\pounds^2 \\ &\implies g\bar{g}\tilde{g}\pounds^2 ab\hbar(h\hbar)^{m+1}\#(\hbar h)^{k+1}\hbar.\end{aligned}$$

The last word belongs to L_3. Because it is obtained after β in the derivation of α, we get a contradiction with the choice of β.

Case III(b). Now consider the situation when the letter h is appended to β, that is, the first step is

$$(13) \implies \tilde{g}\pounds^2 ab\hbar(h\hbar)^m\#(\hbar h)^{k+1}. \tag{19}$$

Since the word obtained has no suffixes from R_-, another rule from $\xrightarrow{\ell r}$ has to follow. The only prefix that can be erased is $\tilde{g}\pounds \in L_-$, while the word from R_+ appended on the right, by Claim 7.7, may be $c¢^n de\hbar\$$, $c¢^n A\hbar\$^{\varphi(A)}$ with $A \in V$, or $(\rho(v)h^{-1})\$^{\varphi(u\to v)}$ with $u \to v \in P$. The resulting word is in $\pounds a\Gamma^*\#\Gamma^*\hbar\$^\ell$ in all these cases, and from the second part of Claim 7.8 one can see that ℓ must be equal to 1, that is, the word $c¢^n de\hbar\$$ is added to the right:

$$(19) \implies \pounds ab\hbar(h\hbar)^m\#(\hbar h)^{k+1}c¢^n de\hbar\$. \tag{20}$$

Claim 7.8 additionally asserts that the rule $(\$, \pounds) \in \xrightarrow{r\ell}$ is employed at the next step:

$$(20) \implies \pounds^2 ab\hbar(h\hbar)^m\#(\hbar h)^{k+1}c¢^n de\hbar. \tag{21}$$

No words in R_- end with $\hbar$, hence a rule from $\xrightarrow{\ell r}$ is applied next. The prefix $\pounds^2 ab$ has to be deleted, and, by Claim 7.7, one of the words $h\hbar b$ and h is appended, resulting in

$$(21) \implies \hbar(h\hbar)^m\#(\hbar h)^{k+1}c¢^n de\hbar h\hbar b \quad \lightning \quad \text{or}$$

$$(21) \implies \hbar(h\hbar)^m\#(\hbar h)^{k+1}c¢^n de\hbar h. \tag{22}$$

None of the resulting words has any prefix from L_-. The former word also has no suffix from R_-, hence no rules are applicable to it and that case is impossible.

Therefore, this derivation step must be (22). At the next step, since no words in L_- start with $\hbar$, only $\xrightarrow{r\ell}$ can be applied: the suffix $e\hbar h$ is removed, and the word added to the beginning must be either $b\hbar h$ or h by Claim 7.7. The derivation proceeds as:

$$(22) \implies b\hbar(h\hbar)^{m+1}\#(\hbar h)^{k+1}c¢^n d \quad \text{or} \tag{23a}$$

$$(22) \implies (h\hbar)^{m+1}\#(\hbar h)^{k+1}c¢^n d. \tag{23b}$$

Let us show that the addition of $b\hbar h$ leads to a contradiction. Since no word of L_- starts with b, only $\xrightarrow{r\ell}$ would be applicable to (23a). The only suffix that could be removed is $¢^2 d \in R_-$, and by Claim 7.7 only $\$\bar{g}\tilde{g}\pounds^2 a \in L_+$ could be added at the left:

$$(23a) \implies \$\bar{g}\tilde{g}\pounds^2 ab\hbar(h\hbar)^{m+1}\#(\hbar h)^{k+1}c¢^{n-2}. \quad \lightning$$

By Claim 7.8, α could not be derived from this word, because the sum of the number of leading \$s and the number of terminating ¢s would be only $n-1$.

Hence, the rewriting proceeds according to (23b). By Claim 7.7, no rules from $\xrightarrow{\ell r}$ can be applied to the resulting word. Consequently, the word $¢^2 d \in R_-$ is removed from the right and, by Claim 7.7, the only words that can be

added to the left are $\$^2\hbar S¢^n c \in L_+$ and $\$^{\varphi(A)}\hbar A¢^n c \in L_+$ with $A \in V$. The latter case is impossible due to Claim 7.8, since adding $\varphi(A)$ \$s would result in the total number of \$s in the beginning and ¢s in the end exceeding n. Therefore, the next step is

$$(23b) \stackrel{r\ell}{\Longrightarrow} \$^2\hbar S¢^n c(h\hbar)^{m+1}\#(\hbar h)^{k+1}c¢^{n-2}. \tag{24}$$

Then, in accordance with Claim 7.8, the rule $(\$, ¢) \in \stackrel{\ell r}{\rightarrow}$ must be applied twice:

$$(24) \stackrel{\ell r}{\Longrightarrow}^2 \hbar S¢^n c(h\hbar)^{m+1}\#(\hbar h)^{k+1}c¢^n. \tag{25}$$

Using the same arguments as for (9) one can show that the rule $(\hbar hc¢^n, h) \in \stackrel{r\ell}{\rightarrow}$ is applied next:

$$(25) \stackrel{r\ell}{\Longrightarrow} \lambda(S)(h\hbar)^{m+1}\#(\hbar h)^k.$$

The resulting word belongs to L_1, and by the definition of β, it has to be α. Hence, the word x_0y_0 corresponding to α is just the start symbol S. This completes the proof of Claim 7.9. □

Now we can verify the converse of Main Claim. For every $w \in V^*$, if the word $\lambda(w)\#$ can be derived in our rewriting system, then by Claim 7.9 the word w can be generated by G. Q. E. D.

Corollary 3. *There exist finite uncontrolled relations $\stackrel{\ell r}{\rightarrow}$ and $\stackrel{r\ell}{\rightarrow}$ and a word w such that the language generated by the two-way rewriting system from w is r.e.-complete.*

Theorem 7 also implies that emptiness of intersection is undecidable already for a very restricted class of context-free languages.

Corollary 4. *The intersection non-emptiness problem is undecidable for binary context-free languages recognized by three-state pushdown automata of the form of Figure 7, where the two ε-transitions define initial and final contents of the stack and the whole computation takes place in the middle state, with transitions labelled by one letter pushing to the stack and those labelled by the other letter popping.*

Sketch of a proof. Using the previous corollary, let L be an r.e.-complete language generated by a two-way rewriting system defined by an initial word $u_0\#v_0$ and uncontrolled relations $\stackrel{\ell r}{\rightarrow}$ and $\stackrel{r\ell}{\rightarrow}$ over the alphabet Σ. We are going to prove that a word $u\#v$ is generated by our rewriting system if and only if the languages recognized by the pushdown automata $\mathcal{A}$ and $\mathcal{B}$ in Figure 7 are not disjoint. Then decidability of the intersection non-emptiness problem would imply that the language L is recursive, and so the problem is necessarily undecidable.

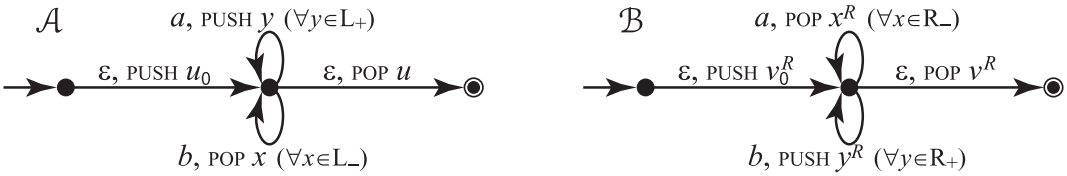


Figure 7: Automata $\mathcal{A}$ and $\mathcal{B}$ in Corollary 4.

Intuitively, each of the automata $\mathcal{A}$ and $\mathcal{B}$ encodes in its stack manipulations of one side of the word, while the letters a and b represent applications of $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$, respectively. When $\mathcal{A}$ reads a, it simulates the impact of $\overset{\ell r}{\to}$ on the left part of the word, that is, removes any prefix belonging to L_- from the word contained in the stack; when $\mathcal{B}$ reads a, it simulates $\overset{\ell r}{\to}$ by appending any element of R_+ to its stack word. The existence of a word recognized by both automata ensures that these manipulations can be synchronized, and therefore realized by the relations $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$. Now we verify our claim more formally.

First, assume that $u\#v \in L$ and fix any derivation producing $u\#v$ from $u_0\#v_0$ in our rewriting system. Let ℓ be the length of this derivation. We construct a word w over $\{a, b\}$ of length ℓ by taking for $n = 1, \ldots, \ell$ the n-th letter of w equal to a if the n-th step of the derivation uses a rule from $\overset{r\ell}{\to}$, and equal to b if this step uses a rule from $\overset{\ell r}{\to}$. Then w is recognized by both automata $\mathcal{A}$ and $\mathcal{B}$. To verify this for the automaton $\mathcal{A}$, we determine the n-th step of $\mathcal{A}$ in the middle state as follows: if the n-th letter of w is a and the rule $x \overset{r\ell}{\to} y$ was used in the n-th step of the derivation, then y is pushed, and if the n-th letter of w is b and the rule $x \overset{\ell r}{\to} y$ was used, then x is popped. During this computation of $\mathcal{A}$, after reading the n-th letter of w, the stack contains exactly the word which is obtained to the left from # after the n-th step of the derivation. In particular, once the whole word w has been read, the stack contains precisely u, which is then popped by the transition to the final state. For the automaton $\mathcal{B}$, the arguments are analogous; the stack of $\mathcal{B}$ always contains the reverse of the word to the right from #. This shows that the languages recognized by $\mathcal{A}$ and $\mathcal{B}$ are not disjoint.

Conversely, let $w \in \{a, b\}^*$ be a word recognized by both $\mathcal{A}$ and $\mathcal{B}$, and let us choose an arbitrary accepting computation on w in each automaton. Then $u\#v$ can be derived in our rewriting system by performing one step for each letter of w as follows: If the n-th letter of w is a, then in the n-th step of the derivation we apply the rule $x \overset{r\ell}{\to} y$, where $x \in R_-$ is the reverse of the word popped by $\mathcal{B}$ and $y \in L_+$ is the word pushed by $\mathcal{A}$ when reading this letter of w in the accepting computation. Similarly, if the n-th letter is b, then we apply $x \overset{\ell r}{\to} y$, where $x \in L_-$ is the word popped by $\mathcal{A}$ and $y \in R_+$ is the reverse of the word pushed by $\mathcal{B}$. □

7 Conclusion

Let us summarize the results of this paper. Four general cases of the communication of two stacks (namely, R-, S-, RS- and 2W-rewriting) were considered in Sections 3–6. In each case, the power of the resulting rewriting systems was determined for different families of relations for $\overset{\ell r}{\to}$, $\overset{r\ell}{\to}$, $\overset{\ell\ell}{\to}$ and $\overset{rr}{\to}$, as well as for different language families for the initial set I. It turned out that for each of the first three modes of rewriting, there is a dichotomy of expressive power according to one particular decisive factor.

- For R-rewriting, this is the initial set. If I is regular, then only regular sets can be generated even if powerful relations are used for $\overset{\ell r}{\to}$ and $\overset{rr}{\to}$. On the other hand, if I is allowed to be context-free, then computational universality is attained already for very weak rewriting relations.

- The power of S-rewriting is determined by the relation $\overset{\ell r}{\to}$. If it is uncontrolled, then such rewriting systems can be simulated by a special class of pushdown automata: the one-turn counter automata. If $\overset{\ell r}{\to}$ is controlled (at least able to copy symbols from the left stack to the right stack), then these systems can simulate context-free grammars even using uncontrolled $\overset{rr}{\to}$. In any case, S-rewriting systems cannot generate anything non-context-free.

- The same relation $\overset{\ell r}{\to}$ is decisive for the power of RS-rewriting. For an uncontrolled $\overset{\ell r}{\to}$, the languages generated by such a rewriting can still be recognized by one-turn counter automata. However, as soon as $\overset{\ell r}{\to}$ is able to copy symbols verbatim, the rewriting becomes computationally universal even if both $\overset{\ell\ell}{\to}$ and $\overset{rr}{\to}$ are uncontrolled.

Finally, two-way rewriting was found to be computationally universal even in its simplest possible case, when both $\overset{\ell r}{\to}$ and $\overset{r\ell}{\to}$ are finite uncontrolled relations. Hence, using any more general types of relations cannot essentially increase their expressive power.

The utmost succinct summary of our results is given in Table 1.

Acknowledgements

Research supported by the Academy of Finland under grants 118540, 206039 and 208414.

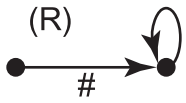
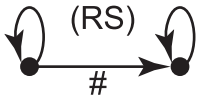
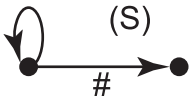
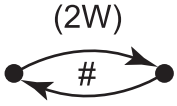
 Regular I: regularity. Context-free I: universality.	 Uncontrolled $\overset{\ell r}{\rightarrow}$: one-turn counters. Controlled $\overset{\ell r}{\rightarrow}$: universality.
 Uncontrolled $\overset{\ell r}{\rightarrow}$: one-turn counters. Controlled $\overset{\ell r}{\rightarrow}$: context-freeness.	 Universality.

Table 1: Legend to our results.

References

[1] B. S. Baker, R. V. Book, "Reversal-bounded multipushdown machines", *Journal of Computer and System Sciences*, 8 (1974), 315–332.

[2] M. Benois, "Parties rationnelles du groupe libre", *C. R. Acad. Sci. Paris Series A*, 269 (1969) 1188–1190.

[3] J. R. Büchi, "Regular canonical systems", *Arch. Math. Logic Grundlagenforsch*, 6 (1964), 91–111.

[4] J. H. Conway, *Regular Algebra and Finite Machines*, Chapman and Hall, 1971.

[5] D. Hofbauer, J. Waldmann, "Deleting string rewriting systems preserve regularity", *Theoretical Computer Science*, 327:3 (2004), 301–317.

[6] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[7] J. Karhumäki, M. Kunc, A. Okhotin, "Computing by commuting", *Theoretical Computer Science*, 356:1–2 (2006), 200–211.

[8] M. I. Kratko, "On a certain class of Post calculi", *Soviet Mathematics Doklady*, 6 (1965), 1544–1545.

[9] M. Kunc, "The power of commuting with finite sets of words", *Theory of Computing Systems*, 40:4 (2007), 521–551.

[10] A. N. Maslov, "Cyclic shift operation for languages", *Problems of Information Transmission*, 9:4 (1973), 333–338.

[11] T. Oshiba, "Closure property of the family of context-free languages under the cyclic shift operation", *Transactions of IECE*, 55D:4 (1972), 119–122.

[12] J.-E. Pin, J. Sakarovitch, "Une application de la représentation matricielle des transductions", *Theoretical Computer Science*, 35 (1985), 271–293.

[13] E. L. Post, "Formal reductions of the general combinatorial decision problem", *American Journal of Mathematics*, 65:2 (1943), 197–215.

[14] J. Sakarovitch, *Elements de théorie des automates*, Vuibert, 2003.

[15] A. Salomaa, *Formal Languages*, Academic Press, 1973.

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematical Sciences

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 952–12–1687–5
ISSN 1239-1891