Theoretical Computer Science 29 (1984) 185–227
North-Holland

HIERARCHIES OF PRIMITIVE RECURSIVE WORDSEQUENCE FUNCTIONS: COMPARISONS AND DECISION PROBLEMS

E. FACHINI and M. NAPOLI
Istituto di Scienze dell'Informazione, Facoltà di Scienze Matematiche, Fisiche e Naturali, Università di Salerno, 84100 Salerno, Italy

Communicated by C. Böhm
Received November 1982
Revised June 1983

Abstract. In this paper we consider wordsequence functions, i.e., functions of the type $f:\Sigma^{*^r}\to\Sigma^{*^s}$ where Σ is a finite alphabet and $r\geq 0$, $s>0$. By starting with finite sets of basic functions and by taking the closure with respect to composition, cylindrification and iteration, we give some characterizations of primitive recursive wordsequence functions. We define some hierarchies of length ω^2 of these functions by bounding the number of successive compositions and the depth of the nested iterations in the definitions of the functions. In such a manner we obtain refinements of the Axt, Grzegorczyk and Meyer and Ritchie generalized hierarchies of length ω of primitive recursive wordfunctions defined by Von Henke, Indermark and Weihrauch (1972).

We consider LOOP programs on words (see Ausiello and Moscarini (1976)) by allowing more than one output register, and we prove that the class of functions computed by these programs coincides with the class of primitive recursive wordsequence functions. The hierarchies of functions induce some hierarchies of programs.

For the case of functions $f:\Sigma^{*^r}\to\Sigma^*$, our hierarchies are compared with the Axt et al. generalized hierarchies.

We also compare our hierarchies with storage hierarchies, and we analyze the power of the LOOP programs as acceptors.

Finally, we state some decidability results for the considered classes.

Introduction

Partial recursive wordfunctions have been defined by Asser in [1]. Partial recursive sequence functions, i.e., partial recursive functions of the type $f:N^r\to N^s$ have been studied by Eilenberg and Elgot in [4] and by Germano and Maggiolo-Schettini in [7]. In this paper we consider wordsequence functions, i.e., functions of the type $f:\Sigma^{*^r}\to\Sigma^{*^s}$ where Σ is a finite alphabet and $r\geq 0$, $s>0$. These functions provide quite directly a semantics to programs for register machines. In fact, these last, ultimately, transform tuples of words into tuples of words.

By starting with finite sets of basic functions and by taking the closure with respect to composition, cylindrification and iteration, we give some characterizations of

0304-3975/84/$3.00 © 1984, Elsevier Science Publishers B.V. (North-Holland)

primitive recursive wordsequence functions. The different characterizations arise from the fact that a word may be read and written both rightwards and leftwards.

We define some hierarchies of length ω^2 of these functions by bounding the number of successive compositions and the depth of the nested iterations in the definitions of the functions. In such a manner we obtain refinements of the Axt, Grzegorczyk and Meyer and Ritchie generalized hierarchies of length ω of primitive recursive wordfunctions defined by Von Henke, Indermark and Weihrauch in [12]. If the cardinality of the alphabet under consideration is 1, the hierarchies coincide up to an isomorphism with the hierarchy of primitive recursive sequence functions defined by Fachini and Maggiolo-Schettini in [5]. Some properties of the classes of primitive recursive sequence functions are easily generalized to the corresponding classes of primitive recursive wordsequence functions. One of the major differences with respect to the numerical case is the fact that even classes of wordfunctions of the type $f:\Sigma^* \to \Sigma^*$ with unnested iterations form hierachies.

We consider LOOP programs on words (see Ausiello and Moscarini [2]) by allowing more than one output register, and we prove that the class of functions computed by these programs coincides with the class of primitive recursive wordsequence functions. The hierarchies of functions induce some hierarchies of programs.

At the level of the elementary functions, i.e., for functions with depth of nested iterations equal to 2, the hierarchies defined here coincide. Below this level we carry out all the comparisons with respect to the set-theoretical relationships among the classes. For the case of functions $f:\Sigma^{*^r} \to \Sigma^*$, our hierarchies are compared with the Axt, Grzegorczyk and Meyer and Ritchie generalized hierarchies.

We also compare our hierarchies with storage hierarchies (generalized sequential machines, two-way finite state transducers and deterministic push-down transducers) and we analyze the power of the LOOP programs as acceptors.

Finally, we state some decidability results for the considered classes. As regards the equivalence problem, the classes of wordsequence functions behave as the corresponding classes of functions $f:N^r \to N^s$ (see [5, 6]). In fact, if we add an instruction of the type IF "the content of the register is different from the empty word" THEN "sequence of instructions" ELSE "sequence of instructions" to the set of basic instructions of LOOP programs on words, we find that the equivalence problem is decidable for the class of functions computed by LOOP programs with unnested loop instructions. If the set of basic instructions also contains an instruction of deleting the first (or the last) symbol of the word contained in a register, the problem becomes undecidable. Moreover, we prove that the graph and the range intersection problems are undecidable for the class of wordfunctions computed by LOOP programs with just one unnested loop instruction, with cardinality of the alphabet greater than 1.

In Section 1, primitive recursive wordsequence functions are introduced and the relationships with primitive recursive wordfunctions and primitive recursive sequence functions are shown.

In Section 2, the chains of classes of primitive recursive wordsequence functions are defined, the behaviour of these classes with respect to some operators is stated, and the relationship with the corresponding classes of primitive recursive sequence functions is shown.

In Section 3 we consider chains of classes of LOOP programs on words and we prove that the corresponding chains of classes of functions coincide with the chains of classes of wordsequence functions already defined. We also prove that they form hierarchies and we study their behaviour with respect to the computation time function. Finally, we state the set-theoretical relationships among these classes and between these and the Axt, Grzegorczyk and Meyer and Ritchie classes of wordfunctions.

In Section 4 we recall the definitions of generalized sequential machine, two-way finite state transducer and deterministic push-down transducer, and we compare the classes of functions defined by such transducers with our classes. Moreover, classes of languages accepted by LOOP programs are defined and compared with the classes of languages of the Chomsky hierarchy.

In Section 5 some decision problems are dealt with.

The complete proofs of some theorems can be found in Appendix A.

1. Primitive recursive wordsequence functions

In this section we introduce a characterization of primitive recursive wordfunctions from sequences of words to sequences of words on a given finite alphabet, and we call them primitive recursive wordsequence functions. These functions are the primitive recursive sequence functions, defined in [5], up to an isomorphism, when the alphabet contains just one element.

Let $\Sigma = \{a_1, \ldots, a_n\}$ be the alphabet. Let us consider the following set of wordfunctions on Σ^*,

$$B_L = \{E = (e), K = \lambda x, y.(x), S_i = \lambda x.(a_i x) \text{ for } 1 \leq i \leq n\}$$

where e is the empty word of Σ^*.

Let us consider the following operators:

(1) the composition operator $\lambda f, g.(f \circ g)$ such that if $f: \Sigma^{*^r} \to \Sigma^{*^s}$ and $g: \Sigma^{*^s} \to \Sigma^{*^t}$, then $f \circ g: \Sigma^{*^r} \to \Sigma^{*^t}$ and $f \circ g(u) = g(f(u))$ for $u \in \Sigma^{*^r}$,

(2) the left cylindrification operator $\lambda f.({}^c f)$ such that if $f: \Sigma^{*^r} \to \Sigma^{*^s}$, then ${}^c f: \Sigma^{*^{r+1}} \to \Sigma^{*^{s+1}}$ and ${}^c f(x, u) = x, f(u)$ for $x \in \Sigma^*$ and $u \in \Sigma^{*^r}$,

(3) the right cylindrification operator $\lambda f.(f^c)$ such that if $f: \Sigma^{*^r} \to \Sigma^{*^s}$, then $f^c: \Sigma^{*^{r+1}} \to \Sigma^{*^{s+1}}$ and $f^c(u, x) = f(u), x$ for $u \in \Sigma^{*^r}$ and $x \in \Sigma^*$,

(4) the ℓ-iteration operator $\lambda f_1, \ldots, f_n.(f_1, \ldots, f_n)^{1*}$ such that if $f_i: \Sigma^{*^r} \to \Sigma^{*^r}$ for $1 \leq i \leq n$, then $(f_1, \ldots, f_n)^{1*}: \Sigma^{*^{r+1}} \to {*^r}$ and

$$(f_1, \ldots, f_n)^{1*}(e, x_1, \ldots, x_r) = x_1, \ldots, x_r,$$

$$(f_1, \ldots, f_n)^{1*}(a_i x, x_1, \ldots, x_r) = (f_1, \ldots, f_n)^{1*}(x, f_i(x_1, \ldots, x_r))$$

for $1 \leq i \leq n$ and $x, x_1, \ldots, x_r \in \Sigma^*$.

Definition 1.1. The set WS_n of primitive recursive wordsequence functions on Σ^*, with $\text{card}(\Sigma) = n$ is defined as the least set of wordsequence functions containing B_L and closed with respect to the composition, the left and right cylindrifications and the ℓ-iteration.

Whenever we are not interested in the cardinality of Σ, we refer to WS_n as WS.

Consider the following functions:

(1) the function $\Theta_i^r : \Sigma^{*^r} \to \Sigma^{*^r}$ $(r > 1,\ 1 < i \leq r)$ such that

$$\Theta_i^r(x_1, \ldots, x_r) = x_i, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_r,$$

(2) the function $\Delta^r : \Sigma^{*^r} \to \Sigma^{*^{2r}} (r > 0)$, such that

$$\Delta^r(u) = u, u \quad \text{for } u \in \Sigma^{*^r},$$

(3) the function $U_i^r : \Sigma^{*^r} \to \Sigma^*$ $(r > 0, 1 \leq i \leq r)$, such that

$$U_i^r(x_1, \ldots, x_r) = x_i,$$

(4) the function $T_{i,j}^r : \Sigma^{*^r} \to \Sigma^{*^r}$ $(r > 1,\ 1 \leq i \neq j \leq r)$, such that

$$T_{i,j}^r(x_1, \ldots, x_r) = x_1, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_r,$$

(5) the function $P_i : \Sigma^* \to \Sigma^*$ $(1 \leq i \leq n)$, such that

$$P_i(u) = ua_i \quad \text{for } u \in \Sigma^*.$$

Proposition 1.2. *The functions Θ_i^r, Δ^r, U_i^r, $T_{i,j}^r$ and P_i belong to WS.*

Proof. The proof immediately follows from the definitions.

Consider the following operators:

(1) the cartesian product $\lambda f, g.(f \times g)$ such that if $f : \Sigma^{*^r} \to \Sigma^{*^p}$ and $g : \Sigma^{*^q} \to \Sigma^{*^s}$, then $f \times g : \Sigma^{*^{r+q}} \to \Sigma^{*^{p+s}}$ and $(f \times g)(u, v) = f(u), g(v)$ for $u \in \Sigma^{*^r}$, $v \in \Sigma^{*^q}$,

(2) the juxtaposition operator $\lambda f, g.(f \hat{} g)$ such that if $f : \Sigma^{*^r} \to : \Sigma^{*^p}$ and $g : \Sigma^{*^r} \to \Sigma^{*^s}$, then $f \hat{} g : \Sigma^{*^r} \to \Sigma^{*^{p+s}}$ and $(f \hat{} g)(u) = f(u), g(u)$ for $u \in \Sigma^{*^r}$,

(3) the γ-iteration operator $\lambda f_1, \ldots, f_n.(f_1, \ldots, f_n)^{I^r}$ such that if $f_i : \Sigma^{*^r} \to \Sigma^{*^r}$, $1 \leq i \leq n$, then $(f_1, \ldots, f_n)^{I^r} : \Sigma^{*^{r+1}} \to \Sigma^{*^r}$ and

$$(f_1, \ldots, f_n)^{I^r}(e, x_1, \ldots, x_r) = x_1, \ldots, x_r,$$

$$(f_1, \ldots, f_n)^{I^r}(xa_i, x_1, \ldots, x_r) = (f_1, \ldots, f_n)^{I^r}(x, f_i(x_1, \ldots, x_r)).$$

Proposition 1.3. *The class WS is closed with respect to the cartesian product, the juxtaposition and the γ-iteration.*

Proof. The proof immediately follows from the definitions.

Definition 1.4. Let W be the set of primitive recursive wordfunctions defined as the smallest set of functions containing

$$\bar{B}_L = \{E = (e), E^1 = \lambda x.(e), S_i = \lambda x.(a_i x), 1 \leq i \leq n,$$
$$U_j^r = \lambda x_1, \ldots, x_r(x_j), r > 0, 1 \leq j \leq r\}$$

and closed with respect to

(1) the substitution operator $\lambda f, g_1, \ldots, g_k.(f \circ (g_1, \ldots, g_k))$, such that if $f : \Sigma^{*^k} \to \Sigma^*$ and $g_i : \Sigma^{*^r} \to \Sigma^*$ for $1 \leq i \leq k$, then $f \circ (g_1, \ldots, g_k)(u) = f(g_1(u), \ldots, g_k(u))$ for $u \in \Sigma^{*^r}$,

(2) the primitive recursion defined as follows: if $f : \Sigma^{*^k} \to \Sigma^*$ and $g_i : \Sigma^{*^{k+2}} \to \Sigma^*$ for $1 \leq i \leq n$, then $h : \Sigma^{*^{k+1}} \to \Sigma^*$ is obtained by primitive recursion from $f, g_1, \ldots, g_n$ iff

$$h(u, e) = f(u),$$
$$h(u, a_j x) = g_j(u, x, h(u, x)) \quad \text{for } u \in \Sigma^{*^k},\ x \in \Sigma^*,\ 1 \leq j \leq n.$$

Theorem 1.5. $WS = \{f = f_1 \hat{} \cdots \hat{} f_s \mid f_i : \Sigma^{*^r} \to \Sigma^* \in W,\ r \geq 0,\ s \geq 0\}$.

Proof. The proof is analogous to the corresponding theorem in the numerical case (see [5]).

Let $\mathbb{N}$ be the set of natural numbers.

A function $f^R : \mathbb{N}^{r+1} \to \mathbb{N}^r$ is obtained by repetition from $f : \mathbb{N}^r \to \mathbb{N}^r$ iff

$$f^R(0, u) = u,$$
$$f^R(S(x), u) = f(f^R(x, u)) \quad \text{for } u \in \mathbb{N}^r \text{ and } x \in \mathbb{N}.$$

Consider now the composition and the left and right cylindrifications on functions $f : \mathbb{N}^r \to \mathbb{N}^s$.

Definition 1.6. The set S of primitive recursive sequence functions $f : \mathbb{N}^r \to \mathbb{N}^s$ with $r \geq 0$, $s < 0$ is defined as the smallest set of functions containing

$$A = \{0 = (0),\ K' = \lambda x, y.(x),\ S = \lambda x.(x+1)\}$$

and closed with respect to composition, left and right cylindrifications and repetition.

We will use $|x|$ to denote the length or number of symbols in the word $x \in \Sigma^*$, and $|x_1, \ldots, x_r|$ to denote the sequence of lengths of the words $x_i \in \Sigma^*$, $1 \leq i \leq r$.

Proposition 1.7. *There exists a bijective function* $\Phi : S \to WS_1$ *such that*

(1) $\Phi f(|u|) = |f(u)|$,
(2) $\Phi(f \circ g) = \Phi f \circ \Phi g$,
(3) $\Phi({}^c f) = {}^c(\Phi f)$ *and* $\Phi(f^c) = (\Phi f)^c$,
(4) $\Phi f^R = (\Phi f)^I$.

Proof. The proof obviously follows from the definitions.

2. Classes of primitive recursive wordsequence functions

In this section we define four chains of classes $L_{i,j}^{\rightarrow L}$, $L_{i,j}^{\leftarrow L}$, $L_{i,j}^{\rightarrow R}$, $L_{i,j}^{\leftarrow R}$ for $i, j \geq 0$, of primitive recursive wordsequence functions, and we study the behaviour of these classes with respect to the operators introduced in the previous section. Moreover, we state the relationship between each chain and the hierarchy of primitive recursive sequence functions defined in [5].

Let

$$B_R = \{E = (e), K = \lambda x, y.(x), P_i = \lambda x.(xa_i) \text{ for } 1 \leq i \leq n\}.$$

Let $\Theta = \Theta_2^2$ and $\Delta = \Delta^1$.

Let $A_L = B_L \cup \{\Theta, \Delta\}$ and $A_R = B_R \cup \{\Theta, \Delta\}$ and, for a set X of wordfunctions, let $\mathscr{C}(X)$ be the closure of X with respect to the composition and left and right cylindrification and $\mathscr{I}^{\leftarrow}(X)$ ($\mathscr{I}^{\rightarrow}(X)$) the set of functions obtained from X by r-iteration (ℓ-iteration).

Definition 2.1

$$L_{0,0}^{\leftarrow L} = \mathscr{C}(A_L),$$

$$L_{i,j+1}^{\leftarrow L} = \{f = f_1 \circ f_2 \mid f_1 \in L_{i,j}^{\leftarrow L}, f_2 \in \mathscr{I}^{\leftarrow}(L_{i,0}^{\leftarrow L})\} \quad \text{for } i, j \geq 0,$$

$$L_{i,0}^{\leftarrow L} = \bigcup_{j \geq 0} L_{i-1,j}^{\leftarrow L} \quad \text{for } i \geq 1.$$

Let us use $L_i^{\leftarrow L}$ to denote the class $L_{i,0}^{\leftarrow L}$, for $i \geq 0$.

The class $L_{i,j}^{\leftarrow R}$ is analogously obtained by considering A_R instead of A_L in the above definition. The class $L_{i,j}^{\rightarrow L}$ ($L_{i,j}^{\rightarrow R}$) is obtained as $L_{i,j}^{\leftarrow L}$ ($L_{i,j}^{\leftarrow R}$) by considering the r-iteration instead of the ℓ-iteration.

Whenever we state a property holding for all the classes $L_{i,j}^{\leftarrow L}$, $L_{i,j}^{\rightarrow L}$, $L_{i,j}^{\rightarrow R}$, $L_{i,j}^{\leftarrow R}$ we will express it for $L_{i,j}$.

Lemma 2.2. *The following properties hold:*

(1) $L_{i,j} \subseteq L_{i,j+1}$, $L_i \subseteq L_{i+1}$,

(2) $\mathscr{C}(\mathscr{I}(L_i)) = L_{i+1}$,

(3) $\mathscr{C}(L_i) = L_i$, $\mathscr{I}(L_i) \subseteq L_{i,1}$ *for* $i \geq 0$,

(4) *if* $f \in L_{i,j}$, *then* $f = f_0 \circ f_1 \circ \cdots \circ f_j$ *with* $f_0 \in L_i$, $f_1, \ldots, f_j \in \mathscr{I}(L_{i,0})$,

(5) *if* $f \in L_{i,j}$, *then* ${}^c f \in L_{i,j}$ *and* $f^c \in L_{i,j}$.

Proof. The proof immediately follows from the definitions.

Lemma 2.3. *If* $f \in L_{i,j}$ *and* $g \in L_{i,k}$, *then* $f \circ g \in L_{i,j+k}$, $f \,\hat{}\, g \in L_{i,j+k}$, $f \times g \in L_{i,j+k}$.

Proof. The proof is analogous to the numerical case.

Lemma 2.4. $WS = \bigcup_{i,j\geq 0} L_{i,j}$.

Proof. The proof obviously follows from the definitions.

We now give a duality result..

Lemma 2.5. *There exists a bijective function* $\Phi: WS \to WS$ *such that*

(1) $\Phi(f_1 \circ f_2) = \Phi(f_1) \circ \Phi(f_2)$,

(2) $\Phi({}^{c}f) = {}^{c}(\Phi(f))$ *and* $\Phi(f^{c}) = (\Phi(f))^{c}$,

(3) $\Phi((g_1, \ldots, g_n)^{1\leftarrow}) = (\Phi(g_1), \ldots, \Phi(g_n))^{1\rightarrow}$ *and* $\Phi((g_1, \ldots, g_n)^{1\rightarrow}) = (\Phi(g_1), \ldots, \Phi(g_n))^{1\leftarrow}$,

(4) $\Phi(L_{i,j}^{\leftarrow L}) = L_{i,j}^{\rightarrow R}$ *and* $\Phi(L_{i,j}^{\rightarrow L}) = L_{i,j}^{\leftarrow R}$.

Proof. Let $\Phi: WS \to WS$ be such that if $f: \Sigma^{*^r} \to \Sigma^{*^s}$, then $\Phi(f) = \text{rev}^r \circ f \circ \text{rev}^s$ where $\text{rev}^1(e) = e$, $\text{rev}^1(a_i x) = (\text{rev}^1 \circ P_i)(x)$ and $\text{rev}^r = \underbrace{\text{rev}^1 \circ \cdots \circ \text{rev}^1}_{r \text{ times}}$ for $r > 1$.

It can easily be seen that properties (1), (2) and (3) hold. By induction on i and j and by the previous properties (1), (2) and (3), property (4) can be easily proved.

Let us now recall the definition of the hierarchy of primitive recursive sequence functions given in [5].

Definition 2.6

$$l_{0,0} = \mathscr{C}(A \cup \{\Theta^{\mathbb{N}} = \lambda x, y.(y, x), \Delta^{\mathbb{N}} = \lambda x.(x, x)\}),$$

$$l_{i,j+1} = \{f = f_1 \circ f_2 \mid f_1 \in l_{i,j} \text{ and } f_2 \in \mathscr{R}(l_{i,0})\} \quad \text{for } i, j \geq 0,$$

$$l_{i,0} = \bigcup_{j\geq 0} l_{i-1,j} \quad \text{for } i \geq 1.$$

Let us denote the class $l_{i,0}$ by l_i for $i \geq 0$.

Lemma 2.7. (1) *For every* $i, j \geq 0$ *it holds that for every* $f: \mathbb{N}^r \to \mathbb{N}^s \in l_{i,j}$ *there exists an* $f^{\Sigma}: \Sigma^{*^r} \to \Sigma^{*^s} \in L_{i,j}$ *such that* $f(|u|) = |f^{\Sigma}(u)|$ *for every* $u \in \Sigma^{*^r}$.

(2) *For every* $i \geq 1$ *and* $j \geq 0$ *it holds that for every* $f^{\Sigma}: \Sigma^{*^r} \to \Sigma^{*^s} \in L_{i,j}$ *there exists a nondecreasing function* $f: \mathbb{N}^r \to \mathbb{N}^s \in l_{i,j}$ *such that* $|f^{\Sigma}(u)| \leq f(|u|)$ *for every* $u \in \Sigma^{*^r}$.

Proof. (1) The proof is straightforward by induction on the structure of $l_{i,j}$.

Suppose that the claim holds for $f \in l_{i,j}$. If a function $f \in l_{i,j+1}$, then $f = f_1 \circ f_2$ where $f_1 \in l_{i,j}$ and $f_2 = g^{R}$ with $g \in l_i$. By induction hypothesis there exists an $f_1^{\Sigma} \in L_{i,j}$ and $g^{\Sigma} \in L_i$ such that $f_1(|u|) = |f_1^{\Sigma}(u)|$ and $g(|u|) = |g^{\Sigma}(u)|$. Then

$$\begin{aligned} f(|u|) &= (f_1 \circ g^{R})(|u|) = g^{R}(f_1(|u|)) = g^{R}(|f_1^{\Sigma}(u)|) = |(g_1^{\Sigma}, \ldots, g_n^{\Sigma})^{1}(f_1^{\Sigma}(u))| \\ &= |f_1^{\Sigma} \circ (g_1^{\Sigma}, \ldots, g_n^{\Sigma})^{1}(u)| \end{aligned}$$

where $g^{\Sigma} = g_1^{\Sigma} = \cdots = g_n^{\Sigma}$. Therefore, $f^{\Sigma} = f_1^{\Sigma} \circ (g_1^{\Sigma}, \ldots, g_n^{\Sigma})^{\mathrm{I}} \in l_{i,j+1}$ is the wanted function.

(2) This claim can easily be proved by induction on the structure of $L_{i,j}$: Suppose it holds for $L_{i,j}$ and consider a function $f^{\Sigma} \in L_{i,j+1}$ where $f^{\Sigma} = f_1^{\Sigma} \circ (g_1^{\Sigma}, \ldots, g_n^{\Sigma})^{\mathrm{I}}$ with $f_1^{\Sigma} : \Sigma^{*^r} \to \Sigma^{*^{s-1}} \in L_{i,j}$ and $g_k^{\Sigma} : \Sigma^{*^s} \to \Sigma^{*^s} \in L_i$ for $1 \leq k \leq n$. By inductive hypothesis there exist nondecreasing functions $f_1 \in l_{i,j}$ and $g_1, \ldots, g_n \in l_i$ such that $|f_1^{\Sigma}(u)| \leq f_1(|u|)$ and $|g_k^{\Sigma}(u)| \leq g_k(|u|)$, $1 \leq k \leq n$.

Let

$$g = \lambda x_1, \ldots, x_s.\left(\sum_{k=1}^{n} g_k(x_1, \ldots, x_s)\right) \quad \text{for } x_1, \ldots, x_s \in \mathbb{N}$$

where $x_1, \ldots, x_s + y_1, \ldots, y_s = x_1 + y_1, \ldots, x_s + y_s$. The function $g \in l_i$ and it holds that $|g_k(u)| \leq g(|u|)$ for $1 \leq k \leq n$; then the function $f = f_1 \circ g^{\mathrm{R}} \in l_{i,j+1}$ is the wanted function. In fact

$$|f^{\Sigma}(u)| = |(g_1^{\Sigma}, \ldots, g_n^{\Sigma})^{\mathrm{I}}(f_1^{\Sigma}(u))| \leq g^{\mathrm{R}}(|f_1(u)|) \leq (f_1 \circ g^{\mathrm{R}})(|u|).$$

Proposition 2.8. *There exists a bijective function $\Phi : S \to WS_1$ such that $\Phi(l_{i,j}) = L_{i,j}$.*

Proof. The proof follows by Proposition 1.7 and by the definition of the classes.

From now on we will identify the functions of S with the functions of WS on an alphabet of cardinality one.

3. LOOP programs computing wordsequence functions

Primitive recursive wordsequence functions give a semantics to the language of LOOP programs on words defined by Ausiello and Moscarini in [2].

In this section we prove that the chains of classes $L_{i,j}^{\rightarrow\mathrm{L}}$, $L_{i,j}^{\leftarrow\mathrm{R}}$, $L_{i,j}^{\rightarrow\mathrm{R}}$ and $L_{i,j}^{\leftarrow\mathrm{L}}$ are hierarchies of functions. Note that the strictness of the containment of $L_{0,j}$ in $L_{0,j+1}$ holds also if we consider the subclasses of $L_{0,j}$ containing only functions of type $f : \Sigma^* \to \Sigma^*$ for $\mathrm{card}(\Sigma) \geq 2$. In the numerical case, instead, the subclass of $l_{0,j}$ containing only functions of type $f : \mathbb{N}^r \to \mathbb{N}$ is contained in $l_{0,r+1}$, for every j (see [9]).

We will state some results in comparing the classes $L_{i,j}^{\rightarrow\mathrm{L}}$, $L_{i,j}^{\leftarrow\mathrm{R}}$, $L_{i,j}^{\rightarrow\mathrm{R}}$ and $L_{i,j}^{\leftarrow\mathrm{L}}$. The hierarchies coincide for $i \geq 2$ and $j \geq 0$; moreover, the classes $L_{i,j}^{\rightarrow\mathrm{L}}$, $L_{i,j}^{\leftarrow\mathrm{R}}$, $L_{i,j}^{\rightarrow\mathrm{R}}$ and $L_{i,j}^{\leftarrow\mathrm{L}}$ turn out to be complexity classes for $i \geq 2$ and $j \geq 0$, like the numerical case. Some weaker results about the computing time function are obtained for the classes $L_{1,j}^{\rightarrow\mathrm{L}}$, $L_{1,j}^{\leftarrow\mathrm{R}}$, $L_{1,j}^{\rightarrow\mathrm{R}}$ and $L_{1,j}^{\leftarrow\mathrm{L}}$, $j \geq 0$.

Let us define the LOOP programs on words on $\Sigma = \{a_1, \ldots, a_n\}$ (see [2]). Let X, Y be names for registers which can contain an arbitrary word on Σ^*.

Let us consider the following instruction:

(a) $X \leftarrow e$, the clear instruction,

(b) $X \leftarrow Y$, the copy instruction,
(c) $X \leftarrow Xa_i$, the R-append instruction, for $a_i \in \Sigma$,
and the following loop control structures:

(d) LOOP$^{\rightarrow}$ X the ℓ-loop instruction,
1. I_1;
⋮
n. I_n;
END

where $I_1, \ldots, I_n$ are lists of instructions of type (a), (b), (c) or (d).

(d′) LOOP$^{\leftarrow}$ X the $\imath$-loop instruction,
1. I_1;
⋮
n. I_n;
END

where $I_1, \ldots, I_n$ are lists of instructions of type (a), (b), (c) or (d′).

The loop control structures (d) [(d′)] are interpreted by the following informal program:

Step 1. Transfer the content of X in the register CONTROL.
Step 2. while CONTROL $\neq e$ execute I_i if the leftmost (rightmost) character of CONTROL is a_i; erase the leftmost (rightmost) character of CONTROL.

Definition 3.1. An Rℓ-LOOP program (R$\imath$-LOOP program) on an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ has the following form:

IN s; I_0;...;I_m; OUT t,

where s is a possibly empty list of names of registers, I_j, for $0 \leq j \leq m$, $m \geq 0$ is an instruction of type (a), (b), (c) or (d) [(a), (b), (c) or (d′)] and t is a nonempty list of names of registers either occurring in the input list or introduced in the program by an instruction of type (a).

Consider now the instruction

(c′) $X \leftarrow a_i X$, the L-append instruction, for $a_i \in \Sigma$.

The Lℓ-LOOP and L$\imath$-LOOP programs are defined as the Rℓ-LOOP and R$\imath$-LOOP programs by considering (c′) instead of (c) wherever (c) appears in the above definitions.

We will refer to LOOP programs for Lℓ-LOOP, L$\imath$-LOOP, Rℓ-LOOP, R$\imath$-LOOP programs.

Definition 3.2. A function $f: \Sigma^{*^p} \rightarrow \Sigma^{*^q} \in WS$ is computed by a LOOP program P with input register list s and output register list t if before the execution the input registers of P contain $x_1, \ldots, x_p \in \Sigma^*$ (and the other registers are empty) and after the execution the output registers of P contain the sequence $f(x_1, \ldots, x_p) \in \Sigma^{*^q}$.

Definition 3.3. Let ℓ-loop(1) be a loop instruction $\text{LOOP}^{\rightarrow}$ X 1. P_1; ... ; n. P_n; END where each P_i is a list of clear, copy and R-append or L-append instructions; let ℓ-loop(i), for $i > 1$, be a loop instruction $\text{LOOP}^{\rightarrow}$ X 1. P_1; ... ; n. P_n; END where each P_i is a list of clear, copy, R-append or L-append and ℓ-loop($i-1$) instructions.

The loop instruction r-loop(i) is defined analogously.

Definition 3.4. Let $M_0^{\leftarrow L}$ be the class of Lr-LOOP programs obtained by using only clear, copy and L-append instructions. Let $M_{i,j}^{\leftarrow L}$ be the class of Lr-LOOP programs obtained by using just j r-loop($i+1$) instructions besides clear, copy, L-append and r-loop(k) instructions with $1 \leqslant k \leqslant i$.

Let us use $M_i^{\leftarrow L}$ to denote the class $M_{i,0}^{\leftarrow L}$, $i \geqslant 0$.

The class $M_{i,j}^{\leftarrow R}$ is obtained analogously by replacing L-append with R-append in the above definition. The class $M_{i,j}^{\rightarrow L}$ ($M_{i,j}^{\rightarrow R}$) is obtained as $M_{i,j}^{\leftarrow L}$ ($M_{i,j}^{\leftarrow R}$) by considering r-loop instead of ℓ-loop instructions.

Theorem 3.5. *$L_{i,j}^{\leftarrow L}$, $L_{i,j}^{\rightarrow L}$, $L_{i,j}^{\leftarrow R}$, $L_{i,j}^{\rightarrow R}$ are the classes of functions computed by programs belonging to $M_{i,j}^{\leftarrow L}$, $M_{i,j}^{\rightarrow L}$, $M_{i,j}^{\leftarrow R}$, $M_{i,j}^{\rightarrow R}$ respectively, for $i, j \geqslant 0$.*

Proof. The proof is tedious but straightforward.

Now we can state that each chain of classes $L_{i,j}$ forms a hierarchy.

The proof of the strict containment of $L_{i,j}$ in $L_{i,j+1}$ for $i > 0$ exploits properties of growth of the functions with respect to the lexicographic order on sequences of lengths of words. The proof is different in the case $i = 0$ because all the functions in L_1 exhibit the same behaviour with respect to the growth.

Consider the function $c^k : \Sigma^* \to \Sigma^*$ with card$(\Sigma) > 2$ and $k > 2$ such that $c^k = \lambda x.(x^k)$ for $x \in \Sigma^*$. For every LOOP program computing c^k, by scanning sequentially the program, we can construct a representation of it in form of binary tree:

'*Initialization*': create a node labelled x;

'*Step*': **for** the next loop instruction giving as a result the concatenation of $u_j \in \{x\}^*$ and $v_j \in \{x\}^*$ for $1 \leqslant j \leqslant p$ and such that nodes n_j and m_j labelled u_j and v_j have been created **do**

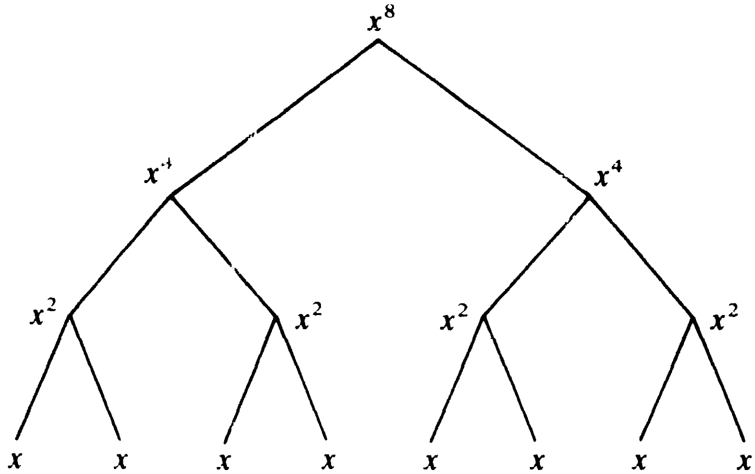
1. **if** $n_j \neq m_j$ and n_j and m_j have not a father **then** create a new node q_j having n_j as left son and m_j as right son and label it u_jv_j;
2. **if** $n_j \neq m_j$ and n_j (or m_j) has a father **then** create a new tree having a new node n_j' (or m_j') as root, with label u_j (or v_j), isomorphic and with the same labels as the subtree of root n_j (or m_j). Moreover, create a new node q_j having n_j' and m_j (or n_j and m_j', or n_j' and m_j') as left son and right son respectively, and label q_j with u_jv_j;

3. **if** $n_j = m_j$ **then** create a new subtree having a new node n_j', with label $u_j = v_j$, as root, isomorphic and with the same labels as the subtree whose root is n_j. Moreover, **if** n_j has not a father **then** create a node q_j having n_j and n_j' as sons and labelled $u_j v_j$, **otherwise** apply rule 2. of Step.

Definition 3.6. The tree corresponding to a program P computing c^k is a tree obtained from P by initialization and by repeating the step until all the program has been scanned and its root is labelled x^k.

Example 3.7. Consider the following program computing c^8 and the corresponding tree:

```
IN X;
LOOP→ X;
1. X ← Xa1;
   ⋮
n. X ← Xan;
END
LOOP→ X;
1. X ← Xa1;
   ⋮
n. X ← Xan;
END
LOOP→ X;
1. X ← Xa1;
   ⋮
n. X ← Xan;
END
OUT X
```



Lemma 3.8. *For every program computing c^k the corresponding tree has k leaves.*

Proof. For every program P we can state that at every step of the construction of the corresponding tree the j subtrees already obtained have roots labelled x^{m_i} for $1 \leq i \leq j$ and m_i leaves. The trees obtained by initialization and one execution of the step of the construction satisfy the claim trivially. Suppose the claim holds for the trees obtained by p repetitions of the step of the construction. In the $(p+1)$st execution of the step we possibly create new trees isomorphic to and equally labelled as the already created subtrees and the claim holds trivially for these new trees. Then, in the same step, we create new nodes such that each one of these nodes has sons which are labelled x^{m_i} and x^{m_j} respectively and are roots of subtrees having m_i and m_j leaves respectively. Thus the new trees obtained have roots labelled $x^{m_i+m_j}$ and have $m_i + m_j$ leaves.

Lemma 3.9. *For every program P computing c^k the number l of loop instructions occurring in P is greater than or equal to the height h of the corresponding tree.*

Proof. For every program P we can state that at every step of the construction of the corresponding tree the j trees already obtained by initialization and one execution of the step of the construction satisfy the claim trivially. Suppose the claim is true for the trees obtained by p executions of the step. Let h be the maximum height of such trees. The $(p+1)$st execution of the step provides trees having height at most $h+1$ and then the claim holds.

Lemma 3.10. *The function c^{2^k} belongs to $L_{0,k} - L_{0,k-1}$ for every $k \geq 1$.*

Proof. It is easy to see that the function $c^{2^k} \in L_{0,k}$ for every $k \geq 1$. Let $P \in M_{0,q}$ be a program computing c^{2^k}. The corresponding tree has 2^k leaves, by Lemma 3.8, and height $h \geq k$. By Lemma 3.9, $q \geq h$, then $c^{2^k} \notin L_{0,k-1}$.

Theorem 3.11. *$L_{0,j} \subsetneq L_{0,j+1}$ for every $j \geq 0$.*

Proof. The containment immediately follows from the definition of the classes. The strictness follows by Lemma 3.10 for card$(\Sigma) \geq 2$. For card$(\Sigma) = 1$ the result has been proved in [5].

Let us define the following strictly increasing functions $F_j^i : \mathbb{N}^2 \to \mathbb{N}$ for $i \geq 1$ and $j \geq 0$:

$$F_0^1 = h_1^{\mathrm{c}} \circ^{\mathrm{c}} (h_1 \circ S) \circ h_1^{\mathrm{R}} = \lambda x, y.(2^{2x}(2y+1)) \quad \text{if } h_1 = \lambda x.(2x),$$

$$F_j^1 = F_{j-1}^1 \circ \Delta \circ h_1^{\mathrm{R}} \quad \text{for } j \geq 1.$$

Let $h_{i+1} = \lambda x.F_0^i(x)$ for $i \geq 1$:

$$F_0^i = h_i^{\mathrm{c}} \circ^{\mathrm{c}} (h_i \circ s) \circ h_i^{\mathrm{R}} \quad \text{for } i \geq 1,$$

$$F_j^i = F_{j-1}^i \circ \Delta \circ h_i^{\mathrm{R}} \quad \text{for } i \geq 1 \text{ and } j \geq 1.$$

Let $H_j^i = \lambda x.F_j^i(x,x)$ for $i \geq 1$ and $j \geq 0$.

It has been stated in [5] that $F_j^i \in l_{i,j+1}$ and $H_j^i \in l_{i,j+1}$ for every $i \geq 1$, $j \geq 0$.

Let now $\bar{F}_j^i : \Sigma^{*^2} \to \Sigma^*$ be the function such that $\bar{F}_j^i(x,y) = F_j^i(|x|,|y|)$ for $i \geq 1$, $j \geq 0$. It results that $\bar{F}_j^i \in L_{i,j+1}$.

If $u = x_1, \ldots, x_r$, then we use $\|u\|$ to denote $\max\{|x_1|, \ldots, |x_r|\}$.

Lemma 3.12. *For every wordfunction $f : \Sigma^{*^r} \to \Sigma^{*^s} \in L_{i,j}$ with $i \geq 1$ and $j \geq 0$, there exists a $w \in \Sigma^*$ such that $|f(u)| \leq \bar{F}_j^i(w, x_k)$ for every $u = x_1, \ldots, x_r$, where x_k is a word such that $|x_k| = \|u\|$.*

Proof. Let $f^{\Sigma} \in L_{i,j}$ with $i \geq 1$, $j \geq 0$. By Lemma 2.7(2) there exists a function $f \in l_{i,j}$ such that $|f^{\Sigma}(u)| \leq f(|u|)$ and by [5] there exists an $m \in \mathbb{N}$ such that $f(|u|) \leq F_j^i(m, \|u\|)$. Then $|f^{\Sigma}(u)| \leq \bar{F}_j^i(a_1^m, x_k)$ where $|x_k| = \|u\|$.

Now we are able to prove the following theorem.

Theorem 3.13. *For every $i \geqslant 1$, $j \geqslant 0$, $L_{i,j} \subsetneq L_{i,j+1}$.*

Proof. The containment holds by the definition of the classes.

Let $\bar{H}_j^i = \lambda x.\bar{F}_j^i(x, x)$. By Lemma 3.12, for every $f: \Sigma^* \to \Sigma^* \in L_{i,j}$ with $i \geqslant 1$, $j \geqslant 0$ there exists a $w \in \Sigma^*$ such that $|f(x)| \leqslant \bar{F}_j^i(w, x)$. Then for every x such that $|x| \geqslant |w|$, $|f(x)| \leqslant \bar{H}_j^i(x)$ as $\bar{F}_j^i$ is a strictly increasing function with respect to the length of its arguments. And then $\bar{H}_j^i \in L_{i,j+1} - L_{i,j}$ for every $i \geqslant 1$, $j \geqslant 0$.

Let us now state a simultaneity result.

Theorem 3.14. *For $i \geqslant 1$ and $j \geqslant 0$, $L_{i,j}$ is closed with respect to the juxtaposition operator.*

Proof. For $\text{card}(\Sigma) = 1$ the proof has been given in [6]; for $\text{card}(\Sigma) \geqslant 2$ the proof is analogous. Consider a function $h = f\,\hat{}\,g$, with $f: \Sigma^{*r} \to \Sigma^{*s}$ and $g: \Sigma^{*r} \to \Sigma^{*t}$ belonging to $L_{i,j}$. By definition it holds that f and g can be computed by programs consisting of j successive loop instructions of depth of nesting i. Let $X_1, \ldots, X_j$, $Y_1, \ldots, Y_j$ be the control registers of the loops. To compute h we can construct a program consisting of j successive loop instructions of depth of nesting i and controlled by $Z_1 = \text{CONC}(X_1, Y_1), \ldots, Z_j = \text{CONC}(X_j, Y_j)$. As the concatenation of words is a function belonging to L_1, the function $h = f\,\hat{}\,g$ belongs to $L_{i,j}$.

Let us now introduce the computing time function.

Definition 3.15. Let P be an $L\ell$-LOOP program IN $X_1, \ldots, X_r; I$; OUT $Y_1, \ldots, Y_s$, let $Z_1, \ldots, Z_p$ be the list of new register names introduced by I and let $q = p + r$. Then $t_P: \Sigma^{*r} \to \Sigma^*$ the computing time function of P is ${}^{c^r}E \circ {}^{c^{r+1}}E \circ \cdots \circ {}^{c^q}E \circ t \circ U_{q+1}^{q+1}$ where $t: \Sigma^{*q+1} \to \Sigma^{*q+1}$ is the stepcounter function of I defined as follows:

- **if** I is the empty sequence **then** $t = U_1^{q+1}\,\hat{}\cdots\hat{}\,U_{q+1}^{q+1}$,
- **if** $I = X_i \leftarrow a_j X_i$ **then** $t = {}^{c^{i-1}}S_j^{c^{q-(i+1)}} \circ {}^{c^q}S_1$,
- **if** $I = X_i \leftarrow e$ **then** $t = {}^{c^{i-1}}(E^1)^{c^{q-(i+1)}} \circ {}^{c^q}S_1$,
- **if** $I = X_i \leftarrow X_j$ **then** $i = T_{i,j}^{q+1} \circ {}^{c^q}S_1$,
- **if** $I = I_1; I_2$ and t_i is the stepcounter function of I_i **then** $t = t_1 \circ t_2$,
- **if** $I = \text{LOOP}^{\leftarrow}\ X_i\ 1.\ I_1'; \ldots; n.\ I_n'$; END and t_j is the stepcounter function for the list of instructions I_j' **then**

$$t = \Delta^{c^q} \circ (t_1 \circ {}^{c^q}(S_1^3), \ldots, t_n \circ {}^{c^q}(S_1^3))^{1-} \circ {}^{c^q}(S_1^2) \quad \text{if } i = 1$$

and

$$t = {}^{c^{i-1}}\Delta^{c^{q-(i+1)}} \circ \Theta_i^{q+1} \circ (t_1 \circ {}^{c^q}(S_1^3), \ldots, t_n \circ {}^{c^q}(S_1^3))^{1\leftarrow} \circ {}^{c^r}(S_1^2) \quad \text{if } i > 1.$$

If P is an L$\imath$, Rℓ, R$\imath$-LOOP program, then the computing time function is defined analogously.

Proposition 3.16. *If $P \in M_{i,j}$, then $t_P \in L_{i,j}$.*

Proof. The proof immediately follows from the definitions.

We will also write t_f for t_P if f is the function computed by program P.

The hierarchies are compared in the following lemmata. Some results are stated for two of the four hierarchies and by Lemma 2.5 they hold for the two dual hierarchies.

Lemma 3.17. $L_0^{\to L} \supset\subset L_0^{\leftarrow R}$.

Proof. The proof obviously follows from the definition.

Lemma 3.18. $L_{0,j}^{\to R} \subsetneq L_{0,2j}^{\leftarrow R}$.

Proof. If the function $f: \Sigma^{*^{r+1}} \to \Sigma^{*^s} \in L_{0,j}^{\to R}$ then $f = f_0 \circ f_1 \circ \cdots \circ f_j$ where

$$f_0: \Sigma^{*^{r+1}} \to \Sigma^{*^{p_1+1}} \in L_0^{\to R},$$

$$f_i = (g_1^i, \ldots, g_n^i)^{1\to}: \Sigma^{*^{p_i+1}} \to \Sigma^{*^{p_{i+1}+1}} \in \mathcal{J}^{\to}(L_0^{\to R})$$

and $p_{j+1} + 1 = s$.

Then $f = f_0 \circ \mathrm{rev}^{c^{p_1}} \circ f_1' \circ \mathrm{rev}^{c^{p_2}} \circ \cdots \circ \mathrm{rev}^{c^{p_j}} \circ f_j'$ where $f_i' = (g_1^i, \ldots, g_n^i)^{1\leftarrow} \in \mathcal{J}^{\leftarrow}(L_0^{\leftarrow R})$. As $\mathrm{rev} \in L_{0,1}^{\leftarrow R}$, by Lemma 2.3 it holds that $f \in L_{0,2j}^{\leftarrow R}$. As $\mathrm{rev} \in L_{0,j}^{\leftarrow R} - L_{0,j}^{\to R}$ for every j, the strictness of the containment holds.

Lemma 3.19. *If $f: \Sigma^{*^r} \to \Sigma^{*^s} \in L_{0,j}^{\to L}$, then $f \in L_{0,j+r+s}^{\leftarrow R}$.*

Proof. Consider $f: \Sigma^{*^r} \to \Sigma^{*^s} \in L_{0,j}^{\to L}$ and the function $\Phi: WS \to WS$, defined in Lemma 2.5. Then $f = \mathrm{rev}^r \circ \Phi(f)) \circ \mathrm{rev}^s$: as $\Phi(f) \in L_{0,j}^{\leftarrow R}$ and $\mathrm{rev}^n \in L_{0,n}^{\leftarrow R}$ for $n \geq 1$, then $f \in L_{0,j+r+s}^{\leftarrow R}$.

Lemma 3.20. $L_1^{\to R} \subsetneq L_1^{\leftarrow R}$.

Proof. By Lemma 3.18 (see also [2]).

Lemma 3.21. $L_{1,j}^{\to R} \subsetneq L_{1,j}^{\leftarrow R}$ *for every $j \geq 1$.*

Proof. Let $f:\Sigma^{*^r}\to\Sigma^{*^s}\in L_{1,1}^{\to R}$; then $f=f_1\circ(g_1,\ldots,g_n)^{1\to}$ where $f_1,g_1,\ldots,g_n\in L_1^{\to R}$. By Lemma 3.20, $f_1,g_1,\ldots,g_n\in L_1^{\leftarrow R}$. As $f=f_1\circ\mathrm{rev}^{c^s}\circ(g_1,\ldots,g_n)^{1\leftarrow}$ and $f_1\circ\mathrm{rev}^{c^s}\in L_1^{\leftarrow R}$, we have $f\in L_{1,1}^{\leftarrow R}$. The inductive step is proved analogously.

Lemma 3.22. $L_{1,j}^{\leftarrow R}\subsetneq L_{1,j+1}^{\to R}$ *for every* $j\geq 0$.

Proof. If $f:\Sigma^{*^r}\to\Sigma^{*^s}\in L_{1,j}^{\leftarrow R}$, then the computing time function $t_f\in L_{1,j}^{\leftarrow R}$. By Lemma 3.12 there exists a $w\in\Sigma^*$ such that, for every $u=x_1,\ldots,x_r$, $|t_f(u)|<\bar{F}_j^1(w,x_k)$ where $|x_k|=\|u\|$. We can write a loop program $P\in M_{1,1}^{\to R}$ computing a function $g:\Sigma^*\to\Sigma^{*^s}$ such that $f(x_1,\ldots,x_r)=g(\bar{F}_j^1(w,\rho^r(x_1,\ldots,x_r)))$ where $\rho^r:\Sigma^{*^r}\to\Sigma^*$ is the function such that $\rho^r(x_1,\ldots,x_r)=x_1\ldots x_r$. As $\lambda x.\bar{F}_j^1(w,x)\in L_{1,j}$ and $\rho^r\in L_1$, we have $f\in L_{1,j+1}^{\to R}$.

The strict containment of the class $L_{1,j}^{\leftarrow R}$ in $L_{1,j+1}^{\to R}$ is proved by considering the function $\bar{H}_j^1\in L_{1,j+1}-L_{1,j}$.

Lemma 3.23. $L_{1,j}^{\to L}=L_{1,j}^{\leftarrow R}$ *for* $j\geq 0$.

Proof. The proof follows by induction on j.

For $j=0$, the claim holds by Lemma 3.19 and its dual result (see also [2]).

Let $L_{1,j}^{\to L}=L_{1,j}^{\leftarrow R}$ and $f:\Sigma^{*^r}\to\Sigma^{*^s}\in L_{1,j+1}^{\to L}$. By definition, $f=f_1\circ(g_1,\ldots,g_n)^{1\to}$ where $f_1\in L_{1,j}^{\to L}$ and $g_h\in L_1^{\to L}$ for $1\leq h\leq n$. By induction hypothesis, $f_1\in L_{1,j}^{\leftarrow R}$ and $g_h\in L_1^{\leftarrow R}$. As $f=f_1\circ\mathrm{rev}^{c^s}\circ(g_1,\ldots,g_n)^{1\leftarrow}$, $f_1\circ\mathrm{rev}^{c^s}\in L_{1,j}^{\leftarrow R}$, we have $f\in L_{1,j+1}^{\leftarrow R}$. The inverse inclusion is proved analogously.

Theorem 3.24. $L_2^{\to R}=L_2^{\leftarrow R}=L_2^{\to L}=L_2^{\leftarrow L}$.

Proof. The proof follows from Lemmas 3.21, 3.22 and 3.23 (see also [2]).

The diagram, pictured in Fig. 1, summarizes the results of Lemmas 3.17–3.23. In Fig. 1, $A\to B$ means that class A is strictly enclosed in class B, $A\longrightarrow B$ means that class A is enclosed in class B.

Theorem 3.25. *The classes* $L_{i,j}$ *with* $i\geq 2$, $j\geq 0$ *are closed with respect to the computing time function.*

Proof. By Proposition 3.16 we only have to prove that if $t_f\in L_{i,j}$, then $f\in L_{i,j}$. Let us prove the thesis for $L_{i,j}^{\leftarrow R}$. Let $f:\Sigma^{*^r}\to\Sigma^{*^s}$. If $t_f:\Sigma^{*^r}\to\Sigma^*\in L_{i,j}^{\leftarrow R}$, then there exists a $w\in\Sigma^*$ such that, for every $u=x_1,\ldots,x_r$ belonging to Σ^{*^r}, $|t_f(u)|<\bar{F}_j^i(w,x_k)$ where $|x_k|=\|u\|$. We can give a program P computing $g:\Sigma^*\to\Sigma^{*^s}\in L_{1,1}^{\leftarrow R}$ such that $f(x_1,\ldots,x_r)=g(\bar{F}_j^i(w,x_k))$. As $\lambda x.\bar{F}_j^i(w,x_k)\in L_{i,j}^{\leftarrow R}$ and $i\geq 2$, we have $f\in L_{i,j}^{\leftarrow R}$.

Theorem 3.26. *Let* $f:\Sigma^{*^r}\to\Sigma^{*^s}\in L_{1,k}^{\leftarrow R}$, $k\geq 0$. *If there exists a* $j<k$ *such that* $t_f\in L_{1,j}^{\leftarrow R}$, *then* $f\in L_{1,j+1}^{\leftarrow R}$.

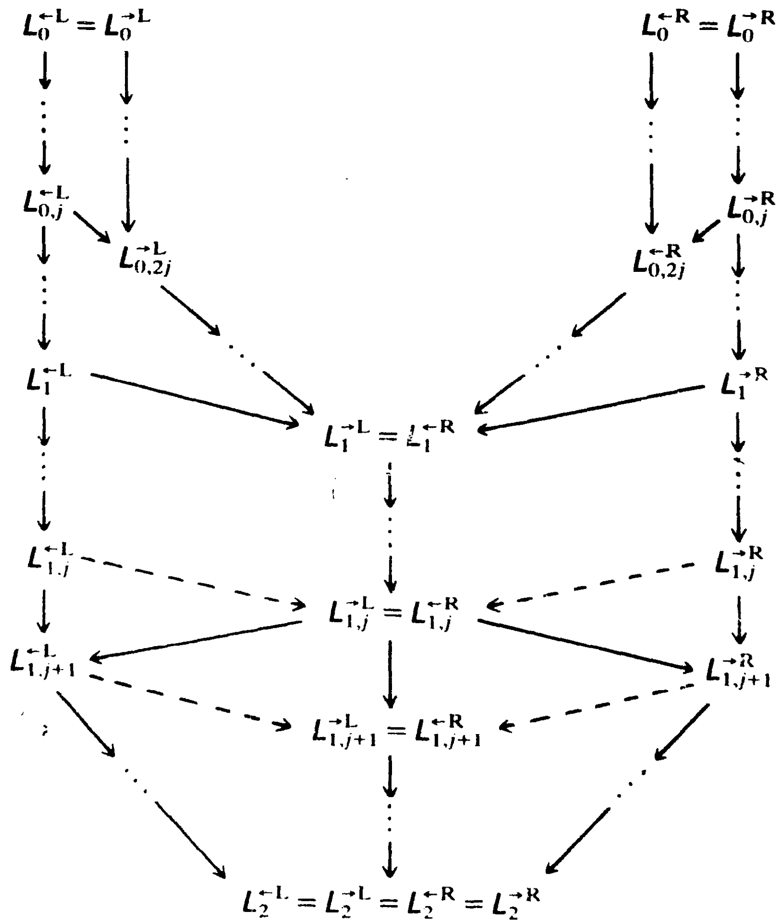


Fig. 1.

Proof. If $t_f : \Sigma^{*r} \to \Sigma^* \in L_{1,j}^{\leftarrow R}$, then there exists a $w \in \Sigma^*$ such that, for every $u = x_1, \ldots, x_r \in \Sigma^{*r}$, $|t_f(u)| < \bar{F}_j^1(w, x_k)$ where $|x_k| = \|u\|$. Then $|t_f(u)| < \bar{F}_j^1(w, \rho^r(u))$. It holds that $f(u) = g(\bar{F}_j^1(w, \rho^r(u)))$ where $g: \Sigma^* \to \Sigma^{*s}$ is the function computed by the program $P \in M_{1,1}^{\leftarrow R}$ defined in the previous theorem. As $\lambda u.({}^c\rho^r \circ \bar{F}_j^1)(w, u) \in L_{1,j}^{\leftarrow R}$, then $f \in L_{1,j+1}^{\leftarrow R}$.

Theorem 3.27. *Let $f: \Sigma^{*r} \to \Sigma^{*s} \in L_{1,k}^{\to R}$, $k \geq 0$. If there exists a $j < k$ such that $t_f \in L_{1,j}^{\to R}$, then $f \in L_{1,j+2}^{\to R}$.*

Proof. Let $f: \Sigma^{*r} \to \Sigma^{*s} \in L_{1,k}^{\to R}$ and $t_f \in L_{1,j}^{\to R}$ with $j < k$. By Lemma 3.21, $f \in L_{1,k}^{\leftarrow R}$ and $t_f \in L_{1,j}^{\leftarrow R}$. By Theorem 3.26, $f \in L_{1,j+1}^{\leftarrow R}$ and by Lemma 3.22, $f \in L_{1,j+2}^{\to R}$.

We now report some results obtained by comparing the loop hierarchies of functions $f: \Sigma^{*r} \to \Sigma^*$ with Axt and Grzegorczyk hierarchies defined in [12].

Let us recall the definitions of these hierarchies.

Let

$$B^1 = \{S_i = \lambda x.(a_i x),\ 1 \leq i \leq n,\ E^0 = (e),\ E^1 = \lambda x.(e),$$

$$U_j^r = \lambda x_1, \ldots, x_r.(x_j)\ r \geq 1,\ 1 \leq j \leq r\}.$$

Definition 3.28. The Axt hierarchy R_j $(j \geq 0)$ is defined by induction: R_0 is the smallest class containing B^1 and closed with respect to substitution; R_{j+1} is the smallest class closed with respect to substitution, containing R_j and the functions obtained by one application of primitive recursion on R_j.

We recall the definition of the generalized Ackermann functions $A_j : \Sigma^{*^2} \to \Sigma^*$ belonging to W for every $j \geq 0$:

$$A_0(x, y) = S_1(y), \qquad A_1(x, e) = x, \qquad A_2(x, e) = e,$$

$$A_j(x, e) = a_1 \quad \text{for } j \geq 3 \qquad \text{and}$$

$$A_{j+1}(x, a_i y) = A_j(x, A_{j+1}(x, y)) \quad \text{for } j \geq 0 \text{ and } 1 \leq i \leq n.$$

Definition 3.29. The Grzegorczyk hierarchy E_j $(j \geq 0)$ is defined as follows: E_j is the smallest class of primitive recursive wordfunctions containing $B^1 \cup \{A_j\}$ and closed with respect to substitution and the limited recursion operator, where a function f is said to be obtained by limited recursion from g, h_i and d if f is obtained from g and h_i by primitive recursion and $|f(u)| \leq |d(u)|$.

Let $L'_{i,j}$ be the subclass of $L_{i,j}$ containing only wordsequence functions $f : \Sigma^{*^r} \to \Sigma^*$.

Note that for the loop hierarchy L_j $(j \geq 0)$ defined in [12] it holds that $L_j = L_j'^{\leftarrow \mathrm{L}}$ for every $j \geq 0$.

Theorem 3.30. *$L_0'^{\leftarrow \mathrm{L}} = R_0 \subsetneq E_0$, $R_1 \subsetneq L_1'^{\leftarrow \mathrm{L}} \subsetneq E_1$, $R_1 \supset\subset E_0 \supset\subset L_1'^{\leftarrow \mathrm{L}}$, $E_2 \subsetneq L_2'^{\leftarrow \mathrm{L}}$, $L_j'^{\leftarrow \mathrm{L}} = R_j = E_{j+1}$ for every $j \geq 2$.*

Proof. For the proof, see [11, 12, 15].

Theorem 3.31. *$L_1'^{\to \mathrm{L}} \supset\subset E_0$, $L_1'^{\to \mathrm{L}} \subsetneq E_1$.*

Proof. For the proof, see [2].

We can now improve the result $E_2 \subsetneq L_2'^{\leftarrow \mathrm{L}}$ of Theorem 3.30.

Theorem 3.32. *$E_2 \subsetneq L_{1,2}'^{\leftarrow \mathrm{L}}$.*

Proof. It is easy to see that a loop program P simulating a deterministic Turing machine with r input tapes, m storage tapes and one output tape can be given in $M_{1,1}^{\leftarrow \mathrm{L}}$ (see also [12]). Let $f : \Sigma^{*^r} \to \Sigma^* \in E_2$, the computing time function of a Turing machine computing f is bounded by $\lambda u.A_3(\rho^r(u), w)$ for a suitable $w \in \Sigma^*$. As $\lambda u.A_3(\rho^r(u), w) \in L_{1,1}'^{\leftarrow \mathrm{L}}$, a loop program computing f can be given in $M_{1,2}^{\leftarrow \mathrm{L}}$. Then $E_2 \subseteq L_{1,2}'^{\leftarrow \mathrm{L}}$. As there does not exist a $w \in \Sigma$ such that $|\bar{F}_1(x, y)| \leq |A_3(\rho^2(x, y), w)|$ for every $x, y \in \Sigma^*$, we have $\bar{F}_1 \notin E_2$ and the claim holds.

4. LOOP programs, automata and transducers

In this section we deal with the power of LOOP programs as transducers and as acceptors.

We show that the functions defined by generalized sequential machines are in $L_{0,1}$ and that the functions defined by push-down transducers are in $L_{1,1}$. In both cases we show that the reverse does not hold. Note that Indermark [14] proved that the functions defined by generalized sequential machines and push-down transducers are in E_1. Besides, every function in L_1 can be defined by a two-way finite state transducer.

As regards the power of LOOP programs as acceptors, it holds that the class of regular languages coincides with the class of languages

$$\mathcal{L}_f = \{x \mid f(x) \neq e \text{ and } f \in L_{0,1}\}$$

and that the class of deterministic context free languages is strictly enclosed in the class

$$\mathcal{L}_f = \{x \mid f(x) \neq e \text{ and } f \in L_{1,1}\}.$$

Definition 4.1. A language $\mathcal{L} \subseteq \Sigma^*$ is said to be accepted by a program P if and only if $\mathcal{L} = \{x \mid f_P(x) \neq e\}$ where $f_P : \Sigma'^* \to \Sigma'$, with $\Sigma' \supseteq \Sigma$, is the function computed by P.

Let us now consider the augmented LOOP programs on words defined by Chytil and Jakl [5].

The loop control structure is interpreted in a slightly different way: The register CONTROL, where the content of the control register of the loop instruction is stored, is supposed to be accessible for inspecting, without changing it, through a window moving backwards and forwards.

The set of basic statement is augmented by the instructions

- LEFT, which causes a leftward move of the control register window,
- RIGHT, which causes a rightward move of the control register window.

The set of control statement is augmented by conditional instructions:

$$\text{IF } X \neq e \text{ THEN } S_1; \ldots; S_r \text{ ELSE } S_{r+1}; \ldots; S_t \text{ FI}$$

which causes the execution of the sequence of statements $S_1; \ldots; S_r$ if the register X is not empty and the execution of the statements $S_{r+1}; \ldots; S_t$ otherwise,

$$\text{IF } X \neq e \text{ THEN } S_1; \ldots; S_r \text{ FI}$$

which causes the execution of the statements $S_1; \ldots; S_r$ if the register X is not empty.

Moreover, the loop control structure

LOOP X
1. I_1;
$\vdots$
n. I_n;
END

(where n is the cardinality of the considered alphabet) is interpreted as follows:
- the content w of X is stored in the register CONTROL and the control window is set to the leftmost symbol of the word w,
- while the window displays a symbol of w execute I_i if the displayed symbol is a_i (possible occurrences of LEFT and RIGHT within I_i do not influence the running execution of I_i).

Definition 4.2. Let $M_1^{\leftrightarrow R}$ be the class of augmented LOOP programs IN $X_1, \ldots, X_r; I_1; \ldots; I_m$; OUT $Y_1, \ldots, Y_s$ where I_j $(1 \leq j \leq m)$ is a clear, copy, R-append, LEFT, RIGHT instruction or a conditional instruction or a loop instruction LOOP X 1. $I'_1; \ldots;$ n. I'_n; END where I'_k $(0 \leq k \leq n)$ is a list of copy, clear, R-append, LEFT, RIGHT or conditional instructions. Let $PL_1^{\leftrightarrow R}$ be the class of partial functions computed by programs in $M_1^{\leftrightarrow R}$.

Definition 4.3. A deterministic two-way finite state transducer is a 7-tuple $\mathcal{S}_2 = (K, \Sigma, \Delta, \delta, q_0, ¢, \$)$ where K, Σ and Δ are the finite sets of states, input symbols and output symbols respectively; $q_0 \in K$ is the initial state and ¢ and \$ are endmarkers; $\delta: K \times (\Sigma \cup \{¢, \$\}) \to K \times \Delta^* \times \{-1, +1\}$.

Definition 4.4. An instantaneous description of $\mathcal{S}_2$ is a couple belonging to $H \subseteq (\Sigma \cup \{¢, \$\})^* K (\Sigma \cup \{¢, \$\})^* \times \Delta^*$.

A binary relation $\vdash$ is defined on H such that

$$(xqax', y) \vdash (xaq'x', yy') \quad \text{iff} \quad \delta(q, a) = (q', y', +1),$$

$$(xaqa'x, y) \vdash (xq'aa'x', yy') \quad \text{iff} \quad \delta(q, a') = (q', y', -1),$$

where $x, x' \in (\Sigma \cup \{¢, \$\})^*$, $a, a' \in \Sigma \cup \{¢, \$\}$, $q, q' \in K$ and $y, y' \in \Delta^*$.
Let $\vdash^*$ be the reflexive and transitive closure of $\vdash$.

Definition 4.5. A function $f: \Sigma^* \to \Sigma^*$ is said to be defined by $\mathcal{S}_2$ iff

$$f(x) = \begin{cases} y & \text{if there exists } q \in K \text{ such that } (q_0¢x\$, e) \vdash^* (¢x\$q, y), \\ \text{undefined} & \text{otherwise.} \end{cases}$$

If $f: \Sigma^{*r} \to \Sigma^{*s}$, then $f_c: (\Sigma \cup \{¢'\})^* \to (\Sigma \cup \{\$'\})^*$ is the function such that $f(x_1, \ldots, x_r) = y_1, \ldots, y_s$ iff $f_c(x_1 ¢' \cdots ¢'^{\,r-1} x_r) = y_1 \$' \cdots \$'^{\,s-1} y_s$.

Theorem 4.6. *A function $f \in PL_1^{\leftrightarrow R}$ iff there exists a two-way finite state transducer which defines f_c.*

Proof. For the proof, see [5].

Now we consider $TL_1^{\leftrightarrow R}$, the class of total functions belonging to $PL_1^{\leftrightarrow R}$.

Theorem 4.7. $L_1^{\leftarrow R} \subseteq TL_1^{\leftrightarrow R}$.

Proof. Note that rev $\in TL_1^{\leftrightarrow R}$. Let $f: \Sigma^{*r} \to \Sigma^{*s} \in L_1^{\leftarrow R}$ and $P \in M_1^{\leftarrow R}$ be a program computing f. A program $P' \in M_1^{\leftrightarrow R}$ can be obtained from P by replacing every instruction LOOP$^{\leftarrow}$ X 1. $I_1; \ldots;$ n. I_n; END by the following sequence of instructions:

$X \leftarrow$ REV(X);
LOOP X
1. I_1; RIGHT;
⋮
n. I_n; RIGHT;
END
$X \leftarrow$ REV(X)

where $X \leftarrow$ REV(X) is shorthand for a program in $M_1^{\leftrightarrow}$ which computes the function rev.

Theorem 4.8. *The languages accepted by programs computing functions in $L_{0,1}$ are exactly the regular languages.*

Proof. Let $\mathcal{L} \subseteq \Sigma^*$ be a regular language; let $\mathcal{A} = \{K, \Sigma, \delta, q_0, F\}$ be the complete deterministic automata accepting $\mathcal{L}$, where $K = \{q_0, \ldots, q_s\}$, $\Sigma = \{a_1, \ldots, a_n\}$, $\delta: K \times \Sigma \to K$ and $F = \{q_{i_1}, \ldots, q_{i_t}\} \subseteq K$. The function $f: \Sigma^* \to \Sigma$ such that

$$f(x) = \begin{cases} a_1 & \text{if } x \in \mathcal{L}, \\ e & \text{otherwise,} \end{cases}$$

is computed by the following program:

IN X;
$Q_0, \ldots, Q_s, F_{q_0}, \ldots, F_{q_s} \leftarrow e$;
$Q_{i_1}, \ldots, Q_{i_t} \leftarrow a_1 Q_{i_1}, \ldots, a_1 Q_{i_t}$;
LOOP$^{\leftarrow}$ X
⋮
i. $F_{q_0}, \ldots, F_{q_s} \leftarrow Q_0, \ldots, Q_s$; $Q_0, \ldots, Q_s \leftarrow F_{\delta(q_0, a_i)}, \ldots, F_{\delta(q_s, a_i)}$;
⋮
END
OUT Q_0

By Theorem 3.5, $f \in L_{0,1}^{\leftarrow L}$.

Each register corresponding to a final state is set to a_1 and the register X is read leftwards. If a_i is the scanned symbol, then, corresponding to each rule $\delta(q_j, a_i) = q_h$, the content of Q_h is transferred to Q_j. Hence if X contains $x = a_{i_1} \dots a_{i_k}$, then Q_0 is set to a_1 iff there exist $q_1, \dots, q_k$ such that $\delta(q_j, a_{i_{j+1}}) = q_{j+1}$ and $q_k \in F$.

Consider a program P' obtained from P by replacing the instructions $Q_{i_j} \leftarrow a_1 Q_{i_j}$ with $Q_{i_j} \leftarrow Q_{i_j} a_1$ for $1 \leq j \leq t$. Program P' computes the same function as P; thus, $f \in L_{0,1}^{\leftarrow R}$ as well.

Let $\mathscr{L}$ be a regular language; let $\tilde{\mathscr{L}} = \{y \mid y = \mathrm{rev}(x) \text{ and } x \in \mathscr{L}\}$. As $\tilde{\mathscr{L}}$ is a regular language there exists a function $f \in L_{0,1}^{\leftarrow L} \cap L_{0,1}^{\leftarrow R}$ such that $\tilde{\mathscr{L}} = \{y \mid f(y) \neq e\}$. By Lemma 3.5 the function $\Phi(f) \in L_{0,1}^{\rightarrow L} \cap L_{0,1}^{\rightarrow R}$ and

$$\mathscr{L} = \{x \mid \mathrm{rev}(x) \in \tilde{\mathscr{L}}\} = \{x \mid f(\mathrm{rev}(x)) \neq e\} = \{x \mid \mathrm{rev}(f(\mathrm{rev}(x))) \neq e\} = \{x \mid \Phi(f)(x) \neq e\}.$$

Vice versa, by Theorems 4.6 and 4.7 the languages accepted by programs in M_1 are regular languages.

Definition 4.9. A generalized sequential machine (gsm) is a 6-tuple $\mathscr{S} = (K, \Sigma, \Delta, \delta, \lambda, q_0)$ where K is a finite set of states, Σ and Δ are the input and output alphabet respectively, $q_0 \in K$ is the initial state, δ and λ are functions such that $\delta: K \times (\Sigma \cup \{¢\}) \to K$ and $\lambda: K \times (\Sigma \cup \{¢\}) \to \Delta^*$ where ¢ is an endmarker and $\delta(q, ¢) = q$ for every $q \in K$.

Let us denote by δ and λ the extensions of δ and λ to the domain $K \times (\Sigma^* \cup \{¢\})$, as usually.

Let $f_{\mathscr{S}}: \Sigma^* \to \Delta^*$ be the function such that $f_{\mathscr{S}}(x) = \lambda(q_0, x)$.

Let $G = \{f \mid \text{there exists a gsm } \mathscr{S} \text{ such that } f = f_{\mathscr{S}}\}$.

Theorem 4.10. $G \subsetneq L_{0,1}^{\leftarrow L}$.

Proof. Let $\mathscr{S} = (K, \Sigma, \Sigma, \delta, \lambda, q_0)$ be a gsm with $K = \{q_0, \dots, q_s\}$ and $\Sigma = \{a_1, \dots, a_n\}$.

The following program computes $f_{\mathscr{S}}$:

IN X;
$Q_0, \dots, Q_s, F_{q_0}, \dots, F_{q_s} \to e$;
LOOP$^{\leftarrow}$ X
⋮
j. $F_{q_0}, \dots, F_{q_s} \leftarrow Q_0, \dots, Q_s$; $Q_0 \leftarrow \lambda(q_0, a_j) F_{\delta(q_0, a_j)}$; …;
$Q_s \leftarrow \lambda(q_s, a_j) F_{\delta(q_s, a_j)}$;
⋮
END
OUT Q_0

where an instruction $A \leftarrow a_{i_1} \ldots a_{i_k}A$ is a short form for the list of instructions $A \leftarrow a_{i_k}A; \ldots; A \leftarrow a_{i_1}A$. Then $f \in L_{0,1}^{\leftarrow 1}$.

The register X is read leftwards. If a_i is the scanned symbol, then, corresponding to each rule $\delta(q_j, a_i) = q_h$, the content of Q_h is transferred to Q_i and $\lambda(q_j, a_i)$ is concatenated to Q_i on the left. If $x = a_{i_1} \ldots a_{i_k}$ and $\lambda(q_0, x) = y_1 \ldots y_k$, then Q_0 will contain $y_1 \ldots y_k$ at the end of the execution of the loop instruction.

Let us consider the function $f: \Sigma^* \to \Sigma^*$ such that $f = \lambda x.(xx)$.

As $f \in L_{0,1}^{\leftarrow 1}$, but there does not exist a gsm $\mathcal{S}$ such that $f = f_{\mathcal{S}}$, we have $f \in L_{0,1}^{\leftarrow 1} - G$.

Theorem 4.11. *The class of languages accepted by programs in $M_{1,1}$ strictly contains the class of deterministic context free languages.*

Proof. Let $\mathcal{L} \subseteq \Sigma^*$ be a deterministic context free language; let $\mathcal{A} = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$ be the deterministic push-down automaton accepting $\mathcal{L}$ with $\Sigma = \{a_1, \ldots, a_n\}$, $\Gamma = \{Z_0, \ldots, Z_{s_1}\}$, $Q = \{q_0, \ldots, q_{r_1}\}$, $F = \{q_{j_1}, \ldots, q_{j_t}\}$ and $\delta: Q \times (\Sigma \cup \{e\}) \times \Gamma \to Q \times \Gamma^*$ be the function such that $\delta(q_i, a, Z_j) = (q_{k_{i,j}}, V_{k_{i,j}})$ with $0 \leq j \leq s \leq s_1$ and $0 \leq i, k_{i,j} \leq r \leq r_1$.

Let $\Sigma' = \{c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+s+1}\}$, where $c_i = a_i$ for $1 \leq i \leq n$ and $c_{n+j+1} = Z_j$ for $0 \leq j \leq s$. We can give a LOOP program on Σ'^* in $M_{1,1}$ which computes the characteristic function of $\mathcal{L}$ by simulating the behaviour of the automata.

As regards the strictness of the containment, let us consider the function $f: \Sigma^* \to \Sigma^*$ such that $f(x) = a_1$ if $x = a_1^n a_2^n a_3^n$ and $f(x) = e$ otherwise. As f can be computed by a program in $M_{1,1}$, the claim holds.

Definition 4.12. A deterministic push-down transducer is a 8-tuple $\mathcal{T} = (\Sigma, \Delta, \Gamma, Q, \delta, q_0, Z_0, F)$ where Σ, Δ, Γ are the finite alphabets of input symbols, output symbols and stack symbols resp., Q is the finite set of states, $q_0 \in Q$, $Z_0 \in \Gamma$, $F \subseteq Q$ and δ is a partial function such that $\delta: Q \times (\Sigma \cap \{e\}) \times \Gamma \to Q \times \Delta^* \times \Gamma^*$ and if $\delta(q, e, Z)$ is defined, then $\delta(q, a_i, Z)$ is undefined for every $q \in Q$, $a_i \in \Sigma$, $Z \in \Gamma$.

Let $DPDT$ be the class of functions computed by deterministic push-down transducers.

Theorem 4.13. $DPDT \subsetneq L_{1,1}$.

Proof. The proof is analogous to that of the Theorem 4.11.

For the strictness of the containment note that the function $f: \Sigma^* \to \Sigma^*$ such that $f(x) = a_1^x a_2^x a_3^x$ belongs to L_1 but cannot be defined by any deterministic push-down transducer.

5. Decision problems

In this section the decidability of the equivalence problem for L_1 is stated by exploiting Gurari's result in [10] on decidability of the equivalence problem for two-way finite state transducers. In the same manner we prove the decidability of the equivalence problem for the class L_1^T of the wordfunctions computed by programs in M_1 which use also an **if–then–else** instruction testing the empty word. Note that the same result when the cardinality of Σ is one has been proved in [5] and [13]. But the equivalence problem turns out to be undecidable for the class $L_{0,1}^{D,T}$ of wordfunctions computed by programs in $M_{0,1}$ with a deleting–a–symbol instruction as a further basic instruction for card$(\Sigma) \geq 1$.

The graph and the range intersection problems are undecidable even for $L_{0,1}$ but only when the cardinality of Σ is greater than one; in fact, the two problems have been shown to be decidable for L_1^D when the cardinality of Σ is one in [5].

Theorem 5.1. *The equivalence problem is decidable for* $PL_1^{\leftrightarrow \mathrm{R}}$.

Proof. The proof follows from [3, 10].

Let $L_{i,j}^{\rightarrow \mathrm{L},T}$ $(i, j \geq 0)$ be the class of functions defined as the class $L_{i,j}^{\rightarrow \mathrm{L}}$ starting with

$$A'_{\mathrm{L}} = A_{\mathrm{L}} \cup \{t = \lambda x, y, z.(\textbf{if } x = 0 \textbf{ then } y \textbf{ else } z)\}$$

instead of A_{L}.

The classes $L_{i,j}^{\rightarrow \mathrm{R},T}$, $L_{i,j}^{\leftarrow \mathrm{L},T}$, $L_{i,j}^{\leftarrow \mathrm{R},T}$ are defined analogously, let $L_{i,j}^T$ stand for one of the above classes.

Theorem 5.2. *The equivalence problem is decidable for L_1 and L_1^T.*

Proof. The proof of Theorem 4.7 can immediately be extended to prove $L_1^T \subseteq TL_1^{\leftrightarrow \mathrm{R}}$ so the claim follows from Theorem 5.1.

Let $D: \Sigma^* \rightarrow \Sigma^*$ be the function such that $D(e) = e$ and $D(xa) = x$, and let $L_{i,j}^{\rightarrow \mathrm{L},T,D}$ be the class of functions defined as the class $L_{i,j}^{\rightarrow \mathrm{L}}$ starting with $A''_{\mathrm{L}} = A'_{\mathrm{L}} \cup \{D\}$.

The classes $L_{i,j}^{\leftarrow \mathrm{L},T,D}$, $L_{i,j}^{\leftarrow \mathrm{R},T,D}$, $L_{i,j}^{\rightarrow \mathrm{R},T,D}$ are defined analogously. Let $L_{i,j}^{T,D}$ stand for one of the above classes.

Theorem 5.3. *The equivalence problem is undecidable for* $L_{0,1}^{T,D}$ *for* card$(\Sigma) \geq 1$.

Proof. The halting problem for register machines can be reduced to the equivalence problem for $L_{0,1}^{T,D}$ with card$(\Sigma) = 1$.

Let a register machine be defined inductively as follows:

(i) $R_i \leftarrow R_i - 1$

(ii) $R_i \leftarrow R_i + 1$

(iii) **stop**

(iv) **if** M and N are register machines **then** $M; N$ is a register machine

(v) **if** M is a register machine **then** $(M)_i$ is a register machine (M is executed until $R_i = 0$)

assuming that the stop instruction occurs just once.

A register machine halts on input $x_1, \ldots, x_r$ iff it starts with the registers $R_1, \ldots, R_r$ containing $x_1, \ldots, x_r$, and the other possible registers containing 0, and reaches the stop instruction.

Given a gödelization of register machines, it holds that it is undecidable if a register machine (r.m.) with Gödel number g, M_g, halts on input g.

We consider a LOOP program computing a function $f: N \to N \in L_{0,1}^{T,D}$ such that $f(x) = 1$ if the r.m. M_g halts on input g in x steps and $f(x) = 0$ otherwise. The function f is equal to the constant function $C_0^1 = \lambda x.(0)$ iff M_g does not halt on input g.

Given a register machine M_g, suppose that every instruction of type (i), (ii), (iii) and every open and closed bracket are labelled by $1, \ldots, p$, and that $s \leqslant p$ is the label of the stop instruction. Consider the following LOOP program:

IN T

$X \leftarrow 0; \underbrace{X \leftarrow X + 1; \ldots; X \leftarrow X + 1}_{g}; \bar{Q}_1, \ldots, \bar{Q}_p \leftarrow 0; \bar{Q}_1 \leftarrow \bar{Q}_1 + 1;$

LOOP T

$Q_1, \ldots, Q_p \leftarrow \bar{Q}_1, \ldots, \bar{Q}_p;$

if $Q_1 = 1$ **then** I_1;

$\vdots$

if $Q_p = 1$ **then** I_p,

END

OUT Q_s

where

(1) if j is the label of an instruction of the type $R_i \leftarrow R_i \dot{-} 1$ or $R_i \leftarrow R_i + 1$, then

$$I_j = R_i \leftarrow R_i \dot{-} 1; \bar{Q}_j \leftarrow 0; \bar{Q}_{j+1} \leftarrow \bar{Q}_{j+1} + 1;$$

or

$$I_j = R_i \leftarrow R_i + 1; \bar{Q}_j \leftarrow 0; \bar{Q}_{j+1} \leftarrow \bar{Q}_{j+1},$$

(2) if j is the label of the stop instruction, then I_j is empty,

(3) if j is the label of the open bracket and $j + k$ is the label of the corresponding closed bracket, indexed by i, then

$$I_j = \textbf{if } X_i = 0 \textbf{ then } \bar{Q}_{j+k+1} \leftarrow \bar{Q}_{j+k+1} + 1; \bar{Q}_j \leftarrow 0;$$

$$\textbf{else } \bar{Q}_{j+1} \leftarrow \bar{Q}_{j+1} + 1; \bar{Q}_j \leftarrow 0;$$

$$I_{j+k} = \bar{Q}_{j+k} \leftarrow 0; \bar{Q}_j \leftarrow \bar{Q}_j + 1;$$

Theorem 5.4. *The graph and the range intersection problems for functions in* $L_{0,1}$ *are undecidable for* $\mathrm{card}(\Sigma) > 1$.

Proof. In [2], Ausiello and Moscarini proved this claim for L_1. But the programs that they give are in $M_{0,1}$, thus the claim holds.

Appendix A

In this appendix we give extended proofs of Theorems 3.5, 3.14, 3.25, 4.11 and 4.13 and Lemma 3.22. Below, some obvious abbreviations are used in the programs and comments are inserted between quotes.

Proof of Theorem 3.5. Let us prove the claim for $L_{i,j}^{\leftarrow L}$, as the proof is analogous for the other classes. The result is stated by induction on i and j.

Consider $L_0^{\leftarrow L}$. The function $S_i = \lambda x.(a_i x)$ is computed by P : IN X ; $X \leftarrow a_i X$; OUT X and $P \in M_0^{\leftarrow L}$; the function $E = (e)$ is computed by P: IN X ; $X \leftarrow e$; OUT X and $P \in M_0^{L}$; the function $K = \lambda x, y.(x)$ is computed by P: IN X, Y ; OUT Y and $P \in M_0^{L}$. Let $f_1, f_2 \in L_0^{L}$,

$$P_1 : \text{IN } X_1, \ldots, X_r ; I_1 ; \text{OUT } Y_1, \ldots, Y_s,$$
$$P_2 : \text{IN } Z_1, \ldots, Z_s ; I_2 ; \text{OUT } T_1, \ldots, T_q$$

such that $P_1, P_2 \in M_0^{\leftarrow L}$, they have disjoint sets of names for registers and compute f_1, f_2 resp. Then the program

$$P : \text{IN } X_1, \ldots, X_r ; I_1 ; Z_1 \leftarrow e ; \ldots ; Z_s \leftarrow e ; Z_1 \leftarrow Y_1 ; \ldots ; Z_s \leftarrow Y_s ; I_2 ;$$
$$\text{OUT } T_1, \ldots, T_q$$

computes the function $f = f_1 \circ f_2 \in L_0^{\leftarrow L}$.

Vice versa, if $P \in M_0^{\leftarrow L}$, then one of the following cases holds:

Case 1. P: IN $X_1, \ldots, X_r$; OUT $X_{i_1}, \ldots, X_{i_q}$ computes the function $f = U_{i_1}^r \,\hat{}\cdots\hat{}\, U_{i_q}^r$ where $i_j \in \{1, \ldots, r\}$ for $1 \leq j \leq q$.

Case 2. P: IN $X_1, \ldots, X_r ; X_j \leftarrow e$; OUT $X_{i_1}, \ldots, X_{i_q}$ computes the function $f = {}^{c^{j-1}}(E^1)^{c^{r-j}} \circ (U_{i_1}^r \,\hat{}\cdots\hat{}\, U_{i_q}^r)$ if $1 \leq j \leq r$ and the function $f = {}^{c^r}E \circ (U_{i_1}^{r+1} \,\hat{}\cdots\hat{}\, U_{i_q}^{r+1})$ otherwise.

Case 3. P: IN $X_1, \ldots, X_r ; X_h \leftarrow X_k$; OUT $X_{i_1}, \ldots, X_{i_q}$ computes the function $f = T_{h,k}^r \circ (U_{i_1}^r \,\hat{}\cdots\hat{}\, U_{i_q}^r)$ where $1 \leq h$, $k \leq r$.

Case 4. P: IN $X_1, \ldots, X_r$; $X_h \leftarrow a_i X_h$; OUT $X_{i_1}, \ldots, X_{i_q}$ computes the function $f = {}^{c^{h-1}}S_i^{c^{r-h}} \circ (U_{i_1}^r \,\hat{}\cdots\hat{}\, U_{i_q}^r)$ where $1 \leq h \leq r$ and $1 \leq i \leq n$. As U_i^r, $T_{h,k}^r$, $E^1 \in L_0^{\leftarrow L}$, we have $f \in L_0^{\leftarrow L}$.

Case 5. If P: IN $X_1, \ldots, X_r ; I_1 ; \cdots ; I_k$; OUT $X_{i_1}, \ldots, X_{i_q}$ where I_i is a clear, copy or L-append instruction, consider the programs P_i : IN s_i ; I_i ; OUT t_i for $1 \leq i \leq k$, with $s_1 = s$ and $s_i = s, s_{i-1}'$ for $1 < i \leq k$ where s_{i-1}' is the list of new registers introduced by $I_1, \ldots, I_{i-1}$ and $t_i = s_{i+1}$ for $1 \leq i < k$ and $t_k = t$. As by induction hypothesis the functions f_i computed by the programs P_i belong to $L_0^{\leftarrow L}$, we have that the function $f = f_1 \circ \cdots \circ f_k$, computed by P, belongs to $L_0^{\leftarrow L}$.

For the inductive step we prove the following two assertions.

(a) Let $f = (g_1, \ldots, g_n)^{1\leftarrow}$; if $g_1, \ldots, g_n$ are computed by $P_1, \ldots, P_n \in M_i^{\leftarrow L}$, then a program P computing f belongs to $M_{i,1}^{\leftarrow M}$.

In fact, if P_i is the program IN $X_1^i, \ldots, X_s^i$; I_i ; OUT $Y_1^i, \ldots, Y_s^i$ for $1 \leq i \leq n$, then $P \in M_{i,1}^{\leftarrow L}$ is obtained as follows:

IN $X_1, \ldots, X_{s+1}$
$X_1^1 \leftarrow e; \cdots ; X_s^n \leftarrow e;$
LOOP$^{\leftarrow}$ X_1
1. $X_1^1 \leftarrow X_2; \ldots ; X_s^1 \leftarrow X_{s+1}; \ldots ; I_1 ; X_2 \leftarrow Y_1^1 ; \ldots ; X_{s+1} \leftarrow Y_s^1$;
⋮
n. $X_1^n \leftarrow X_2; \ldots ; X_s^n \leftarrow X_{s+1}; I_n ; X_2 \leftarrow Y_1^n ; \ldots ; X_{s+1} \leftarrow Y_s^n$;
END
OUT $X_2, \ldots, X_{s+1}$

(b) If $f = f_1 \circ f_2$ and f_1 and f_2 are computed by $P_1 \in M_{i,j}^{\leftarrow L}$ and $P_2 \in M_{i,k}^{\leftarrow L}$ then a program P computing f belongs to $M_{i,j+k}^{\leftarrow L}$.

In fact if P_1 and P_2 are the programs

IN $X_1, \ldots, X_r ; I_1$; OUT $Y_1, \ldots, Y_m$
and IN $Z_1, \ldots, Z_m ; I_2$; OUT $T_1, \ldots, T_s$,

then the program $P \in M_{i,j+k}^{\leftarrow L}$ is obtained as follows:

P: IN $X_1, \ldots, X_r ; I_1 ; Z_1 \leftarrow e ; \ldots ; Z_m \leftarrow e ; Z_1 \leftarrow Y_1 ; \ldots ;$

$Z_m \leftarrow Y_m$; I_2; OUT $T_1, \ldots, T_s$.

Vice versa, consider the following two assertions.

(a′) Let P_h: IN $X_1^h, \ldots, X_r^h$; I_h ; OUT $Y_1^h, \ldots, Y_s^h$ be a program computing $g_h \in L_i^{\leftarrow L}$ for $1 \leq h \leq n$. Then the program P, obtained from $P_1, \ldots, P_n$ as in assertion (a), computes a function $f \in L_{i,1}^{\leftarrow L}$.

In fact, P computes $f = (g_1, \ldots, g_n)^{1\leftarrow}$ which belongs to $L_{i,1}^{\leftarrow L}$ by definition.

(b′) Let

P_1: IN $X_1, \ldots, X_r$; I_1 ; OUT $Y_1, \ldots, Y_m$ and
P_2: IN $Z_1, \ldots, Z_m$; I_2 ; OUT $T_1, \ldots, T_s$

be programs computing $f_1 \in L_{i,j}^{\leftarrow L}$ and $f_2 \in L_{i,h}^{\leftarrow L}$; then the program P obtained from P_1 and P_2 as in assertion (b) computes $f \in L_{i,j+h}^{\leftarrow L}$.

In fact, P computes $f = f_1 \circ f_2$ which belongs to $L_{i,j+h}^{\leftarrow L}$ by Lemma 2.3.

Proof of Theorem 3.14. Let us prove the thesis for $L_{i,j}^{\rightarrow R}$. Consider a function $h = f \,\hat{}\, g$ with $f : \Sigma^{*'} \to \Sigma^{*'}$ and $g : \Sigma^{*'} \to \Sigma^{*'}$ belonging to $L_{i,j}^{\rightarrow R}$. By definition it holds that

$f = f_0 \circ f_1 \circ \cdots \circ f_j$ and $g = g_0 \circ g_1 \circ \cdots \circ g_j$ with $f_0, g_0 \in L_i^{\to R}$ and $f_1, \ldots, f_j, g_1, \ldots, g_j \in \mathcal{I}^{\to}(L_i^{\to R})$. Then f, g can be computed by the following programs P and R respectively:

IN $X_1, \ldots, X_r$	IN $Y_1, \ldots, Y_r$
P_0;	R_0;
LOOP$^{\to}$ $\bar{X}_1$	LOOP$^{\to}$ $\bar{Y}_1$
1. P_1^1;	1. R_1^1;
$\vdots$	$\vdots$
$n.$ P_n^1;	$n.$ R_n^1;
END	END
$\vdots$	$\vdots$
LOOP$^{\to}$ $\bar{X}_j$	LOOP$^{\to}$ $\bar{Y}_j$
1. P_1^j;	1. R_1^j;
$\vdots$	$\vdots$
$n.$ P_n^j;	$n.$ R_n^j;
END	END
OUT $X_1', \ldots, X_s'$	OUT $Y_1', \ldots, Y_t'$

where

P_0, LOOP$^{\to}$ $\bar{X}_1$ 1. $P_1^1; \ldots; n.\ P_n^1$; END, $\ldots$, LOOP$^{\to}$ $\bar{X}_j$ 1. $P_1^j; \ldots; n.\ P_n^j$; END,

R_0, LOOP$^{\to}$ $\bar{Y}_1$ 1. $R_1^1; \ldots; n.\ R_n^1$; END, $\ldots$, LOOP$^{\to}$ $\bar{Y}_j$ 1. $R_1^j; \ldots; n.\ R_n^j$; END,

with a proper specification of input and output registers, compute $f_0, f_1, \ldots, f_j$, $g_0, g_1, \ldots, g_j$ respectively. Suppose that P and R use different names for registers.

Now consider the following program $\bar{P}$:

IN $A_1, \ldots, A_r$
$\bar{P}_0$;
$P_0; R_0$;
$A_1^1, B_1^1, C_1^1, \ldots, A_n^j, B_n^j, C_n^j \leftarrow e$;
$C_1^1 \leftarrow C_1^1 a_1, \ldots, C_n^j \leftarrow C_n^j a_1; Q \leftarrow C_1^1$;
$B_1^1 \leftarrow C_1^1; \ldots; B_n^j \leftarrow C_n^j$;
$W_1 \leftarrow \bar{X}_1; T_1 \leftarrow$ CONC$(\bar{X}_1, \bar{Y}_1)$;
LOOP$^{\to}$ T_1
1. S_1^1;
$\vdots$
$n.$ S_n^1
END;
$\bar{P}_1^1; \ldots; \bar{P}_n^1$;
$\vdots$
$W_j \leftarrow X_j; T_j \leftarrow$ CONC$(\bar{X}_j, \bar{Y}_j)$;

LOOP$^{\to}$ T_j
1. S_1^j;
⋮
n. S_n^j;
END;
$\bar{\bar{P}}_1^j; \ldots; \bar{\bar{P}}_n^j$;
OUT $X_1', \ldots, X_s', Y_1', \ldots, Y_t'$

where S_i^h, for $1 \leq h \leq j$, $1 \leq i \leq n$, is the following sequence of instructions:

S_i^h : LOOP$^{\to}$ W_h
1. $B_i^h \leftarrow e; A_i^h \leftarrow Q$;
⋮
n. $B_i^h \leftarrow e; A_i^h \leftarrow Q$;
END
LOOP$^{\to}$ B_i^h;
1. $\bar{P}_1^h; \ldots; \bar{P}_n^h; C_1^h, B_1^h, \ldots, C_n^h, B_n^h \leftarrow e$;
⋮
n. $X \leftarrow X$;
END
LOOP$^{\to}$ C_i^h
1. $\bar{R}_1^h, \ldots, \bar{R}_n^h$;
⋮
n. $X \leftarrow X$;
END

$R_i^h; P_i^h; W_h \leftarrow$ DELL(W_h);
LOOP$^{\to}$ W_h
1. $A_i^h \leftarrow e; C_1^h, \ldots, C_n^h \leftarrow e$;
⋮
n. $A_i^h \leftarrow e; C_1^h, \ldots, C_n^h \leftarrow e$;
END
LOOP$^{\to}$ A_i^h
1. $\bar{P}_1^h; \ldots; \bar{P}_n^h; \bar{\bar{R}}_i^h; \ldots; \bar{\bar{R}}_n^h; C_1^h \leftarrow e; \ldots; C_n^h \leftarrow e$;
$B_1^h, \ldots, B_n^h \leftarrow e; A_i^h \leftarrow e$;
⋮
n. $X \leftarrow X$;
END

where $\bar{P}_0$ provides the assignment of the input data to the input registers of P and R, $\bar{R}_i^h$ and $\bar{P}_i^h$ provide to the assignment of the content of the registers occurring in the sequence of instructions R_i^h and P_i^h resp. to new registers and $\bar{\bar{R}}_i^h$

and $\bar{P}_i^h$ provide to the restoration of the values stored in these new registers in those occurring in R_i^h and P_i^h resp. Moreover, $T \leftarrow \text{CONC}(X, Y)$ is shorthand for a list of instructions whose effect is to store the concatenation of the content of X and Y, in the order, in T and $W \leftarrow \text{DELL}(W)$ is shorthand for the following list of instructions, which has the effect of deleting the last symbol of the word contained in the register W:

$W' \leftarrow e;$
$\text{LOOP}^{\rightarrow}\ W$
1. $W \leftarrow W'; W' \leftarrow W'a_1;$
$\vdots$
$n.$ $W \leftarrow W'; W' \leftarrow W'a_n;$
END

The proof for the other hierarchies is obtained in a similar manner taking in account that we might delete the first symbol instead of the last one because W_h is only a counter.

Proof of Lemma 3.22. If $f: \Sigma^{*^r} \to \Sigma^{*^s} \in L_{1,j}^{\leftarrow R}$, then the computing time function $t_f \in L_{1,j}^{\leftarrow R}$. By Lemma 3.12 there exists a $w \in \Sigma^*$ such that, for every $u = x_1, \ldots, x_r$, $|t_f(u)| < \bar{F}_j^1(w, x_k)$ where $|x_k| = \|u\|$. We can write a loop program $P \in M_{1,1}^{\rightarrow R}$ computing a function $g: \Sigma^* \to \Sigma^{*^s}$ such that

$$f(x_1, \ldots, x_r) = g(\bar{F}_j^1(w, \rho^r(x_1, \ldots, x_r)))$$

where $\rho^r: \Sigma^{*^r} \to \Sigma^*$ is a function such that $\rho^r(x_1, \ldots, x_r) = x_1 \ldots x_r$. As $\lambda x.\bar{F}_j^1(w, x) \in L_{1,j}$ and $\rho^r \in L_1$, we have $f \in L_{1,j+1}^{\rightarrow R}$.

Let the following program P' compute f:

$$\text{IN } X_1, \ldots, X_r; I_0; I_1^0; \ldots; I_q^0; I_1^1; \ldots; I_j^1; \text{OUT } Y_1, \ldots, Y_s,$$

where

$I_k^1 = \text{LOOP}^{\leftarrow}\ T_k$
$\vdots$
$i.$ $I_{i,k}^1;$
$\vdots$
$\text{LOOP}^{\leftarrow}\ U_{i,k}^l$
$\vdots$
$i'.I_{i,k,i'}^{2,l}$
$\vdots$
END
$\vdots$
END

for $1 \leq k \leq j$, $1 \leq i, i' \leq n$, $0 \leq l \leq s_{i,k}$, where $s_{i,k}$, is the number of LOOP instructions occurring in the ith branch of the instruction I_k^1.

$I_h^0 = \text{LOOP}^{\leftarrow}\ V_h$
⋮
$i.\ I_{i,h}^0$;
⋮
END

for $1 \leq h \leq q$, $1 \leq i \leq n$, and I_0, $I_{i,k}^1$, $I_{i,k,i'}^{2,l}$, $I_{i,h}^0$ are lists of clear, copy and R-append instructions.

The program P computing g is the following:

IN T
$Z \leftarrow e; Z \leftarrow Za_1; M_1 \leftarrow Z; Z_1 \leftarrow Z; R_1, \ldots, R_{q+j} \leftarrow e$;
$\text{LOOP}^{\leftarrow}\ T$
1. $\bar{I}_0$; "$\bar{I}_0$ simulates the execution of I_0"
$\bar{I}_1^0$;
⋮ "$\bar{I}_h^0$ simulates the execution of I_h^0 for $1 \leq h \leq q$"
$\bar{I}_q^0$;
$\bar{I}_1^1$;
⋮ "$\bar{I}_k^1$ simulates the execution of I_k^1 for $1 \leq k \leq j$"
$\bar{I}_j^1$;
2. $X \leftarrow X$
⋮
$n.\ X \leftarrow X$
END

where

$\bar{I}_0 = \text{LOOP}^{\leftarrow}\ M_1$
1. $I_0; M_1 \leftarrow e$;
2. $X \leftarrow X$;
⋮
$n.\ X \leftarrow X$
END

$\bar{I}_1^0 = \bar{V}_1 \leftarrow \text{LAST}(V_1); V_1 \leftarrow \text{DELL}(V_1); R_1 \leftarrow Z_1$;
$\text{LOOP}^{\rightarrow}\ \bar{V}_1$
1. $I_{1,1}^0$; $R_1 \leftarrow e$;
⋮
$n.\ I_{n,1}^0$; $R_1 \leftarrow e$;
END

$\bar{I}_h^0 = \text{LOOP}^{\rightarrow} R_{h-1}$ $\qquad 2 \leq h \leq q$

1. $V_h' \leftarrow V_h; Z_{h-1} \leftarrow e; Z_h \leftarrow Z;$

$\vdots$

n. $X \leftarrow X;$

END

$\bar{V}_h \leftarrow \text{LAST}(V_h'); V_h' \leftarrow \text{DELL}(V_h'); R_h \leftarrow Z_h;$

$\text{LOOP}^{\rightarrow} \bar{V}_h;$

1. $I_{1,h}^0; R_h \leftarrow e;$

$\vdots$

n. $I_{n,h}^0; R_h \leftarrow e;$

END

$\bar{I}_k^1 = \text{LOOP}^{\rightarrow} R_{q+k-1}$ $\qquad 1 \leq k \leq j$

1. $C_k \leftarrow T_k; Z_{q+k-1} \leftarrow e; Z_{q+k} \leftarrow Z;$

$\vdots$

n. $X \leftarrow X;$

END

$A_k \leftarrow \text{LAST}(C_k); C_k \leftarrow \text{DELL}(C_k); R_{q+k} \leftarrow Z_{q+k};$

$\vdots$

$G_k, C_{1,k}^1, N_{1,k}^1, \ldots, C_{n,k}^{s_{n,k}}, N_{n,k}^{s_{n,k}}, B_1^k, \ldots, B_n^k \leftarrow e;$

$\text{LOOP}^{\rightarrow} A_k$

1. $B_1^k \leftarrow Z; N_{1,k}^1 \leftarrow U_{1,k}^1; \ldots; N_{1,k}^{s_{1,k}} \leftarrow U_{1,k}^{s_{1,k}};$

$\vdots$

n. $B_n^k \leftarrow Z; N_{n,k}^1 \leftarrow U_{n,k}^1; \ldots; N_{n,k}^{s_{n,k}} \leftarrow U_{n,k}^{s_{n,k}};$

END

"If A_k contains a_i, then the simulation of the ith branch is prepared."

$\text{LOOP}^{\rightarrow} B_1^k$

1. $I_{1,k}^1; C_{1,k}^1 \leftarrow N_{1,k}^1; B_1^k \leftarrow e; R_{q+k} \leftarrow e; Z_{1,k}^1 \leftarrow e; Z_{1,k}^1 \leftarrow Z; G_k \leftarrow Z;$
2. $X \leftarrow X;$

$\vdots$

n. $X \leftarrow X;$

END

$\vdots$

$\text{LOOP}^{\rightarrow} B_n^k$

1. $I_{n,k}^1; C_{n,k}^1 \leftarrow N_{n,k}^1; B_n^k \leftarrow e; R_{q+k} \leftarrow e; Z_{n,k}^1 \leftarrow e; Z_{n,k}^1 \leftarrow Z; G_k \leftarrow Z;$
2. $X \leftarrow X;$

$\vdots$

n. $X \leftarrow X;$

END

LOOP$^{\rightarrow}$ G_k
1. $D_k \leftarrow C_k; C_k \leftarrow e; G_k \leftarrow e;$
2. $X \leftarrow X;$
$\vdots$
n. $X \leftarrow X;$
END
$A^1_{i,k} \leftarrow$ LAST$(C^1_{i,k}); C^1_{i,k} \leftarrow$ DELL$(C^1_{i,k}); M^1_{i,k} \leftarrow e; M^1_{i,k} \leftarrow Z^1_{i,k};$
LOOP$^{\rightarrow}$ $A^1_{i,k}$
1. $I^{2,1}_{i,k,1};$
$\vdots$
n. $I^{2,1}_{i,k,n};$
END
LOOP$^{\rightarrow}$ $C^1_{i,k}$
1. $M^1_{i,k} \leftarrow e; R_{q+k} \leftarrow e;$
$\vdots$
n. $M^1_{i,k} \leftarrow e; R_{q+k} \leftarrow e;$
END
LOOP$^{\rightarrow}$ $M^1_{i,k}$
1. $C^2_{i,k} \leftarrow N^2_{i,k}; Z^1_{i,k} \leftarrow e; Z^2_{i,k} \leftarrow e; Z^2_{i,k} \leftarrow Z;$
2. $X \leftarrow X;$
$\vdots$
n. $X \leftarrow X;$
END
'If the first nested loop has been executed, the simulation of the second one starts"
$\vdots$
$A^{s_{i,k}}_{i,k} \leftarrow$ LAST$(C^{s_{i,k}}_{i,k}); C^{s_{i,k}}_{i,k} \leftarrow$ DELL$(C^{s_{i,k}}_{i,k});$
$M^{s_{i,k}}_{i,k} \leftarrow e; M^{s_{i,k}}_{i,k} \leftarrow Z^{s_{i,k}}_{i,k}:$
LOOP$^{\rightarrow}$ $A^{s_{i,k}}_{i,k}$
1. $I^{2,s_{i,k}}_{i,k,1};$
$\vdots$
n. $I^{2,s_{i,k}}_{i,k,n};$
END
LOOP$^{\rightarrow}$ $C^{s_{i,k}}_{i,k}$
1. $M^{s_{i,k}} \leftarrow e; R_{q+k} \leftarrow e;$
$\vdots$
n. $M^{s_{i,k}}_{i,k} \leftarrow e; R_{q+k} \leftarrow e;$
END
LOOP$^{\rightarrow}$ $M^{s_{i,k}}_{i,k}$
1. $C_k \leftarrow D_k; Z^{s_{i,k}}_{i,k} \leftarrow e;$

2. $X \leftarrow Y$;
   ⋮
n. $X \leftarrow X$;
END

"The content of the control register of the instruction I_k^1 is restored in C_k so that a new symbol can be scanned"

Proof of Theorem 3.25. By Proposition 3.16 we have only to prove that if $t_f \in L_{i,j}$, then $f \in L_{i,j}$. Let us prove the claim for $L_{i,j}^{\leftarrow R}$. Let $f: \Sigma^{*^r} \to \Sigma^{*^s}$. If $t_f: \Sigma^{*^r} \to \Sigma^* \in L_{i,j}^{\leftarrow R}$, then there exists a $w \in \Sigma^*$ such that, for every $u = x_1, \ldots, x_r$ belonging to Σ^{*^r}, $|t_f(u)| < \bar{F}_j^i(w, x_k)$ where $|x_k| = \|u\|$. We can give a program P computing $g: \Sigma^* \to \Sigma^{*^s} \in L_{1,1}^{\leftarrow R}$ such that $f(x_1, \ldots, x_r) = g(F_j^i(w, x_k))$.

As $\lambda x.\bar{F}_j^i(w, x_k) \in L_{i,j}^{\leftarrow R}$ and $i \geq 2$, we have $f \in L_{i,j}^{\leftarrow R}$. Let $P' \in M_{h,k}^{R}$ with $h \geq i$ or $k \geq j$ be a program computing f. Suppose that every clear, copy and R-append instructions and every keywords LOOP, END of P' are labelled and let $l_1, \ldots, l_m$ be the list of such labels.

The following program P has to simulate the execution of $P' =$ IN $X_1, \ldots, X_r; I$; OUT $Y_1, \ldots, Y_s$.

We will use some obvious short forms: we will write $Z_1, \ldots, Z_p \leftarrow e$ for $Z_1 \leftarrow e; \ldots; Z_p \leftarrow e$; and $Z_1, \ldots, Z_p \leftarrow Z_1', \ldots, Z_p'$ for $Z_1 \leftarrow Z_1'; \ldots; Z_p \leftarrow Z_p'$; for every $p \geq 2$.

$P =$ IN T
  $B_1, \ldots, B_m, A_1, \ldots, A_m \leftarrow e$;
  $B_1 \leftarrow B_1 a_1$;
  LOOP$^{\leftarrow}$ T;
  1. $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_m; \bar{I}_1; \ldots; \bar{I}_m$;
  2. $X \leftarrow X$;
     ⋮
  n. $X \leftarrow X$;
  END
  OUT $Y_1, \ldots, Y_s$

The list of instructions $\bar{I}_q$ is defined as follows:

(1) If l_q is a label of a clear, copy or R-append instruction I_q, then

$\bar{I}_q =$ LOOP$^{\leftarrow}$ A_q
  1. $A_q \leftarrow e; B_q \leftarrow e; B_{q+1} \leftarrow B_{q+1} a_1; I_q$;
  2. $X \leftarrow X$;
     ⋮
  n. $X \leftarrow X$;
  END

"If A_q is not empty, I_q is executed and B_{q+1} is set to a_1 to prepare the next step of the simulation."

(2) Let l_q be the label of a word LOOP X_q, let $l_{q+p_1+\cdots+p_{t-1}}+1, \ldots, l_{q+p_1+\cdots+p_t}$, $1 \leq t \leq n$, be the labels inside the branch corresponding to a_t in the loop instruction, let $l_{q+p_1+\cdots+p_n}+1$ be the label of the word END. Let us write p'_t for $p_1+\cdots+p_t$. Then

$\bar{I}_q =$ LOOP$^{\leftarrow}$ A_q
  1. $A_q \leftarrow e; B_q \leftarrow e; B_{q+p'_n+1} \leftarrow B_{q+p'_n+1}a_1; U \leftarrow X_q;$
  2. $X \leftarrow X;$
  $\vdots$
  n. $X \leftarrow X;$
  END

"$\bar{I}_q$ set $B_{q+p'_n+1}$ to a_1 to prepare the exit from the loop"

  $T_1, \ldots, T_n \leftarrow B_{q+p'_1}, \ldots, B_{q+p'_n};$
  $\vdots$
  LOOP$^{\leftarrow}$ T_i
  1. $U \leftarrow S; B_{q+p'_i} \leftarrow e; T_i \leftarrow e; B_{q+p'_n+1} \leftarrow B_{q+p'_n+1}a_1;$
  2. $X \leftarrow X;$
  $\vdots$
  n. $X \leftarrow X;$
  END   "$1 \leq i \leq n$"

"If T_i is not empty, then the simulation of the instructions of the ith branch has been executed and the content of the control register X_q is restored in U to test a new symbol."

  $Z \leftarrow$ LAST(U);
  $U \leftarrow$ DELL(U);
  $C_q \leftarrow U;$
  $U \leftarrow e;$
  LOOP$^{\leftarrow}$ Z
  $\vdots$
  i. $B_{q+p'_{i-1}+1} \leftarrow B_{q+p'_{i-1}+1}a_1; Z \leftarrow e; B_{q+p'_n+1} \leftarrow e; S \leftarrow C_q;$
  $\vdots$
  END

"If Z contains a_i, $1 \leq i \leq n$, the simulation of the ith branch starts"

(3) If l_q is a label of a keyword END, then

$\bar{I}_q =$ LOOP$^{\leftarrow}$ A_q
  1. $A_q \leftarrow e; B_q \leftarrow e; B_{q+1} \leftarrow B_{q+1}a_1;$
  2. $X \leftarrow X;$
  $\vdots$
  n. $X \leftarrow X;$
  END

Proof of Theorem 4.11. Let $\mathcal{L} \subseteq \Sigma^*$ be a deterministic context free language; let $\mathcal{A} = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$ be the deterministic push-down automaton accepting $\mathcal{L}$ with $\Sigma = \{a_1, \ldots, a_n\}$, $\Gamma = \{Z_0, \ldots, Z_{s_1}\}$, $Q = \{q_0, \ldots, q_{r_1}\}$, $F = \{q_{j_1}, \ldots, q_{j_t}\}$ and $\delta : Q \times (\Sigma \cup \{e\}) \times \Gamma \to Q \times \Gamma^*$ be the function such that $\delta(q_i, a, Z_j) = (q_{k_{i,j}}, V_{k_{i,j}})$ with $0 \leqslant j \leqslant s \leqslant s_1$ and $0 \leqslant i, k_{i,j} \leqslant r \leqslant r_1$.

We will suppose that $\mathcal{A}$ goes through the input word rightwards and the top of the push-down stack be the leftmost symbol.

Let

$$h = \max\{|V_{k_{i,j}}| \mid \delta(q_i, a, Z_j) = (q_{k_{i,j}}, V_{k_{i,j}}) \text{ for some } a \in \Sigma \cup \{e\},$$
$$q_i, q_{k_{i,j}} \in Q, Z_j \in \Gamma\}.$$

The maximum number of steps of the push-down automaton $\mathcal{A}$ on the input x is $(h+1)(2|x|+1)$ (see [8]). Let us prove the claim for $L^{\leftarrow 1}_{1,1}$.

Let us consider the following program on Σ'^* with $\Sigma' = \{c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+s+1}\}$, where $c_i = a_i$ for $1 \leqslant i \leqslant n$ and $c_{n+j+1} = Z_j$ for $0 \leqslant j \leqslant s$:

IN X;
$H \leftarrow e$;
LOOP$^{\leftarrow}$ X
1. $H \leftarrow a_1 H; H \leftarrow a_1 H$;
$\vdots$
n. $H \leftarrow a_1 H; H \leftarrow a_1 H$;
$n+1$. $H \leftarrow H$;
$\vdots$
$n+s+1$. $H \leftarrow H$;
END
$H \leftarrow a_1 H$;
LOOP$^{\leftarrow}$ H
1. $H \leftarrow \underbrace{a_1 H; \ldots; H \leftarrow a_1 H}_{h}$;
$\vdots$
n. $H \leftarrow \underbrace{a_1 H; \ldots; H \leftarrow a_1 H}_{h}$;
$n+1$. $H \leftarrow H$;
$\vdots$
$n+s+1$. $H \leftarrow H$;
END
"H contains $a_1^{(h+1)(2|x|+1)}$"
$T, Y, W, V, V', U, X' \leftarrow e$;
$Q_1, \ldots, Q_r, A_1, \ldots, A_r, A'_1, \ldots, A'_r, B_1, \ldots, B_r \leftarrow e$;

$Q_0 \leftarrow a_1 Q_0$; $W \leftarrow Z_0 W$;
LOOP$^{\leftarrow}$ H
1. $Y \leftarrow$ FIRST(W); $W \leftarrow$ DELF(W); $V \leftarrow a_1 V$;
$I_{q_0}; \ldots; I_{q_r}$;

"W contains the word in the stack. I_{q_i} will test whether Q_i is nonempty (i.e., whether q_i is the current state or not): if $Q_i \neq e$, then the top symbol of the stack is put in the register A_i)"

$V' \leftarrow a_1 V'; I^e_{A_1}; \ldots; I^e_{A_r}$;

"$I^e_{A_i}$ updates W and the suitable Q_k in accordance to $\delta(q_i, e, Z)$ for the current value of Z". If $\delta(q_i, e, Z)$ is undefined for every i, then $V' \neq e$"

LOOP$^{\leftarrow}$ V';
1. $U \leftarrow X$;
⋮
$n+s+1$. $X \leftarrow X$;
END;
$X' \leftarrow$ FIRST(U); $U \leftarrow$ DELF(U);

"The current input symbol is put in X' iff $V' \neq e$, that is if no e-move has been executed."

LOOP$^{\leftarrow}$ V';
1. $X \leftarrow U$; $U \leftarrow e$; $V' \leftarrow e$;
⋮
$n+s+1$. $X \leftarrow X$;
END;
LOOP$^{\leftarrow}$ X'
1. $B^1_1, \ldots, B^1_r \leftarrow A_1, \ldots, A_r; A_1, \ldots, A_r \leftarrow e$;
⋮
n. $B^n_1, \ldots, B^n_r \leftarrow A_1, \ldots, A_r; A_1, \ldots, A_r \leftarrow e$;
⋮
$n+s+1$. $X \leftarrow X$;
END

"If the current symbol is a_j and the current state is q_i, then $B^j_i \neq e$ and I^j_i modifies the suitable Q_k and W in accordance to $\delta(q_i, a_j, Z)$."

LOOP$^{\leftarrow}$ V;
1. $W \leftarrow e; Q_1, \ldots, Q_r \leftarrow e$;
⋮
$n+s+1$. $X \leftarrow X$;
END;

"If no move has been executed, then W and every Q_i must be deleted."

$$
\begin{array}{rl}
2. & X \leftarrow X; \\
 & \vdots \\
n+s+1. & X \leftarrow X; \\
\multicolumn{2}{l}{\text{END};} \\
\multicolumn{2}{l}{\text{LOOP}^{\leftarrow}\ Q_{j_1};} \\
1. & T \leftarrow a_1 T; \\
 & \vdots \\
n+s+1. & X \leftarrow X; \\
\multicolumn{2}{l}{\text{END}} \\
\multicolumn{2}{l}{\vdots} \\
\multicolumn{2}{l}{\text{LOOP}^{\leftarrow}\ Q_{j_t};} \\
1. & T \leftarrow a_1 T; \\
 & \vdots \\
n+s+1. & X \leftarrow X; \\
\multicolumn{2}{l}{\text{END}}
\end{array}
$$

"If a register among $Q_{j_1}, \ldots, Q_{j_1}$, corresponding to the final states $q_{j_1}, \ldots, q_{j_t}$, is nonempty, T is set to a_1."

where

$$
\begin{array}{ll}
I_{q_i} = \text{LOOP } Q_i; & \\
\quad 1.\ A_i \leftarrow Y; Q_i \leftarrow e; Y \leftarrow e; V \leftarrow e; & \\
\quad \vdots & \\
\quad n+s+1.\ X \leftarrow X; & \\
\quad \text{END}; & 0 \leqslant i \leqslant r \\
I^{e}_{A_i} = \text{LOOP}^{\leftarrow}\ A_i; & \\
\quad 1.\ X \leftarrow X; & \\
\quad \vdots & \\
\quad n+m.\ Q_i \leftarrow e; W \leftarrow \tilde{v}_{k_{i,m}} W; Q_{k_{i,m}} \leftarrow a_1 Q_{k_{i,m}}; V \leftarrow e; A_1, \ldots, A_r \leftarrow e; & \\
\quad \vdots & \\
\quad n+s+1.\ X \leftarrow X; & \\
\quad \text{END}; &
\end{array}
$$

where

$$\delta(q_i e, Z_m) = (q_{k_{i,m}}, v_{k_{i,m}}) \quad \text{and} \quad W \leftarrow \widetilde{Z_{i_1} \ldots Z_{i_p}} W \quad \text{for } Z_{i_1} \ldots Z_{i_p} \in \Gamma^*$$

is used instead of $W \leftarrow Z_{i_1} W; \ldots; W \leftarrow Z_{i_p} W$.

$$
\begin{aligned}
I_i^j = \text{LOOP}^{\leftarrow}\ B_i^j \\
&1.\ X \leftarrow X;\\
&\quad\vdots\\
&n+1.\ V \leftarrow a_1 V;\\
&\quad\vdots\\
&n+m+1.\ Q_i \leftarrow e; W \leftarrow \tilde{v}_{k_{i,m}} W; Q_{k_{i,m}} \leftarrow a_1 Q_{k_{i,m}}; B_1^1, \ldots, B_r^n \leftarrow e;\\
&\quad\vdots\\
&n+s+1.\ V \leftarrow a_1 V;\\
&\text{END};
\end{aligned}
$$

where $\delta(q_i, a_j, Z_l)$ is undefined and $\delta(q_i, a_j, Z_m) = (q_{k_{i,m}}, v_{k_{i,m}})$ for $0 \leqslant i \leqslant r$ and $1 \leqslant j \leqslant n$, $0 \leqslant l \neq m \leqslant s$.

As regards the strictness of the containment let us consider the function $f : \Sigma^* \to \Sigma^*$ such that $f(x) = a_1$ if $x = a_1^n a_2^n a_3^n$ and $f(x) = e$ otherwise. The following program computes f:

$$
\begin{aligned}
&\text{IN } X;\\
&A, A', C, X', Y', Y, Y_1, Z, Z_1, Z_2, Z_3, T \leftarrow e;\\
&X' \leftarrow X; C \leftarrow X; A' \leftarrow a_1 A', T \leftarrow a_1 T; A \leftarrow A';\\
&\text{LOOP}^{\leftarrow}\ X\\
&1.\ C \leftarrow a_1 C;\\
&\quad\vdots\\
&n.\ C \leftarrow a_n C;\\
&\text{END};\\
&\text{LOOP}^{\leftarrow}\ C\\
&1.\ Y' \leftarrow S_1(Y'); M; Z \leftarrow \text{DELF}(Z); Y_1 \leftarrow \text{DELF}(Y_1); Z_3 \leftarrow \text{DELF}(Z_3);\\
&\quad Z_3 \leftarrow \text{DELF}(Z_3); A \leftarrow A';\\
&2.\ Y' \leftarrow S_2(Y'); M; Z_2 \leftarrow \text{DELF}(Z_2); A \leftarrow A';\\
&3.\ Y' \leftarrow S_3(Y'); M; Z_1 \leftarrow \text{DELF}(Z_1); Y \leftarrow \text{DELF}(Y); A \leftarrow A';\\
&4.\ T \leftarrow e;\\
&\quad\vdots\\
&n.\ T \leftarrow e;\\
&\text{END};
\end{aligned}
$$

"Where $Y' \leftarrow S_i(Y')$ is a shorthand for a list of instructions whose effect is to concatenate a_i on the right end of the word in Y'. The first loop instruction stores in C the concatenation of X with itself. In the second loop after the first $|X|$ steps of computation in Y' is stored the reverse word of X. M is a list of instructions whose effect is to copy the content of Y' in Z, Z_1, Z_2 and Z_3 and the content of X in Y and Y_1, but only at the $(|X|+1)$st step of computation."

```
LOOP← Y
1. X←X;
2. X←X;
3. T←e;
   ⋮
n. X←X;
END;
LOOP← Z
1. T←e;
   ⋮
n. X←X;
END;
LOOP← Y1
1. X←X;
2. X←X;
3. T←e;
4. X←X;
   ⋮
n. X←X;
END;
LOOP← Z1
1. T←e;
2. X←X;
   ⋮
n. X←X;
END;
LOOP← Z2
1. T←e;
2. X←X;
   ⋮
n. X←X;
END;
LOOP← Z3
1. X←X;
2. T←e;
3. X←X;
   ⋮
n. X←X;
END;
OUT T
```

where

$M = X' \leftarrow \text{DELF}(X')$;
$\quad \text{LOOP}^{\leftarrow}\ X'$
$\quad$ 1. $A \leftarrow e$;
$\quad \vdots$
$\quad$ n. $A \leftarrow e$;
$\quad \text{END}$;
$\quad \text{LOOP}^{\leftarrow}\ A$
$\quad$ 1. $Z \leftarrow X; Z_1 \leftarrow X; Z_2 \leftarrow X; Z_3 \leftarrow X; Y \leftarrow Y'; Y_1 \leftarrow Y'; A' \leftarrow e;$
$\quad\quad X' \leftarrow X;$
$\quad \vdots$
$\quad$ n. $X \leftarrow X$;
$\quad \text{END}$;

In order to convince ourself that P really computes f let us consider the situation of the registers Y, Y_1, Z, Z_1, Z_2, Z_3 after the execution of the loop instruction with control register C. Let $X \neq a_1^n a_2^n a_3^n$ and let us denote the number of the occurrences of a_i in x by n_i. We have the following five cases:

$$\left.\begin{array}{lll}
(1) & n_1 < n_2, & a_1 \text{ occurs in } Z_3 \text{ or } a_1 \text{ occurs in } Z_2 \\
(2) & n_2 < n_1, & a_1 \text{ occurs in } Z_2 \\
(3) & n_1 > n_3, & Z_1 \text{ contains } a_1 \\
(4) & n_1 < n_3, & Y_1 \text{ contains } a_3
\end{array}\right\} \quad \text{if } X = a_1^{n_1} a_2^{n_2} a_3^{n_3},$$

$$\begin{array}{lll}
(5) & n_1 = n_2 = n_3, & \text{(a) } x = wa_1 a_3^m \text{ or } x = wa_2 a_3^m \text{ with } m < n_3, \\
 & & \quad Y \text{ contains } a_3, \\
 & & \text{(b) } x = a_1^m a_2 w \text{ or } x = a_1^m a_3 w \text{ with } m < n_1, \\
 & & \quad Z \text{ contains } a_1.
\end{array}$$

In all these cases the program puts e in T. Moreover, if x contains an occurrence of a_j with $j \neq 1, 2, 3$, then e is put in T as well. At the end T contains a_1 if and only if $x = a_1^n a_2^n a_3^n$ for $n \geq 0$.

The proof for the class $M_{1,1}^{\rightarrow R}$ follows by the closure of the class of deterministic context-free languages with respect to the reversal and by Lemma 2.5(4).

Proof of Theorem 4.13. The proof is analogous to that of Theorem 4.11, but we suppose now that the top of the stack of the push-down transducer to be considered, is the rightmost symbol. Let us consider the following program $P \in M_{1,1}^{\rightarrow R}$ on the same alphabet as the program of Theorem 4.11:

$\text{IN}\ X$
$H \leftarrow e; H \leftarrow X, B \leftarrow e; C \leftarrow e;$

LOOP X

$\quad 1.\ H \leftarrow Ha_{n+1}; H \leftarrow Ha_{n+1};$

$\quad \vdots$

$\quad n.\ H \leftarrow Ha_{n+1}; H \leftarrow Ha_{n+1};$

$\quad n+1.\ B \leftarrow Ba_1;$

$\quad \vdots$

$n+s+1.\ B \leftarrow Ba_1;$

END;

$H \leftarrow Ha_{n+1};$

LOOP H

$\quad 1.\ H \leftarrow \underbrace{Ha_{n+1}; \ldots;}_{h} H \leftarrow Ha_{n+1};$

$\quad \vdots$

$\quad n.\ H \leftarrow \underbrace{Ha_{n+1}; \ldots;}_{h} H \leftarrow Ha_{n+1};$

$\quad n+1.\ X \leftarrow X;$

$\quad \vdots$

$n+s+1.\ X \leftarrow X;$

END;

LOOP$^{\leftarrow}$ B

$\quad 1.\ H \leftarrow e;$

$\quad 2.\ X \leftarrow X;$

$\quad \vdots$

$n+s+1.\ X \leftarrow X;$

END;

"H contains $xa_{n+1}^{(h+1)(2x+1)}$ if $x \in \{c_1, \ldots, c_n\}$ and e otherwise."

$Q_0, W, A', R', S \leftarrow e; Q_0 \leftarrow Q_0 a_1; W \leftarrow WZ_0;$

LOOP$^{\rightarrow}$ H

$\quad 1.\ C \leftarrow S_1(C);$

$\quad \vdots$

$\quad n.\ C \leftarrow S_n(C);$

$\quad n+1.\ I;$

$\quad n+2.\ X \leftarrow X;$

$\quad \vdots$

$n+s+1.\ X \leftarrow X;$

END;

$T' \leftarrow e;$

LOOP$^{\rightarrow}$ Q_{j_1}

$\quad 1.\ T' \leftarrow T;$

$$
\begin{array}{rl}
2. & X \leftarrow X; \\
& \vdots \\
n+s+1. & X \leftarrow X;
\end{array}
$$

END;

$\vdots$

LOOP$^{\rightarrow}$ Q_{j_t}

$$
\begin{array}{rl}
1. & T' \leftarrow T; \\
2. & X \leftarrow X; \\
& \vdots \\
n+s+1. & X \leftarrow X;
\end{array}
$$

END;

OUT T'

where I is a list of instructions similar to those of the first branch of the loop on H in the program of Theorem 4.11, with these differences: C is used instead of X, C and W are scanned leftward, using the LAST and DELL lists of instructions, suitable outputs are written in T by $I^{e}_{A_i}$ and I^{i}_{j} $(0 \leq i \leq s,\ 1 \leq j \leq n)$ and, obviously, R-append and r-iteration replace L-append and ℓ-iteration.

An equivalent program $P \in M^{\leftarrow 1}_{1,1}$ can easily be written.

As regards the strictness of the containment note that the function $f: \Sigma^* \rightarrow \Sigma^*$ such that $f(x) = a_1^{|x|} a_2^{|x|} a_3^{|x|}$ belongs to $L_1^{\rightarrow R}$ but it cannot be defined by any deterministic push-down transducer.

Acknowledgment

The authors would like to thank Prof. A. Maggiolo-Schettini for his careful reading of the manuscript and many interesting discussions.

References

[1] G. Asser, Rekursive wortfunktionen, *Zeitschr.f.math. Logik und Grundl. d.Math.* **6** (1960) 258–278.

[2] G. Ausiello and M. Moscarini, On the complexity of decision problems for classes of simple programs on strings, *GI-6 Jahresstagung, Informatik Fachberichte* **5** (Springer, Berlin, 1976) pp. 148–163.

[3] M.P. Chytil and V. Jakl, Serial composition of 2-way finite-state transducers and simple programs on strings, *Proc. ICALP Conf. in Turku*, Lecture Notes in Comput. Sci. **52** (Springer, Berlin, 1977) pp. 135–147.

[4] S. Eilenberg and C.C. Elgot, *Recursiveness* (Academic Press, New York, 1970).

[5] E. Fachini and A. Maggiolo-Schettini, A hierarchy of primitive recursive sequence functions, *RAIRO Inform.Théor.* **13** (1979) 49–67.

[6] E. Fachini and A. Maggiolo-Schettini, Comparing hierarchies of primitive recursive sequence functions, *Zeitschr.f.math.Logik und Grundl.d.Math.* **28** (5) (1982) 431–445.

[7] G. Germano and A. Maggiolo-Schettini, Quelques caractérisation des fonctions récursives partielles, *C.R.Acad.Sci. Paris* **276** (1973) 1325–1327.

[8] S. Ginsburg and S.A. Greibach, Deterministic context-free languages, *Inform. Control* **9** (1966) 620–648.
[9] B. Goetze and W. Nehrlich, The number of loops necessary and sufficient for computing simple functions, *Elektr. Inform. Kybernet.*, to appear.
[10] E.M.Gurari, The equivalence problem for deterministic two-way sequential transducers is decidable, *SIAM J. Comput.* **11** (1982) 448–452.
[11] F.W. Von Henke, K. Indermark, G. Rose and K. Weihrauch, On primitive recursive wordfunctions, *Comput.* **15** (1975) 217–234.
[12] F.W. v. Henke, K. Indermark and K. Weihrauch, Hierarchies of primitive recursive wordfunctions and transductions defined by automata, in: M. Nivat, ed., *Automata, Languages and Programming* (North-Holland, Amsterdam, 1972) pp. 549–562.
[13] H. Huwig and V. Claus, Das Aquivalenz problem für Spezielle Klassen von LOOP Programmen, *Theoret. Comput. Sci. 3rd GI Conf.*, Lecture Notes in Comput. Sci. **48** (Springer, Berlin, 1977) pp. 73–82.
[14] K. Indermark, Push-down transductions as primitive recursive wordfunctions, *Proc. IRIA Sem.*, 1972.
[15] G. Rose and K. Weihrauch, Eine Charakterisierung der Klassen L_1 und R_1 primitive rekursiver Wortfunktionen, *GMD Bericht* **63** (1973).