Complete Symmetry in D2L Systems and Cellular Automata*

Peter R.J. Asveld

Department of Computer Science, Twente University of Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands

We introduce completely symmetric D2L systems and cellular automata by means of an additional restriction on the corresponding classes of symmetric devices. Then we show that completely symmetric D2L systems and cellular automata are still able to simulate Turing machine computations. As corollaries we obtain new characterizations of the recursively enumerable languages and of some space-bounded complexity classes.

KEY WORDS: D2L system, cellular automaton, symmetry, r.e. language, space-bounded complexity class.
C.R. CATEGORIES: F.4.3, F.1.1, F.1.3.

1. Introduction

Lindenmayer or L systems have originally been introduced to model the development of one-dimensional filaments by means of linear arrays of cells, each of which may exchange information with its n closest neighbors (n is a fixed natural number). For examples from developmental biology and for a mathematical treatment of the subject we refer to [9] (particularly, to Chapter 0 and Part III) and to [13], respectively.

In so-called D2L systems n equals 2, and the offspring w of a cell depends on its present state x as well as the present states x_l and x_r of its left and right neighbors, respectively. Thus it can be described by a function f with $f(x_l,x,x_r) = w$. Both in [1] and in [6] it has been shown independently that D2L systems are able to simulate Turing machine computations. This implies that for each recursively enumerable language L_0, there exists a D2L system G and a dgsm mapping g such that $L_0 = g(L(G))$.

In [7,8] (cf. also Sections 17.1 and 17.2 of [9]) similar results have been established for symmetric D2L systems, i.e., D2L systems that are both internal symmetric (Each offspring equals its reversal or mirror image. Or, equivalently, each offspring is a palindrome.) and external symmetric (For all possible states x_l, x and x_r we have $f(x_l,x,x_r) = f(x_r,x,x_l)$). The latter condition models the case in which a cell receives some information from its neighbors but regardless of orientation. For additional biological motivation the reader should consult [7,8,9,14]. Mathematically, it means that f is invariant under a specific permutation of its arguments.

From a mathematical point of view the extension to the notion of complete symmetry is obvious: we call a D2L system *completely external symmetric* if f is invariant under every permutation of its arguments. And it is *completely symmetric* when it is both internal symmetric and completely external symmetric.

The aim of the present paper is to show that the above-mentioned characterization of the r.e. (or recursively enumerable) languages remains valid when we restrict our attention to completely symmetric D2L systems (Section 3) or to the related class of accepting devices, the completely

* This paper appeared in *International Journal of Computer Mathematics* **19** (1986) pp. 211-223.

symmetric (one-dimensional) cellular automata (Section 4). Due to a result of Engelfriet and Rozenberg in [3] a further simplification of these characterizations is possible.

2. Definitions

For all unexplained notation and terminology on formal languages, such as e.g. *dgsm mapping* (mapping induced by a deterministic generalized sequential machine with accepting states), λ-*free nft transducer* (λ-free nondeterministic finite state transducer or λ-free a-transducer; λ is the empty word) and *reversal* or *mirror operation*, we refer to standard texts like [4,10,11,12,15].

Turing machines may be defined in several ways. For our present purpose a slightly modified formulation in terms of a rewriting system from [15] happens to be convenient.

Definition 2.1. A (single-tape) *nondeterministic Turing machine* or *NTM* $T = (Q, V, \Sigma, q_0, Q_F, P)$ is a rewriting system consisting of

- a set Q of states that contains an initial state q_0 and that includes a subset Q_F of final or accepting states;
- an alphabet V that includes a subalphabet Σ of input symbols;
- a set P of productions over the alphabet $V \cup Q \cup \{\$^L, \$^R\}$, i.e., P is a relation over $(V \cup Q \cup \{\$^L, \$^R\})^+$, where $\$^L$ and $\$^R$ are two special symbols, called left and right end-marker respectively, that are not in $V \cup Q$.

Each production in P has one of the following forms:

(2.1.1)	$pa \to qb$	overprint
(2.1.2)	$pc \to cq$	move right
(2.1.3)	$cpd \to qcd$	move left
(2.1.4)	$p\$^R \to qa\$^R$	extend to the right
(2.1.5)	$p\$^L \to \$^L qa$	extend to the left
(2.1.6)	$pa\$^R \to q\$^R$	reduce to the right
(2.1.7)	$\$^L pa \to q\$^L$	reduce to the left

where $p,q \in Q$; $a,b \in V$; $c \in V \cup \{\$^L\}$, and $d \in V \cup \{\$^R\}$.

Let $\Rightarrow$ be the derivation relation induced by P (cf. [15] for a formal definition) and $\Rightarrow^*$ its reflexive and transitive closure. The language $L(T)$ over Σ accepted by T is defined by

$$L(T) = \{a_1 a_2 ... a_n \mid \$^L q_0 a_1 a_2 ... a_n \$^R \Rightarrow^* \$^L b_1 b_2 ... b_i q b_{i+1} ... b_m \$^R, \text{ for some } b_1, b_2, ..., b_m \in V (m \geq 0) \text{ and some } q \in Q_F;\ a_1 a_2 ... a_n \in \Sigma^*\}.$$

A NTM $T = (Q, V, \Sigma, q_0, Q_F, P)$ with P a partial function on $(V \cup Q \cup \{\$^L, \$^R\})^+$ is called a *deterministic Turing machine* or *DTM*. With each Turing machine T and each input x over Σ we associate the set $C(T,x)$ of tape contents derivable from x; formally,

$$C(T,x) = \{b_1 ... b_m \mid \$^L q_0 x \$^R \Rightarrow^* \$^L b_1 ... b_i q b_{i+1} ... b_m \$^R \text{ for some } b_1, ..., b_m \in V (m \geq 0) \text{ and some } q \in Q\}.$$ □

We use Turing machines as acceptors and as generators of r.e. sets; cf. e.g. the proofs of Theorem 4.2 and Theorem 3.1, respectively.

From the different possibilities to define D2L systems we choose the approach taken in [18]. Let λ denote the empty word.

Definition 2.2. A *deterministic 2-Lindenmayer* or *D2L system* is a 3-tuple $G = (V, f_0, w)$ where

- V is an alphabet,
- w is the initial word ($w \in V^+$),
- f_0 is a total function from $(V \cup \{\lambda\}) \times V \times (V \cup \{\lambda\})$ into V^*.

The function f_0 induces a total mapping $f : V^* \to V^*$ as follows: $f(\lambda) = \lambda$, and for each $k \geq 1$, $f(v) = v'$ if and only if $v = \alpha_1 \alpha_2 \cdots \alpha_k$ ($\alpha_1, \alpha_2, \ldots, \alpha_k \in V$), $v' = v_1 v_2 \ldots v_k$ and for all i ($1 \leq i \leq k$), $f_0(\alpha_{i-1}, \alpha_i, \alpha_{i+1}) = v_i$ where we take α_j equal to λ for all j such that $j \leq 0$ or $j \geq k+1$. As usual $f^0(v) = v$ and $f^{i+1}(v) = f(f^i(v))$ for all v and all $i \geq 0$. The language $L(G)$ generated by G is defined by $L(G) = \{f^i(w) \mid i \geq 0\}$. □

Deterministic cellular automata (DCA) may be considered as acceptors corresponding to D2L systems. In Definition 2.3 we slightly deviate from standard definitions [12] by omitting (irrelevant semiinfinite sequences of) quiescent states, which yields even a closer relationship between DCA's and D2L systems.

Definition 2.3. A (one-dimensional) *deterministic cellular automaton* or *DCA* is a 4-tuple $C = (K, \Sigma, f_0, F)$ where

- K is a finite set of states that includes a set F of final states,
- Σ is the set of input symbols or input states ($\Sigma \subseteq K$),
- $f_0 : (K \cup \{\lambda\}) \times K \times (K \cup \{\lambda\}) \to K$ is a total function.

The function f_0 induces a total mapping $f : K^* \to K^*$ in the same way as in the previous definition.

The language $L(C)$ accepted by a DCA C is defined by

$$L(C) = \{x \in \Sigma^* \mid f^i(x) \in K^* F K^* \text{ for some } i \geq 0\}.$$ □

The following notions originated from [7]; cf. also [9]. Although introduced for D2L systems they obviously apply to DCA's too. A D2L system G [or DCA C] is called *internal symmetric* if for all $(\alpha_1, \alpha_2, \alpha_3)$ from $(V \cup \{\lambda\}) \times V \times (V \cup \{\lambda\})$ [from $(K \cup \{\lambda\}) \times K \times (K \cup \{\lambda\})$, respectively] we have $f_0(\alpha_1, \alpha_2, \alpha_3) = v$ implies that v is a palindrome, i.e., $v = \rho(v)$ where ρ is the reversal or mirror operation. It is called *external symmetric* if for all $\alpha_1, \alpha_2, \alpha_3$ we have $f_0(\alpha_1, \alpha_2, \alpha_3) = f_0(\alpha_3, \alpha_2, \alpha_1)$. And it is *symmetric* if it is both internal and external symmetric.

In this paper we call a D2L system G or a DCA C *completely external symmetric* if for all $\alpha_1, \alpha_2, \alpha_3$ and for all permutations π in S_3,

$$f_0(\alpha_1, \alpha_2, \alpha_3) = f_0(\alpha_{\pi(1)}, \alpha_{\pi(2)}, \alpha_{\pi(3)})$$

holds, where S_3 is the symmetric group on $\{1,2,3\}$. If, in addition, G or C is also internal symmetric, it is called *completely symmetric*.

For each completely (external) symmetric D2L system or DCA we write $f_0[\alpha_1, \alpha_2, \alpha_3] = v$ as an abbreviation of

$$f_0(\alpha_{\pi_1(1)}, \alpha_{\pi_1(2)}, \alpha_{\pi_1(3)}) = \cdots = f_0(\alpha_{\pi_6(1)}, \alpha_{\pi_6(2)}, \alpha_{\pi_6(3)}) = v$$

with $\{\pi_1, \ldots, \pi_6\} = S_3$.

3. Main Results

In this section we prove that completely symmetric D2L systems are able to simulate Turing machine computations. More precisely, each r.e. language L_0 can be obtained by applying a rather simple dgsm to $L(G)$ where G is a completely symmetric D2L system that depends on L_0.

Theorem 3.1. *For each r.e. language L_0, there exist a dgsm mapping g and a completely symmetric D2L system G such that $L_0 = g(L(G))$.*

Proof: According to the proof of Theorem 5.1 in [18] there exists for each r.e. language L_0 over Σ_0, a DTM $T_0 = (Q, V_0, \varnothing, q_0, Q_F, P)$ with $\Sigma_0 \subseteq V_0$ such that $L_0 = \Sigma_0^* \cap C(T_0, \lambda)$. Notice that the only possible input of T_0 is λ since $\varnothing^* = \{\lambda\}$, and that T_0 never halts if L_0 happens to be infinite.

We construct a dgsm mapping g and a completely symmetric D2L system $G = (V, f_0, w)$ that simulate T_0 in the sense that $g(L(G)) = \Sigma_0^* \cap C(T_0, \lambda) = L_0$. Define

$$V = Q \times (V_0 \cup \{\$^L, \$^R\}) \cup \{\alpha^X \mid \alpha \in V_0 \cup \{\$\}, X \in D\} \cup$$
$$\cup \{(q_X, \alpha) \mid q \in Q,\ \alpha \in V_0 \cup \{\$^Y\};\ X, Y \in D \text{ with } X \neq Y\}$$

where $D = \{L, R\}$. To each tape

$$\$^L b_1 \ldots b_i p b_{i+1} \ldots b_m \$^R \tag{1}$$

of T_0, there is a string of the form

$$\$^L \$^L b_1^L b_1^L \ldots b_i^L b_i^L (p, b_{i+1})(p, b_{i+1}) b_{i+2}^R b_{i+2}^R \cdots b_m^R b_m^R \$^R \$^R \tag{2}$$

in $L(G)$, but $L(G)$ contains other ''intermediate'' strings too; see below. So each symbol on the tape has been doubled and provided with a superscript that indicates whether its position is to the left or to the right of T_0's head position. T_0's head position itself is represented by a pair $(p, b_{i+1})(p, b_{i+1})$, in which p is the present state of T_0 and b_{i+1} is the symbol under the head. As the input of T_0 is λ, we define the initial word w of G by $w = \$^L \$^L (q_0, \$^R)(q_0, \$^R)$, accordingly. In essence the construction of G is simple. We have doubled the symbols on T_0's tape and we let the majority of three consecutive cells determine the next state of the middle cell. The main complication occurs when T_0 makes a head move into direction $X \in D$. Since all tape symbols have been doubled a head move requires now two steps (via an ''intermediate'' string).

The definition of f_0 consists of two parts: we define f_0 for

(1) the symbols not in the close neighborhood of $(p, b_{i+1})(p, b_{i+1})$; viz.

$f_0[\alpha^X, \alpha^X, \beta^X] = \alpha^X$ for all $\alpha, \beta \in V_0 \cup \{\$\}, X \in D$.

(This definition, as many of the following ones, includes cases that will never occur in the simulation of T_0; e.g. $f_0(\$^R, \$^R, b^R) = \$^R$. They are only included to meet the complete symmetry condition; however, in no way they affect the simulation of T_0 by G).

(2) the symbols in the close neighborhood of $(p, b_{i+1})(p, b_{i+1})$: f_0 depends on the value of (p, b_{i+1}), viz. on the production in P that applies to ""$\ldots p b_{i+1} \ldots$. So we distinguish seven cases (cf. Definition 2.1):

(2.1.1) $f_0[(p,a),(p,a),\alpha^X] = (q,b)$
$f_0[(p,a),\alpha^X,\alpha^X] = \alpha^X$
iff $pa \to qb$ is in P

(2.1.2) $f_0[\alpha^L, \alpha^L, (p,c)] = \alpha^L$
$f_0[(p,c),\beta^R,\beta^R] = (q_R, \beta)$

$f_0[\alpha^L,(p,c),(p,c)] = f_0[(p,c),(p,c),\beta^R] = f_0[c^L,c^L,(q_R,\beta)] = c^L$
$f_0[c^L,(q_R,\beta),\beta^R] = f_0[(q_R,\beta),\beta^R,\gamma^R] = (q,\beta)$
iff $pc \rightarrow cq$ is in P

(2.1.3) $f_0[(p,d),\beta^R,\beta^R] = \beta^R$
$f_0[c^L,c^L,(p,d)] = (q_L,c)$
$f_0[(p,d),(p,d),\beta^R] = f_0[c^L,(p,d),(p,d)] = f_0[(q_L,c),d^R,d^R] = d^R$
$f_0[c^L,(q_L,c),d^R] = f_0[\alpha^L,c^L,(q_L,c)] = (q,c)$
iff $cpd \rightarrow qcd$ is in P

(2.1.4) $f_0[\alpha^L,\alpha^L,(p,\$^R)] = \alpha^L$
$f_0[\alpha^L,(p,\$^R),(p,\$^R)] = (q,a)(q,a)$
$f_0[(p,\$^R),(p,\$^R),\lambda] = \$^R\$^R$
iff $p\$^R \rightarrow qa\$^R$ is in P

(2.1.5) $f_0[(p,\$^L),\beta^R,\beta^R] = \beta^R$
$f_0[(p,\$^L),(p,\$^L),\beta^R] = (q,a)(q,a)$
$f_0[\lambda,(p,\$^L),(p,\$^L)] = \$^L\$^L$
iff $p\$^L \rightarrow \$^L qa$ is in P

(2.1.6) $f_0[(p,a),\$^R,\$^R] = (q_R,\$^R)$
$f_0[\alpha^L,\alpha^L,(p,a)] = f_0[\alpha^L,\alpha^L,(q_R,\$^R)] = \alpha^L$
$f_0[\alpha^L,(p,a),(p,a)] = f_0[(p,a),(p,a),\$^R] = \lambda$
$f_0[\alpha^L,(q_R,\$^R),\$^R] = f_0[(q_R,\$^R),\$^R,\lambda] = (q,\$^R)$
iff $pa\$^R \rightarrow q\$^R$ is in P

(2.1.7) $f_0[\$^L,\$^L,(p,a)] = (q_L,\$^L)$
$f_0[(p,a),\beta^R,\beta^R] = f_0[(q_L,\$^L),\beta^R,\beta^R] = \beta^R$
$f_0[(p,a),(p,a),\beta^R] = f_0[\$^L,(p,a),(p,a)] = \lambda$
$f_0[\$^L,(q_L,\$^L),\beta^R] = f_0[\lambda,\$^L,(q_L,\$^L)] = (q,\$^L)$
iff $\$^L pa \rightarrow q\$^L$ is in P.

Defining f_0 in this way implies that each string of the form (2) - which corresponds to a tape contents (1) of T_0 - will be transformed in either one step (viz. when (2.1.1), (2.1.4) or (2.1.5) applies) or in two steps (if (2.1.2), (2.1.3), (2.1.6) or (2.1.7) is applicable) into a similar string that corresponds to T_0's new tape contents. Of both cases we show an example.

Firstly, assume that $pb_{i+1} \rightarrow qb$ is in P. Then in a single step (2) will be changed into

$$\$^L\$^L b_1^L b_1^L ... b_i^L b_i^L (q,b)(q,b) b_{i+2}^R b_{i+2}^R ... b_m^R b_m^R \$^R\$^R.$$

Secondly, we assume that $b_i pb_{i+1} \rightarrow qb_i b_{i+1}$ is in P. In this case (2) is transformed into the intermediate string

$$\$^L\$^L b_1^L b_1^L b_1^L ... b_i^L (q_L,b_i) b_{i+1}^R b_{i+1}^R ... b_m^R b_m^R \$^R\$^R$$

that in its turn is changed into

$$\$^L\$^L b_1^L b_1^L b_1^L ... (q,b_i)(q,b_i) b_{i+1}^R b_{i+1}^R ... b_m^R b_m^R \$^R\$^R.$$

The detailed definition of the dgsm g that maps strings of the form (2) into $b_1...b_m$ if and only if $b_i \in \Sigma$ for all $i (1 \leq i \leq m)$ as well as the correctness proof of the entire construction is

straightforward and left to the reader. □

Notice that g satisfies

$$|g(w)| = ½(|w|-4) \tag{3}$$

for each $w \in V^*$ such that $g(w) \neq \varnothing$ ($|w|$ denotes the length of the string w).

Using a result of Engelfriet and Rozenberg [3] we can improve on Theorem 3.1 in the sense that there is a single completely symmetric D2L system from which each r.e. language can be obtained by applying an appropriate dgsm mapping. The price we have to pay for this uniformity is that this dgsm mapping is much more complicated than the one in Theorem 3.1; in particular it no longer possesses property (3).

Theorem 3.2. *There exists a completely symmetric D2L system G such that for each r.e. language L_0 there exists a dgsm mapping g_0 with $L_0 = g_0(L(G))$.*

Proof: According to Theorem 13(iii) in [3] there is an r.e. language L_1 such that for each r.e. language L_0 there exists a dgsm mapping g_1 with $L_0 = g_1(L_1)$. Let G be the completely symmetric D2L system and g the dgsm mapping as in the proof of Theorem 3.1 such that $L_1 = g(L(G))$. Then for each r.e. language L_0 we have $L_0 = g_1(g(L(G)))$. Since dgsm mappings are closed under composition [2], the statement follows with $g_0(w) = g_1(g(w))$ for each w.

4. Applications

Basic in dealing with cellular automata is the notion of simulation. The following definition is a variation of a concept studied in [17].

Definition 4.1. A DCA $C = (K, \Sigma, f_0, F)$ *simulates* a DTM $T = (Q, V, \Sigma, q_0, Q_F, P)$ *in at most k times real-time* ($k \geq 1$) if there exist mappings $s_1: V^* \to K^*$, and $s_2: K^* \to V^*$, such that for all $x, y \in V^*$: $x \Rightarrow y$ holds if and only if

$$y = s_2(f^i(s_1(x)))$$

for some i ($1 \leq i \leq k$). □

Of course, one wants to keep the (de)coding functions s_1 and s_2 as simple as possible. In this paper we restrict our attention to dgsm mappings.

Theorem 4.2. *(1) Each DTM can be simulated by a completely symmetric DCA in at most* 2 *times real-time. Consequently, for each r.e. language L_0 there exist a dgsm mapping g and a completely symmetric DCA C such that $L_0 = g(L(C))$.*

(2) There exists a completely symmetric DCA C such that for each r.e. language L_0 there exists a dgsm mapping g_0 with $L_0 = g_0(L(C))$.

Proof: Let T be a DTM that acts as an acceptor (For the simulation of T by a not necessarily completely symmetric DCA we refer to [16]). If x is the string that corresponds to the input of T and y satisfies $x \Rightarrow^* y$, then y is transformed by s_1 into a string of the form (2); cf. the proof of Theorem 3.1. The function f_0 is defined as in the proof of Theorem 3.1 (F equals $Q_F \times V$). Finally, s_2 transforms a string of the form (2) into a string that corresponds to a tape contents of T. Hence the completely symmetric DCA C simulates T in at most 2 times real-time. The remaining part of (1) as well as (2) are proved in a similar way as in Section 3. □

Finally, we turn to space-bounded complexity classes. Henceforth, S is a function on the natural numbers that satisfies $S(n) \geq n$ for each $n \geq 1$. Let DSPACE(S) [NSPACE(S), respectively] be the family of languages accepted by [non]deterministic single-tape Turing machines that use at

most $S(n)$ tape cells during a computation on an input of length n.

A (one-dimensional) *nondeterministic cellular automaton* or *NCA* C is a 4-tuple $C=(K,\Sigma,f_0,F)$ where K,Σ and F are as in Definition 2.3 but f_0 is now a function from $(K\cup\{\lambda\})\times K\times(K\cup\{\lambda\})$ to the (finite) subsets of K. The language $L(C)$ accepted by C is defined by $L(C)=\{x\in\Sigma^* \mid f^i(x)\cap K^*FK^*\neq\emptyset \text{ for some } i\geq 0\}$.

A DCA [NCA, respectively] $C=(K,\Sigma,f_0,F)$ is *S-space-bounded* if for each $i\geq 0$, $|f^i(x)|\leq S(|x|)$ [$y\in f^i(x)$ implies $|y|\leq S(|x|)$]. Let DCASPACE(S) [NCASPACE(S), respectively] be the family of languages accepted by S-space-bounded DCA's [NCA's].

Lemma 4.3. *(1)* NCASPACE(n) *equals the family of context-sensitive languages,*

(2) DCASPACE(n) *equals the family of deterministic context-sensitive languages,*

(3) NCASPACE(n^2) *equals the family of two-way nondeterministic nonerasing stack automaton languages,*

(4) DCASPACE($n\log n$) *equals the family of two-way deterministic nonerasing stack automaton languages.*

Proof: The equalities DSPACE(n)=DCASPACE(n) and NSPACE(n)=NCASPACE(n) have been established in [12]. In a similar fashion one can prove NSPACE(n^2)=NCASPACE(n^2) and DSPACE($n\log n$)=DCASPACE($n\log n$). For the characterization of the former classes in the latter two equalities in terms of nonerasing stack automaton languages we refer to [10]. □

Theorem 4.4. *(1) For each deterministic context-sensitive* [*two-way deterministic nonerasing stack automaton*] *language* L_0 *there exist a completely symmetric linear space-bounded* [$n\log n$*-space-bounded*] *DCA* C *and a dgsm mapping* g *such that* $L_0=g(L(C))$.

(2) For each context-sensitive [*two-way nondeterministic nonerasing stack automaton*] *language* L_0 *there exist a completely symmetric linear space-bounded* [n^2*-space-bounded*] *NCA* C *and a dgsm mapping* g *such that* $L_0=g(L(C))$.

Proof: By Lemma 4.3 it suffices to simulate $S(n)$-space-bounded Turing machine computations by completely symmetric $S(n)$-space-bounded cellular automata.

(1) First, we reduce the space $S(n)$ of the DTM T to $\lceil \tfrac{1}{2}S(n)\rceil$; cf. e.g. Theorem 12.1 in [10]. Then we apply the construction sketched in the proof of Theorem 4.2(1) (Since there seems to be no obvious way to linearly reduce the space of a completely symmetric DCA, the reduction is performed on T in advance).

(2) The nondeterministic case is a bit more complicated. Without loss of generality we assume that all T's nondeterminism is concentrated in the productions of the form (2.1.1). In other words, if we remove all productions of the form (2.1.1) from T, then the resulting Turing machine is deterministic. The next step is a modification of the construction outlined in the proof of Theorem 4.2(1).

Due to nondeterminism a string over K of the form

$$.....(p,a)(p,a)..... \tag{4}$$

may be transformed by f into a word

$$.....(q,b)(r,c)..... \tag{5}$$

with $q\neq r$ or $b\neq c$. To eliminate these undesirable ''diverging'' transformations we add a new (dead alley) symbol Δ to K, while we extend the definition of f_0 by

$$f_0[(q,b),(r,c),\alpha^X]=\Delta \quad \text{iff } q\neq r \text{ or } b\neq c$$

$f_0[\Delta,\alpha,\beta] = \Delta \qquad \alpha,\beta \in K \cup \{\Delta\}.$

This extended definition of f_0 guarantees that words of the form (5) will only yield strings that contain at least one occurrence of Δ. Since $g(w)$ is undefined for strings w that contain a Δ, transformations from (4) to (5) no longer harm. □

We are also able to prove a uniform analogue of Theorem 4.4. However, now we need λ-free nft transductions rather than dgsm mappings.

Theorem 4.5. *(1) There exists a completely symmetric linear* [$n\log n$-] *space-bounded DCA C such that for each deterministic context-sensitive* [*two-way deterministic nonerasing stack automaton*] *language* L_0 *there exists a λ-free nft transduction g with* $L_0 = g(L(C))$.

(2) There exists a completely symmetric linear [n^2-] *space-bounded NCA C such that for each context-sensitive* [*two-way nondeterministic nonerasing stack automaton*] *language* L_0 *there exists a λ-free nft transduction g with* $L_0 = g(L(C))$.

Proof: The argument is similar to the one of Theorem 3.2. Instead of Theorem 13(iii) of [3] we use the main results of [5,19]. Thus g_1 is now a λ-free nft transduction. Composing a dgsm mapping that satisfies (3) with a λ-free nft transduction yields another λ-free nft transduction [2,4]. □

Acknowledgment . I am indebted to Ruud Sommerhalder who brought [17] to my notice.

References

1. D. van Dalen: A note on some systems of Lindenmayer, *Math. Systems Theory* **5** (1971) 128-140.
2. C.C. Elgot & J.E. Mezei: On relations defined by generalized finite automata, *IBM J. Res. Develop.* **9** (1965) 47-65.
3. J. Engelfriet & G. Rozenberg: Fixed point languages, equality languages, and representation of recursively enumerable languages, *J. Assoc. Comput. Mach.* **27** (1980) 499-518.
4. S. Ginsburg: Algebraic and Automata-Theoretic Properties of Formal Languages (1975), North-Holland, Amsterdam.
5. S. Ginsburg & G.F. Rose: On the existence of generators of certain AFL, *Inform. Sci.* **2** (1970) 431-446.
6. G.T. Herman: The computing ability of a developmental model for filamentous organisms, *J. Theor. Biology* **25** (1969) 421-435.
7. G.T. Herman: Models for cellular interactions in development without polarity of individual cells. Part I - General description and the problem of universal computing ability, *Internat. J. Systems Sci.* **2** (1971) 271-289.
8. G.T. Herman: Models for cellular interactions in development without polarity of individual cells. Part II - Problems of synchronization and regulation, *Internat. J. Systems Sci.* **3** (1972) 149-175.
9. G.T. Herman & G. Rozenberg: Developmental Systems and Languages (1975), North-Holland, Amsterdam.
10. J.E. Hopcroft & J.D. Ullman: Introduction to Automata Theory, Languages, and Computation (1979), Addison-Wesley, Reading, Mass.

11. G.E. Révész: Introduction to Formal Languages (1983), McGraw-Hill, New York.
12. A. Rosenfeld: Picture Languages: Formal Models for Picture Recognition (1979), Academic Press, New York.
13. G. Rozenberg & A. Salomaa: The Mathematical Theory of L Systems (1980), Academic Press, New York.
14. K. Ruohonen: A symmetric D2L model for string regeneration, pp. 263-270 in: A. Lindenmayer & G. Rozenberg (Eds.): ‘‘Automata, Languages, and Development’’ (1976), North-Holland, Amsterdam.
15. A. Salomaa: Formal Languages (1973), Academic Press, New York.
16. A.R. Smith III: Simple computation-universal cellular spaces, *J. Assoc. Comput. Mach.* **18** (1971) 339-353.
17. H. Szwerinski: Realzeitsimulation eindimensionaler Zellularautomaten durch Zellularautomaten mit symmetrischer lokaler Ueberfuehrungsfunktion, pp. 128-138 in R. Vollmar(Ed.): ‘‘Beitraege zur Theorie der Polyautomaten II’’(1982), Technische Universitaet Braunschweig, FGR.
18. P.M.B. Vitányi: Deterministic Lindenmayer languages, nonterminals and homomorphisms, *Theor. Comp. Sci.* **2** (1976) 49-71.
19. B. Wegbreit: A generator of context-sensitive languages, *J. Comput. System Sci.* **3** (1969) 456-461.