Theoretical Computer Science 40 (1985) 149–162
North-Holland

CTS SYSTEMS AND PETRI NETS

IJ.J. AALBERSBERG and G. ROZENBERG*

Institute of Applied Mathematics and Computer Science, University of Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands

Communicated by A. Salomaa
Received July 1984
Revised March 1985

Abstract. The general theory of coordinated table selective substitution systems (*cts systems* for short) (see Rozenberg (1985)), provides a unifying framework for a considerable number of grammar *and* automaton models considered in the literature. This paper is mainly devoted to the investigation of a natural subclass of cts systems (which uses the 'context-free grammar selector' for its memory access) and it turns out that this subclass closely corresponds to the Petri net model of concurrent processes.

Introduction

Selective substitution grammars (*s-grammars*, for short) (see, e.g., [10, 11]) provide a quite natural and useful framework for a general theory of rewriting systems (grammars). Roughly speaking, two basic components of an s-grammar G are: the set of context-free *productions* P and the *selector* K. This selector is a language over the alphabet $\Sigma \cup \bar{\Sigma}$, where Σ is the alphabet of G and $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$, $\Sigma \cap \bar{\Sigma} = \emptyset$. To *rewrite directly* a word x over Σ, one has to find a word $y \in K$ which differs from x only by the fact that some occurrences of letters from Σ in x are replaced by their barred ('activated') counterparts from $\bar{\Sigma}$. Then, *all* occurrences of letters in x that correspond to activated occurrences in y are rewritten in the usual fashion using productions from P. The *derivation* process consists of a finite number of iterations of the direct rewriting process and the *language* of G is defined in the usual way (using the intersection with Δ^*, where Δ is the terminal alphabet of G). If $K \subseteq \Sigma^* \bar{\Sigma} \Sigma^*$, then K (and, consequently, G) is called *sequential*. Perhaps the two most 'famous' sequential selectors are $\Sigma^* \bar{\Sigma}$ and $\Sigma^* \bar{\Sigma} \Sigma^*$; the first one is called *right-linear* (it underlies right-linear grammars) and the second one is called *0-sequential* (it underlies context-free grammars).

The framework of s-grammars was extended in [15] to the so-called coordinated table selective substitution systems (*cts systems*, for short); in this framework both grammars and automata can easily be modeled and investigated. Roughly speaking,

* This author was supported by NSF Grant MCS 83-05245.

a cts system G (and, in the terminology of [15], we will mainly consider sequential versions of them with one table on each coordinate) consists of n sequential s-grammars $G_1, \ldots, G_n$, $n \geq 1$, and a set R of *rewrites*, where $R \subseteq P_1 \times \cdots \times P_n$ and each P_i is the set of productions of G_i. In G one rewrites n-tuples of words rather than single words. Given an n-tuple $x = (x_1, \ldots, x_n)$, where each x_i is over the alphabet of G_i, it can be *directly rewritten* into an n-tuple $y = (y_1, \ldots, y_n)$ if R contains a rewrite $r = (r_1, \ldots, r_n)$ such that each x_i can be directly rewritten (in G_i) into y_i using r_i. Then, the *derivation* (*computation*) process consists of an iteration of the direct rewriting process.

In the modeling of automata by cts systems, it is often convenient to assume that G_1 (the grammar of the first coordinate) is a right-linear grammar, because this essentially corresponds to the quite natural and customary process of reading the input tape from left to right, one-way only.

In this paper we continue the systematic investigation of 2-coordinate models (i.e., $n = 2$) initiated in [2, 3]. Once—as discussed above—the first coordinate (*input*) is fixed as a right-linear grammar, two very natural choices for the selectors on the second coordinate (*memory*) are: right-linear selectors and 0-sequential selectors; as pointed out already, these selector types are very well understood when used in grammars (roughly speaking, grammars correspond to cts systems with one coordinate only). It is easily seen that using right-linear selectors as the memory access with erasing productions in the second grammar essentially yields pushdown automata (see, e.g., [2, 3]).

The main purpose of this paper is to demonstrate that using 0-sequential selectors as the memory access yields systems very closely related to Petri nets—a basic model of concurrent processes (see, e.g., [1, 13, 14]). We shall indicate how this relationship can be exploited to the advantage of both theories.

To put the results indicated above in a better perspective we also investigate (in Section 4) 2-coordinate cts systems where the first coordinate is a right-linear grammar and the second coordinate is a *0S²-grammar*, i.e., a grammar based on context-free productions but using the selector of the type $\Sigma^* \bar{\Sigma} \bar{\Sigma} \Sigma^*$, where Σ is the total alphabet involved. These systems are of independent interest since the used selector (called a *0-bisequential* selector) can be seen as forming the basis of the selector used in context-sensitive grammars.

0. Preliminaries

We assume the reader to be familiar with basic formal language theory, in particular basic grammar models (see, e.g., [16]) and with basic Petri net theory (see, e.g., [1, 13, 14]).

We mostly use standard notation and terminology; perhaps only the following points require some additional attention.

For a set A, $\# A$ denotes its cardinality. For sets A, B, $A - B$ denotes their difference. If $K_1, \ldots, K_n$, $n \geq 1$, is a sequence of sets, then $\bigtimes_{i=1}^{n} K_i$ denotes their

cartesian product. For a set A and a positive integer n, $A^{(n)}$ denotes the cartesian power.

Unless stated otherwise, we only consider finite nonempty alphabets. For a word x, $\#_a(x)$ denotes the number of occurrences of a in x and $alph(x)$ denotes the set of letters occurring in x. λ denotes the empty word.

Throughout this paper, barred versions of symbols are used with a 'special' reserved meaning. All symbols to be used are elements of an arbitrary but fixed infinite alphabet $\mathscr{A} \cup \bar{\mathscr{A}}$, where $\bar{\mathscr{A}} = \{\bar{a} \mid a \in \mathscr{A}\}$ and $\mathscr{A}$ and $\bar{\mathscr{A}}$ are disjoint. Whenever we consider an alphabet Σ and the alphabet $\bar{\Sigma} = \{\bar{a} \mid a \in \Sigma\}$, it is assumed that $\Sigma \subseteq \mathscr{A}$. Moreover, $iden_\Sigma$ denotes the homomorphism from $(\Sigma \cup \bar{\Sigma})^*$ into Σ^* defined by: $iden_\Sigma(\bar{a}) = a$ and $iden_\Sigma(a) = a$ for all $a \in \Sigma$.

A labeled marked Petri net, abbreviated lmPN, will be specified as a 6-tuple $\mathscr{P} = (P, T, F, \Sigma, l, M_0)$, where P is the set of places, T is the set of transitions, F is the flow relation, Σ is the alphabet, l is the labeling function (from T into $\Sigma \cup \{\lambda\}$) and M_0 is the initial marking such that there exists a $p \in P$ with $M_0(p) \neq 0$. For an lmPN $\mathscr{P} = (P, T, F, \Sigma, l, M_0)$, P, T, F, Σ, l, and M_0 will be denoted by $pl(\mathscr{P})$, $tr(\mathscr{P})$, $fl(\mathscr{P})$, $al(\mathscr{P})$, $lab(\mathscr{P})$, and $inm(\mathscr{P})$, respectively. The language of an lmPN $\mathscr{P}$, denoted by $L(\mathscr{P})$, is then the set of all 'labeled' firing-sequences of $\mathscr{P}$ from $inm(\mathscr{P})$ to the final zero-marking of $\mathscr{P}$ (denoted by $fzm(\mathscr{P})$); $L(\mathscr{P})$ is referred to as an lmPNfz language and the class of all lmPNfz languages is denoted by $\mathscr{L}$(lmPNfz).

1. Basic definitions

In this section we introduce the class of (RL; 0S)-systems, which forms a subclass of the (sequential) cts systems considered in [15].

Definition 1.1. (1) Let Σ be an alphabet. A *selector* (*over* Σ) is a subset of $(\Sigma \cup \bar{\Sigma})^* \bar{\Sigma} (\Sigma \cup \bar{\Sigma})^*$.

(2) A *table* is a triple $T = (\Sigma, h, K)$, where Σ is an alphabet, $h \subseteq \Sigma \times \Sigma^*$ is a finite nonempty set, and K is a selector over Σ. The alphabet Σ is referred to as the *alphabet of* T (denoted by $al(T)$), h is called the *set of productions of* T (denoted by $prod(T)$), and K is called the *selector of* T (denoted by $sel(T)$).

(3) Let $T = (\Sigma, h, K)$ be a table. For $x, y \in \Sigma^*$ we say that x *directly derives* y *in* T, denoted by $x \Rightarrow_T y$, if $x = b_1 \dots b_n$, $n \geq 1$, $b_1, \dots, b_n \in \Sigma$, $y = \beta_1 \dots \beta_n$, $\beta_1, \dots, \beta_n \in \Sigma^*$ and if there exists a $z \in K$, $z = a_1 \dots a_n$, $a_1, \dots, a_n \in \Sigma \cup \bar{\Sigma}$ such that $iden_\Sigma(z) = x$ and, for $1 \leq i \leq n$, if $a_i \in \Sigma$, then $b_i = \beta_i$ and if $a_i \in \bar{\Sigma}$, then $(b_i, \beta_i) \in h$. Furthermore, if $S = \{(b_i, \beta_i) \in h \mid a_i \in \bar{\Sigma} \text{ and } 1 \leq i \leq n\}$, then we also say that x *directly derives* y *in* T *using* S, denoted by $x \Rightarrow_T^S y$.

Note that if T is a table such that $sel(T) \subseteq (al(T))^* \overline{al(T)} (al(T))^*$ and if $x \Rightarrow_T^S y$ for some $x, y \in (al(T))^*$, then $S = \{s\}$ for some $s \in prod(T)$; we will write $x \Rightarrow_T^s y$ rather than $x \Rightarrow_T^{\{s\}} y$.

Definition 1.2. (1) A *right-linear grammar*, abbreviated *RL-grammar*, is a 5-tuple $G=(\Sigma, h, S, \Delta, K)$, where:

(a) (Σ, h, K) is a table, called the *table of* G and denoted by $tab(G)$,

(b) $\Delta \subseteq \Sigma$ is the *terminal alphabet of* G, denoted by $term(G)$; $\Sigma - \Delta$ is called the *nonterminal alphabet of* G and is denoted by $nterm(G)$,

(c) $S \in nterm(G)$ is the *axiom of* G, denoted by $ax(G)$,

(d) $(X, \alpha) \in h$ implies:

(i) $X \in nterm(G)$, and

(ii) $\alpha \in \Sigma \cup \{\lambda\} \cup (term(G) \cdot nterm(G))$, and

(e) $K = (term(G))^* \cdot \overline{(nterm(G))}$.

(2) A *0-sequential grammar*, abbreviated *0S-grammar*, is a 4-tuple $G=(\Sigma, h, S, K)$, where:

(a) (Σ, h, K) is a table, called the *table of* G and denoted by $tab(G)$,

(b) $S \in \Sigma$ is the *axiom of* G, denoted by $ax(G)$, and

(c) $K = \Sigma^* \bar{\Sigma} \Sigma^*$.

All the terminology and notations concerning tables carry over to RL- and 0S-grammars (through their tables) in the obvious way.

Furthermore, we will use the following notations. If G is an RL- or an 0S-grammar, then $al_\lambda(G)$ denotes the set $al(G) \cup \{\lambda\}$. If G is an RL-grammar, then $term_\lambda(G)$ denotes the set $term(G) \cup \{\lambda\}$ and $nterm_\lambda(G)$ denotes the set $nterm(G) \cup \{\lambda\}$.

Definition 1.3. (1) A *right-linear 0-sequential system*, abbreviated *(RL; 0S)-system*, is a triple $G=(G_1, G_2, R)$, where:

(a) G_1 is an RL-grammar,

(b) G_2 is an 0S-grammar, and

(c) $R \subseteq prod(G_1) \times prod(G_2)$ is referred to as the set of *rewrites of* G, denoted by $rew(G)$.

(2) Let $G=(G_1, G_2, R)$ be an (RL; 0S)-system.

(2.1) Let $x=(x_1, x_2)$, $y=(y_1, y_2) \in (al(G_1))^* \times (al(G_2))^*$. We say that x *directly derives* y *in* G, denoted by $x \Rightarrow_G y$, if there exists an $r=(r_1, r_2) \in R$ such that $x_1 \Rightarrow_{G_1}^{r_1} y_1$ and $x_2 \Rightarrow_{G_2}^{r_2} y_2$. We then say that x *directly derives* y *in* G *using* r and write $x \Rightarrow_G^r y$. As usual, $\Rightarrow_G^*$ is the reflexive transitive closure of $\Rightarrow_G$; if $x \Rightarrow_G^* y$, then we say that x *derives* y *in* G.

(2.2) The *language generated by* G, denoted by $L(G)$, is defined by $L(G) = \{w \in (term(G_1))^* | (ax(G_1), ax(G_2)) \Rightarrow_G^* (w, \lambda)\}$; $L(G)$ is referred to as an *(RL; 0S)-language*.

The class of all (RL; 0S)-languages is denoted by $\mathscr{L}$(RL; 0S).

Since one may view the effects of a derivation process in an (RL; 0S)-system on the second coordinate as (a special sort of) counting (of occurrences of symbols), one can establish a relationship between (RL; 0S)-systems and multicounter

automata (satisfying particular restrictions). In the last section we shall say more about this relationship.

Furthermore, the following notations turn out to be very useful. If $G=(G_1, G_2, R)$ is an (RL; 0S)-system and $r=((X, \sigma Y), (A, \alpha)) \in R$, where $X \in nterm(G_1)$, $Y \in nterm_\lambda(G_1)$, $\sigma \in term_\lambda(G_1)$, $A \in al(G_2)$, and $\alpha \in (al(G_2))^*$, then

$$lhs_1(r)=X, \qquad rhs_1(r)=\sigma Y,$$

$gt_1(r)=\sigma$ (gt abbreviates *g*enerated *t*erminal),

$gnt_1(r)=Y$ (gnt abbreviates *g*enerated *n*on*t*erminal),

$lhs_2(r)=A$ and $rhs_2(r)=\alpha$.

Clearly, without loss of generality (as far as the class of generated languages is concerned) we may and will assume that whenever we consider an (RL; 0S)-system $G=(G_1, G_2, R)$, $al(G_1)$ and $al(G_2)$ are disjoint.

2. Main theorem

In the previous sections we defined two classes of languages, namely $\mathscr{L}$(RL; 0S) and $\mathscr{L}$(lmPNfz). As the following theorem shows, these classes are equal.

Theorem 2.1. $\mathscr{L}(\text{RL; 0S}) = \mathscr{L}(\text{lmPNfz})$.

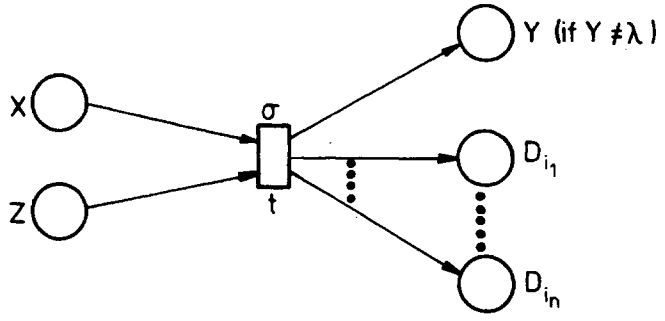
Proof. The proof consists of two steps, each taken care of by a lemma.

Lemma 2.2. $\mathscr{L}(\text{RL; 0S}) \subseteq \mathscr{L}(\text{lmPNfz})$.

Proof idea. For an arbitrary (RL; 0S)-system G we can construct an lmPN $\mathscr{P}$ with $L(G)=L(\mathscr{P})$ as follows. If $G=(G_1, G_2, R)$, then each letter of $nterm(G_1) \cup al(G_2)$ uniquely corresponds to a place of $\mathscr{P}$. Furthermore, each rewrite of R uniquely corresponds to a transition of $\mathscr{P}$. More specifically, R contains a rewrite r if and only if $\mathscr{P}$ contains a transition t labeled by $gt_1(r)$ with inputs $lhs_1(r)$ and $lhs_2(r)$ and outputs $gnt_1(r)$ (if $gnt_1(r) \neq \lambda$) and $alph(rhs_2(r))$. Hence, the use of a rewrite $((X, \sigma Y), (Z, \gamma))$ in G, where $\sigma \in term_\lambda(G_1)$ and $Y \in nterm_\lambda(G_1)$, uniquely corresponds to the firing of a transition in $\mathscr{P}$, which is labeled by σ, consumes one token from the place in $\mathscr{P}$ corresponding to X and one token from the place in $\mathscr{P}$ corresponding to Z and produces one token in the place in $\mathscr{P}$ corresponding to Y (if $Y \neq \lambda$) and, for all $A \in al(G_2)$, $\#_A(\gamma)$ tokens in the place in $\mathscr{P}$ corresponding to A. In this way the correspondence between letters of $nterm(G_1) \cup al(G_2)$ and tokens in appropriate places in $\mathscr{P}$ becomes obvious. This may be graphically

represented as follows:

$((X, \sigma Y), (Z, D_{i_1} \ldots D_{i_n})) \in R$ if and only if

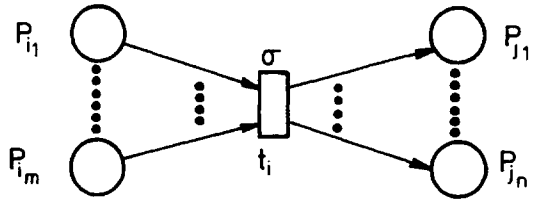


is a transition (with its input and output places, where output places appearing more than once mean multiple arcs) in $\mathscr{P}$.

The easy but tedious formal details concerning the above construction and the proof of the lemma are left to the reader. □

Lemma 2.3. $\mathscr{L}(\text{lmPNfz}) \subseteq \mathscr{L}(\text{RL; 0S})$.

Proof idea. For an arbitrary lmPN $\mathscr{P}$ we can construct an (RL; 0S)-system $G = (G_1, G_2, R)$ with $L(\mathscr{P}) = L(G)$ as follows. Each place of P uniquely corresponds to an element of $al(G_2) - \{S_2\}$, where S_2 is a distinguished element of $al(G_2)$, and the number of tokens in each place (at a step in the firing process) equals the number of occurrences of the corresponding symbol on the second coordinate (from the corresponding step in the derivation process). Furthermore, each transition uniquely corresponds to a subset of rewrites of $R - \{r_b, r_e\}$, where r_b and r_e are two different distinguished elements of R which take care of an appropriate beginning and ending in G, respectively. More specifically, the firing of a transition $t \in tr(\mathscr{P})$ uniquely corresponds to the use of a sequence of rewrites which generate $lab(\mathscr{P})(t)$ on the first coordinate, consume, for every place $p \in pl(\mathscr{P})$, $fl(\mathscr{P})(p, t)$ occurrences of the letter from $al(G_2) - \{S_2\}$ corresponding to p on the second coordinate, and generate, for every place $p \in pl(\mathscr{P})$, $fl(\mathscr{P})(t, p)$ occurrences of the letter from $al(G_2) - \{S_2\}$ corresponding to p on the second coordinate. This may be graphically represented as follows:



is a transition (with its inputs and outputs, where input places or output places appearing more than once mean multiple arcs) in $\mathscr{P}$, if and only if

$$((S_1', T_{i,0}), (S_2, S_2)), ((T_{i,0}, T_{i,1}), (p_{i_1}, \lambda)), \ldots,$$
$$((T_{i,m-1}, T_{i,m}), (p_{i_m}, \lambda)), ((T_{i,m}, \sigma S_1'), (S_2, S_2 p_{j_1} \ldots p_{j_n}))$$

is a sequence of elements of R for some pairwise distinct elements $S_1', T_{i,0}, \ldots, T_{i,m}$ in $nterm(G_1)$.

The easy but tedious formal details concerning the above construction and the proof of the lemma are left to the reader. □

Theorem 2.1 follows from Lemmas 2.2 and 2.3. □

Since it is well known that $\mathscr{L}$(lmPNfz) equals the class of languages generated by labeled marked Petri nets with an *arbitrary* final marking different from the initial marking, denoted by $\mathscr{L}_0^\lambda$, and since it is proved in [8] (see also [9]) that $\mathscr{L}_0^\lambda$ equals the class of languages generated by zero-testing-bounded multicounter machines, denoted $\mathscr{Z}^\lambda$, we get the following result.

Corollary 2.4. $\mathscr{L}(\text{RL; 0S}) = \mathscr{Z}^\lambda$.

3. Subclasses

In this section we demonstrate that the relationship between (RL; 0S)-systems and Petri nets is even deeper than indicated by Theorem 2.1. It turns out that a natural subclass of the class of (RL; 0S)-systems corresponds to a natural subclass of the class of lmPN's.

First we recall (see, e.g., [6]) the definition of an often considered subclass of lmPN's.

Definition 3.1. Let $\mathscr{P}$ be an lmPN. $\mathscr{P}$ is a *λ-free* lmPN if $lab(\mathscr{P})(t) \neq \lambda$ for every $t \in tr(\mathscr{P})$.

Next we define a natural subclass of the class of (RL; 0S)-systems.

Definition 3.2. (1) Let G be an RL-grammar. G is *real-time* if $(X, w) \in prod(G)$ implies $w \in term(G) \cdot nterm_\lambda(G)$.

(2) Let $G = (G_1, G_2, R)$ be an (RL; 0S)-system. G is *real-time* if G_1 is real-time.

Analyzing the proof of Lemma 2.2 one easily gets the following result.

Lemma 3.3. *Let K be a language. If K is generated by a real-time* (RL; 0S)*-system, then K is also generated by a λ-free* lmPN.

The proof of the 'converse' of the above lemma is somewhat more involved.

Lemma 3.4. *Let K be a language. If K is generated by a λ-free* lmPN, *then K is generated by a real-time* (RL; 0S)*-system.*

Proof. For an arbitrary λ-free lmPN $\mathscr{P}$ we will construct a real-time (RL; 0S)-system G such that $L(\mathscr{P}) = L(G)$.

The idea behind the construction is as follows. Let $\mathcal{P}$ be a λ-free lmPN, let $n = \#pl(\mathcal{P})$ and let $G = (G_1, G_2, R)$ be the constructed real-time (RL; 0S)-system. Every nonterminal of G_1 is an $(n+1)$-dimensional vector over $\mathbb{N}^{(n)} \times \{1, \ldots, n\}$—such a nonterminal represents a marking of $\mathcal{P}$ together with a distinguished place of $\mathcal{P}$ (by suitably alternating the distinguished last field one assures that each place becomes 'pointed out' once during n consecutive steps of a derivation process). However, if $\mathcal{P}$ has an infinite number of different reachable markings, then not every reachable marking can be represented by a nonterminal of G_1 since $nterm(G_1)$ has to be finite. This representation problem (as far as G_1 is concerned) is taken care of by G_2, which arranges the second coordinate to be an infinite store of ('packages' of) tokens (each 'package' consisting of tokens from the same place). At each step of a derivation process, only the place 'pointed out' by the (last field of the) nonterminal from the first coordinate is able to get from or to deposit on the second coordinate (a 'package' of) tokens. *Getting* tokens is only allowed if there is a possibility that these tokens are needed on the first coordinate during the next n steps of the derivation process; *depositing* tokens is only allowed if it is certain that these tokens will not be needed on the first coordinate during the next n steps of the derivation process. Consequently, *getting* tokens is only allowed if the current nonterminal on the first coordinate represents less than a certain fixed amount of tokens in the distinguished place and *depositing* tokens is only allowed if the current nonterminal on the first coordinate represents more than a (possibly different) fixed certain amount of tokens in the distinguished place.

Formally, the construction is as follows. Let $\mathcal{P} = (P, T, F, \Gamma, l, M_0)$ be a λ-free lmPN, where $P = \{p_1, \ldots, p_n\}$, $n \geq 1$ (the case $n = 0$ is impossible). Define for all $1 \leq i \leq n$:

- $\text{in}_i = n \max\{F(t, p_i) \mid t \in T\}$ (hence, in_i is greater or equal to the maximal increase of the number of tokens in place p_i resulting from a firing sequence of length n),
- $\text{out}_i = n \max\{F(p_i, t) \mid t \in T\}$ (hence, out_i is greater or equal to the maximal decrease of the number of tokens in place p_i resulting from a firing sequence of length n), and
- $\max_i = \max\{\text{in}_i, \text{out}_i\}$ (hence, $\max_i$ is greater or equal to the maximal change of the number of tokens in place p_i resulting from a firing sequence of length n).

Note that if M_1 and M_2 are markings of $\mathcal{P}$, s is a firing sequence of length n from M_1 to M_2, and, for some $1 \leq i \leq n$, $p_i \in P$ is such that $\text{out}_i \leq M_1(p_i) \leq \text{out}_i + \max_i$, then $0 \leq M_2(p_i) \leq \text{out}_i + \max_i + \text{in}_i$. Moreover, if $0 \leq M_2(p_i) < \text{out}_i$, then $\text{out}_i \leq M_2(p_i) + \max_i \leq \text{out}_i + \max_i$ and if $\text{out}_i + \max_i < M_2(p_i) \leq \text{out}_i + \max_i + \text{in}_i$, then $\text{out}_i \leq M_2(p_i) - \max_i \leq \text{out}_i + \max_i$. Hence, if we have to our disposal an infinite store of 'packages' (each one of size $\max_i$) of tokens, then the 'working region' for place p_i can stretch from 0 to $(\text{out}_i + \max_i + \text{in}_i)$, because getting from or depositing on the store at most one 'package' of tokens every n steps can give us a value between out_i and $(\text{out}_i + \max_i)$ for place p_i again.

Define for all $1 \leq i \leq n$:

- $k_i = \text{out}_i + \max_i + \text{in}_i + M_0(p_i)$ (the size of the 'working region' of p_i is enlarged with $M_0(p_i)$, because the initial marking of p_i can be arbitrary large and one wants the initial marking of p_i to be in the 'working region' of p_i),
- $W_i = \{0, 1, \ldots, k_i\}$ (hence, W_i is the set of all integers from the 'working region' of p_i), and, for all $t \in T$,
- $\Delta_i(t) = F(t, p_i) - F(p_i, t)$ (hence, $\Delta_i(t)$ denotes the change of the marking of p_i resulting from the firing of t).

Let $W = \{1, \ldots, n\}$.

In the following construction of the real-time (RL; 0S)-system $G = (G_1, G_2, R)$, G_1 takes (by its nonterminals) care of the 'working regions' of the places of $\mathcal{P}$ and G_2 takes care of the infinite store of ('packages' of) tokens.

Let $G = (G_1, G_2, R)$ be the (RL; 0S)-system such that:

- $al(G_1) = \Gamma \cup ((\bigtimes_{i=1}^{n} W_i) \times W)$, where $\Gamma \cap ((\bigtimes_{i=1}^{n} W_i) \times W) = \emptyset$,
- $al(G_2) = \{S_2\} \cup \{T_1, \ldots, T_n\}$, where $S_2, T_1, \ldots, T_n$ are all different elements,
- $term(G_1) = \Gamma$,
- $ax(G_1) = [M_0(p_1), \ldots, M_0(p_n), 1]$,
- $ax(G_2) = S_2$, and,

for all $t \in T$ and $w \in W$, R contains the following rewrites:

(i) $(([i_1, \ldots, i_n, w], l(t)[i_1 + \Delta_1(t), \ldots, i_n + \Delta_n(t), (w \bmod n) + 1]), (S_2, S_2))$

if, for all $j \in W$, $F(p_j, t) \leq i_j \leq k_j$ and $0 \leq i_j + \Delta_j(t) \leq k_j$ (rewrites of this group simulate the firing of a transition t of $\mathcal{P}$ without storing a 'package' of tokens on or getting a 'package' of tokens from the second coordinate),

(ii) $(([i_1, \ldots, i_n, w], l(t)[i_1 + \Delta_1(t), \ldots, i_{w-1} + \Delta_{w-1}(t), i_w + \Delta_w(t) - \max_w, i_{w+1} + \Delta_{w+1}(t), \ldots, i_n + \Delta_n(t), (w \bmod n) + 1]), (S_2, S_2 T_w))$

if, for all $j \in W$, $F(p_j, t) \leq i_j \leq k_j$ and $0 \leq i_j + \Delta_j(t) \leq k_j$, and $i_w + \Delta_w(t) > M_0(p_w) + \text{out}_w + \max_w$ (rewrites of this group simulate the firing of a transition t of $\mathcal{P}$ and store a 'package' of $\max_w$ tokens from place p_w on the second coordinate),

(iii) $(([i_1, \ldots, i_n, w], l(t)[i_1 + \Delta_1(t), \ldots, i_{w-1} + \Delta_{w-1}(t), i_w + \Delta_w(t) + \max_w, i_{w+1} + \Delta_{w+1}(t), \ldots, i_n + \Delta_n(t), (w \bmod n) + 1]), (T_w, \lambda))$

if, for all $j \in W$, $F(p_j, t) \leq i_j \leq k_j$ and $0 \leq i_j + \Delta_j(t) \leq k_j$, and $i_w + \Delta_w(t) < M_0(p_w) + \text{out}_w$ (rewrites of this group simulate the firing of a transition t of $\mathcal{P}$ and get a 'package' of $\max_w$ tokens for place p_w from the second coordinate), and

(iv) $(([-\Delta_1(t), \ldots, -\Delta_n(t), w], l(t)), (S_2, \lambda))$

if $F(p_j, t) = -\Delta_j(t)$ for all $j \in W$ (rewrites of this group simulate the firing of a transition t of $\mathcal{P}$ resulting in the final zero-marking; they end the simulation of the system).

R contains no rewrites other than those described under (i), (ii), (iii), and (iv).

It is obvious that G is real-time and it is not difficult to prove that $L(G) = L(\mathcal{P})$. Consequently, the lemma holds. □

Lemma 3.3 together with Lemma 3.4 yields the following characterization theorem.

Theorem 3.5. *Let K be a language. K is generated by a λ-free* lmPN *if and only if K is generated by a real-time* (RL; 0S)*-system.*

It turns out that the real-time restriction on (RL; 0S)-systems restricts the class of languages obtained. To prove this we make use of the following known result from the theory of Petri nets (see [7, 9]).

Proposition 3.6. *There exists a language $K \in \mathscr{L}$*(lmPNfz)*, such that $K - \{\lambda\}$ cannot be generated by a λ-free* lmPN.

Thus, from Theorems 2.1 and 3.5, and the above result, we immediately get the following result.

Theorem 3.7. *There exists a language $K \in \mathscr{L}$*(RL; 0S)*, such that $K - \{\lambda\}$ cannot be generated by a real-time* (RL; 0S)*-system.*

4. (RL; $0S^2$)-systems

As we have already indicated, one of the basic motivations to investigate (RL; 0S)-systems was to investigate the power of a well-established selector (0S) when it is used as a selector for memory access (that is, on the second coordinate with the first coordinate being a right-linear grammar).

Another natural selector is the selector of the form $\Sigma^*\bar{\Sigma}\bar{\Sigma}\Sigma^*$—to which we refer as a *0-bisequential* or *$0S^2$*-selector. Such a selector lies at the very basis of context-sensitive grammars.

Consequently (following the line of investigation that compares the power of various classes of selectors used in 'grammatical' and in 'storage' mood, see [15]), it is natural to investigate the power of the $0S^2$-selector used as a selector for memory access. Such an investigation sets the results we have obtained so far in a better perspective.

We start with formally defining $0S^2$-grammars and (RL; $0S^2$)-systems.

Definition 4.1. A *0-bisequential grammar*, abbreviated *$0S^2$-grammar*, is a 5-tuple $G = (\Sigma, h, S_1, S_2, K)$, where:

(a) (Σ, h, K) is a table, called the *table of G* and denoted by *tab*(G),

(b) $S_1 \in \Sigma$ ($S_2 \in \Sigma$ respectively) is the *left* (*right* respectively) *axiom of G*, denoted by $ax_\ell(G)$ ($ax_r(G)$ respectively), and

(c) $K = \Sigma^*\bar{\Sigma}\bar{\Sigma}\Sigma^*$.

All the terminology and notations concerning tables carry over to $0S^2$-grammars (through their tables) in the obvious way.

Definition 4.2. (1) A *right-linear 0-bisequential system*, abbreviated (*RL*; *0S*^{2})-*system*, is a triple $G=(G_1, G_2, R)$, where:

(a) G_1 is an RL-grammar,

(b) G_2 is a 0S^2-grammar, and

(c) R, called the set of *rewrites of* G and denoted by $rew(G)$, is a set of pairs of the form (r, U), where $r \in prod(G_1)$ and $U \subseteq prod(G_2)$ such that $1 \leq \# U \leq 2$.

(2) Let $G=(G_1, G_2, R)$ be an (RL; 0S^2)-system.

(2.1) Let $x=(x_1, x_2)$, $y=(y_1, y_2) \in (al(G_1))^* \times (al(G_2))^*$. We say that x *directly derives* y *in* G, denoted $x \Rightarrow_G y$, if there exists an $r=(r_1, U) \in R$ such that $x_1 \Rightarrow_{G_1}^{r_1} y_1$, and, for some $S \subseteq U$, $x_2 \Rightarrow_{G_2}^{S} y_2$. We then say that x *directly derives* y *in* G *using* r and write $x \Rightarrow_G^r y$. As usual, $\Rightarrow_G^*$ is the reflexive transitive closure of $\Rightarrow_G$; if $x \Rightarrow_G^* y$, then we say that x *derives* y *in* G.

(2.2) The *language generated by* G, denoted by $L(G)$, is defined by $L(G)=\{w \in (term(G_1))^* \mid (ax(G_1), ax_\ell(G_2)ax_r(G_2)) \Rightarrow_G^* (w, \lambda)\}$; $L(G)$ is referred to as an (RL; 0S^2)-*language*.

The class of all (RL; 0S^2)-languages is denoted by $\mathscr{L}$(RL; 0S^2).

For an (RL; 0S^2) system G we have required that $1 \leq \# U \leq 2$, whenever $(r, U) \in rew(G)$. The reason for this restriction is rather 'esthetical': in a single derivation step of a 0S^2-grammar, at most two different productions can be applied.

In the rest of this section we demonstrate that using the 0S^2-selector as a selector for memory access yields all (and nothing but) recursively enumerable languages. (The class of all recursively enumerable languages will be denoted by $\mathscr{L}_{RE}$.)

The following well-known result (see, e.g., [4]) will help us to establish the above-mentioned result.

Proposition 4.3. *Let* $\mathscr{L}$ *be a family of languages, such that*:

(i) $\{a^n b^n \mid n \geq 1\} \in \mathscr{L}$, *and*

(ii) $\mathscr{L}$ *is closed under union, concatenation,* $+$, *intersection with regular sets, arbitrary homomorphism, inverse homomorphism, and intersection.*

Then, $\mathscr{L}_{RE} \subseteq \mathscr{L}$.

Theorem 4.4. $\mathscr{L}_{RE} = \mathscr{L}$(RL; 0S^2).

Proof. Clearly, $\mathscr{L}$(RL; 0S^2) $\subseteq \mathscr{L}_{RE}$. Thus, it suffices to show the 'converse' inclusion. This will be done in two steps, each taken care of by a lemma.

Lemma 4.5. $\{a^n b^n \mid n \geq 1\} \in \mathscr{L}$(RL; 0S^2) *and* $\mathscr{L}$(RL; 0S^2) *is closed under union, concatenation, intersection with regular sets, arbitrary homomorphism, inverse homomorphism, and intersection.*

Proof. The straightforward constructions proving this lemma are left to the reader. □

Lemma 4.6. $\mathscr{L}(\mathrm{RL}; 0\mathrm{S}^2)$ *is closed under* $+$.

Proof. Let $G=(G_1, G_2, R)$ be an arbitrary $(\mathrm{RL}; 0\mathrm{S}^2)$-system. The constructed $(\mathrm{RL}; 0\mathrm{S}^2)$-system $G'=(G_1', G_2', R')$ which generates $(L(G))^+$ works as follows. In its first step it 'switches' from (axG_1'), $ax_\ell(G_2')ax_\mathrm{r}(G_2'))$ to $(ax(G_1), ax_\ell(G_2')ax_\ell(G_2)ax_\mathrm{r}(G_2)ax_\mathrm{r}(G_2'))$ and then it runs according to rewrites of R. Then at some point of its computation, G' introduces a special 'choice' symbol Z on the first coordinate. The choice symbol gives a possibility of:
either, to end the computation by using the rewrite

$$((Z, \lambda), \{(ax_\ell(G_2'), \lambda), (ax_\mathrm{r}(G_2'), \lambda)\}),$$

or, to 'start all over again' by using the rewrite

$$((Z, ax(G_1')), \{(ax_\ell(G_2'), ax_\ell(G_2')), (ax_\mathrm{r}(G_2'), ax_\mathrm{r}(G_2'))\}).$$

It is easily seen that a given derivation in G' produces λ on the second coordinate if and only if each occurrence of the choice symbol Z on the first coordinate corresponds to completing (the simulation of) a 'successful derivation' in G, where the nonterminal on the first coordinate disappears and the corresponding word on the second coordinate equals λ. □

From Lemmas 4.5 and 4.6, and Proposition 4.3 it follows that $\mathscr{L}_{\mathrm{RE}} \subseteq \mathscr{L}(\mathrm{RL}; 0\mathrm{S}^2)$. □

Directly from the above result, from Theorem 2.1, and from the well-known fact (see, e.g., [9]) that $\mathscr{L}(\mathrm{lmPNfz}) \subsetneq \mathscr{L}_{\mathrm{RE}}$ we get the following result.

Corollary 4.7. $\mathscr{L}(\mathrm{RL}; 0\mathrm{S}) \subsetneq \mathscr{L}(\mathrm{RL}; 0\mathrm{S}^2)$.

5. Discussion

The present paper directly continues the results in [15] in the sense that it elaborates in depth (in more detail) on the flexibility of cts systems to model various types of grammars and automata discussed in the literature.

The particular purpose of this paper has been to investigate a very specific instance of cts systems, namely (RL; 0S)-systems. It turns out that these systems are very closely related to Petri nets which form an established model of concurrent processes.

As indicated in (the remarks preceding) Corollary 2.4 it is well known that there is a close relationship between (specific kinds of) counter machines and Petri nets (see, e.g., [5, 8, 9]). And, as indicated in the remarks following Definition 1.3, it is also quite evident that (RL; 0S)-systems are well suitable for simulating (special kinds of) multicounter machines. So, in this way, there is quite a close relationship

between our Theorem 2.1 and the results in [5, 8, 9]. However, we have aimed at showing *direct* relationships between Petri nets and (RL; 0S)-systems (rather than to use multicounter machines as a 'bridge' for showing these relationships). Also, as opposed to [5], we have obtained an explicit characterization for all labeled Petri net languages (and not only for λ-free labeled variants) and, as opposed to [5, 8] as well as to [9], we have obtained a close relationship between subclasses. Although the basic idea (counting tokens) behind the main correspondence theorem is common to [5, 8, 9] and our paper, our proof seems different from those in the references just mentioned.

In Section 4 we have investigated another specific instance of cts systems, namely (RL; $0S^2$)-systems. It is interesting to notice that we have been able to establish that $0S^2$-selectors on the second coordinate (used for memory access) are more powerful that 0S-selectors on the second coordinate, because the question whether or not grammars using $0S^2$-selectors are more powerful than grammars using 0S-selectors is an intriguing open problem of grammatical formal language theory (see [12]).

Acknowledgment

The authors are indebted to J. Engelfriet, H.J. Hoogeboom, H.C.M. Kleijn, E. Welzl and an unknown referee for the comments on the previous versions of this paper.

References

[1] W. Brauer, ed., *Net Theory and Applications*, Lecture Notes in Computer Science **84** (Springer, Berlin, 1980).

[2] A. Ehrenfeucht, H.J. Hoogeboom and G. Rozenberg, Computations in coordinated pair systems, Tech. Rept. 84-25, Inst. of Appl. Math. and Comput. Sci., Univ. of Leiden, Leiden, 1984.

[3] A. Ehrenfeucht, H.J. Hoogeboom and G. Rozenberg, Coordinated pair systems; Part 1: Dyck words and classical pumping, Tech. Rept., Dept. of Comput. Sci., Univ. of Colorado, Boulder, 1985.

[4] S. Ginsburg, *Algebraic and Automata-Theoretic Properties of Formal Languages* (North-Holland, Amsterdam, 1975).

[5] S.A. Greibach, Remarks on blind and partially blind one-way multicounter machines, *Theoret. Comput. Sci.* **7** (1978) 311-324.

[6] M. Hack, Petri net languages, Tech. Rept. 159, Lab. for Comput. Sci., MIT, Cambridge, MA, 1976.

[7] M. Jantzen, On the hierarchy of Petri net languages, *RAIRO Inform. Théor.* **13** (1979) 19-30.

[8] M. Jantzen, On zerotesting-bounded multicounter machines, *Lecture Notes in Computer Science* **67** (Springer, Berlin, 1979) 158-169.

[9] M. Jantzen, Eigenschaften von Petrinetzsprachen, Bericht No. IFI-HH-B-64, Fachber. Informatik, Univ. Hamburg, Hamburg, 1979.

[10] H.C.M. Kleijn, Selective substitution grammars based on context-free productions, Ph.D. Thesis, Inst. of Appl. Math. and Comput. Sci., Univ. of Leiden, Leiden, 1983.

[11] H.C.M. Kleijn and G. Rozenberg, Context-free like restrictions on selective rewriting, *Theor. Comput. Sci.* **16** (1981) 237-269.

[12] H.C.M. Kleijn and G. Rozenberg, Problem section, *Bull. of EATCS* **26**, 1985.

[13] A. Pagnoni and G. Rozenberg, eds., *Applications and Theory of Petri Nets*, Informatik-Fachberichte 66 (Springer, Berlin, 1983).
[14] W. Reisig, *Petri Nets: An Introduction* (Springer, Berlin, 1985).
[15] G. Rozenberg, On coordinated selective substitutions: Towards a unified theory of grammars and machines, *Theoret. Comput. Sci.* 37 (1985) 31-50.
[16] A. Salomaa, *Formal Languages* (Academic Press, London/New York, 1973).