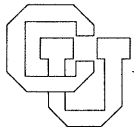
On Coordinated Selective Substitutions: Towards a Unified Theory of Grammars and Machines

G. Rozenberg*

CU-CS-256-83 September 1984



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

*This research was supported in part by NSF Grant number MCS 83-05245.

ABSTRACT

The notion of a *coordinated table selective substitution system (* a *cts system)* is introduced. It provides a unifying framework for both grammars *and* machines (automata) and hence a really broad framework for formal language theory. An extensive number of examples is given which illustrate how a quite considerable number of grammars and automata considered in the literature may be "naturally" interpreted as special instances (subclasses of the class) of cts systems.

INTRODUCTION

Two basic constructs used in formal language theory to define languages are grammars and automata (machines). The literature (see, e.g., [H] and [S]) is full of various instances of grammars and automata where each model is motivated by specific needs arising either from theoretical or practical considerations. For a number of obvious reasons it is very desirable to have a general model of a grammar or a general model of an automaton which is both, general enough to cover a considerable number of specific instances and "concrete" enough so that one can develop its theory (prove specific results).

In [R] the notion of a selective substitution grammar was introduced to provide a unifying framework for many of the rewriting systems met in the literature. Roughly speaking this model has formalized a number of essential features basic to many grammars (rewriting systems). The two most fundamental of these features are:

(1) the *rewriting of a single occurrence of a letter* which is mathematically formalized as a *substitution*, and

(2) the combination (spreading) of these elementary rewritings through a word so as to yield the *rewriting of a word* corresponding to a *direct derivation step;* this is formalized through the notion of *selector*.

The theory of selective substitution grammars developed since (see, e.g., [K] and [KR1]) has turned out to be successful in the sense that several central notions of "grammatically oriented" formal language theory got captured (and investigated) in a rather natural way within this theory and moreover a number of new (and clearly central) notions have emerged.

A natural next step in the build up of a "unified formal language theory" is to construct an analogous framework for the "automata oriented formal

language theory".

However rather than to do this we propose in this paper a common framework for grammars and automata. Surprisingly enough it turns out that very many instances of grammars and automata discussed in the literature are instances of one general model, which (using the "classic" intuition) is grammar oriented.

Roughly speaking the basic idea is as follows.

As discussed above, a substitution together with a selector form the most basic concept of rewriting - let's call it a *table*.

Now rather than rewrite a word let's rewrite n-tuples of words for some $n \geq 1$. The basic device directing such a rewriting is an n-tuple of tables called a *rewrite*. Thus given an n-tuple of words $(x_1, \ldots, x_n)$ a rewrite $(T_1, \ldots, T_n)$ rewrites it into an n-tuple $(y_1, \ldots, y_n)$ if, for each $1 \leq i \leq n$, T_i rewrites x_i into y_i. Thus we deal with a *coordinated rewriting* (coordinated use of component tables).

Finite sets of rewrites (together with the specification of alphabets of components and the sets of tables "available" at each component) form a *coordinated table selective substitution (cts) system* or *scheme* depending on whether or not one provides an *axiom* (equal to an n-tuple of *component axioms)* which gives a uniform starting point for all computations.

Then the most straightforward (perhaps somewhat rough) division within this model is into systems of dimension 1 (i.e, $n = 1$) and systems of dimension at least 2 (i.e., $n \geq 2$) - the first class corresponds naturally to *grammars* while the second class corresponds naturally to *automata* (machines).

In this paper the above intuition is formalized and the formal model of a *cts system* (and its extension: an *ects system)* is introduced and then quite extensively illustrated by examples. The role of the examples is to demonstrate how a

considerable number of grammars and machines may be quite easily interpreted as systems discussed in the paper. It is interesting to notice that indeed most types of automata are modeled using systems of dimension at least 2, while most types of grammars are modeled using systems of dimension 1.

We also demonstrate a rather intriguing result that the relative strength of (types of) selectors depends rather drastically on the fact whether they are used "directly" in the production (generation) of results (words) or whether they are used "indirectly" (i.e. through "auxiliary storage") for this purpose.

0. PRELIMINARIES

We assume the reader to be familiar with basic formal language theory, in particular we assume the reader to be familiar with basic grammar and machine (automata) models, e.g., as presented in [H] and [S].

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

For a set A, 2^A denotes the set of all its subsets and $\#A$ denotes its cardinality; $\emptyset$ denotes the empty set. For sets A, B, $A \backslash B$ 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.

We consider only finite and nonempty alphabets.

For a finite sequence ρ, $\rho(n)$ denotes the n'th element of ρ and $last(\rho)$ denotes the last element of ρ. In the considerations of this paper it is sometimes more convenient to index sequences starting with 0 and sometimes it is more convenient to index sequences starting with 1. In the former case we will write a sequence ρ in the form $\rho(0), \rho(1), \ldots$ and in the latter case we will write a

sequence ρ in the (tuple) form $\rho = (\rho(1), \rho(2), \ldots)$.

For a word x, $|x|$ denotes its length and $pref(x)$ denotes the set of all prefixes of x; Λ denotes the empty word.

For a language K, $pref(K) = \bigcup_{x \in K} pref(x)$.

If Σ, Δ are alphabets such that $\Delta \subseteq \Sigma$, then $pres_{\Sigma,\Delta}$ is a homomorphism of Σ^* defined by: $pres_{\Sigma,\Delta}(a) = \Lambda$ if $a \in \Sigma \backslash \Delta$ and $pres_{\Sigma,\Delta}(a) = a$ if $a \in \Delta$. Whenever Σ is clear from the context we write $pres_\Delta$ rather than $pres_{\Sigma,\Delta}$.

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

We proceed now to define a number of notions very basic for this paper.

Let Σ be an alphabet.

A *production (over* Σ*)* is a pair (X, x), where $X \in \Sigma$ and $x \in \Sigma^*$; it is also written in the form $X \to x$. Since each production is an element of $\Sigma \times \Sigma^*$, a set of productions h is a subset of $\Sigma \times \Sigma^*$; hence we use also the "functional notation" - for $X \in \Sigma$, $h(X) = \{x \in \Sigma^* : (X, x) \in h\}$.

A *selector (over* Σ*)* is a subset of $(\Sigma \cup \overline{\Sigma})^* \, \overline{\Sigma} (\Sigma \cup \overline{\Sigma})^*$; elements of a selector are referred to as *selector words*.

Definition 0.1. A *table* is a triple $T = (\Sigma, h, K)$, where Σ is an alphabet, h is a finite set of productions over Σ and K is a selector over Σ. The alphabet Σ is referred to as the *alphabet of* T and denoted by $al(T)$, h is called the set of productions of T and denoted by $prod(T)$ and K is called the *selector of* T and denoted by $sel(T)$. The pair (h, K) is called the *core of* T and denoted by

$core(T)$. ■

For a set of tables $\mathbf{T}$, $al(\mathbf{T}) = \{al(T) : T \in \mathbf{T}\}$, $prod(\mathbf{T}) = \{prod(T) : T \in \mathbf{T}\}$, $sel(\mathbf{T}) = \{sel(T) : T \in \mathbf{T}\}$ and $core(\mathbf{T}) = \{core(T) : T \in \mathbf{T}\}$. We say that $\mathbf{T}$ is *alphabet* (respectively *selector*) *uniform* if $\#al(\mathbf{T}) = 1$ (respectively $\#sel(\mathbf{T}) = 1$).

Definition 0.2. Let $T = (\Sigma, h, K)$ be a table, $x, y \in \Sigma^*$, where $x = b_1 \cdots b_n$, $n \geq 1$ and $b_i \in \Sigma$ for $1 \leq i \leq n$. We say that x *directly derives* y *in* T, denoted $x \underset{T}{\Longrightarrow} y$, if $y = \beta_1 \cdots \beta_n$, with $\beta_i \in \Sigma^*$ for $1 \leq i \leq n$, and there exists a $z \in K$, $z = a_1 \cdots a_n$ with $a_i \in \Sigma \bigcup \overline{\Sigma}$ for $1 \leq i \leq n$, such that

$iden_{\Sigma}(z) = x$ and, for $1 \leq i \leq n$,

if $a_i \in \Sigma$, then $\beta_i = b_i$ and if $a_i \in \overline{\Sigma}$, then $\beta_i \in h(b_i)$. ■

Whenever $x \underset{T}{\Longrightarrow} y$ and $U = core(T)$ we also write $x \underset{U}{\Longrightarrow} y$.

1. ON cts SCHEMES

In this section the notion of a cts scheme, which is central to this paper, is introduced and various subclasses of the class of cts schemes are discussed.

Definition 1.1. A *coordinated table selective substitution scheme*, abbreviated *cts scheme*, is a construct $H = (\mathbf{T}_1, \ldots, \mathbf{T}_n, R)$, where $n \geq 1$, for each $1 \leq i \leq n$, $\mathbf{T}_i$ is an alphabet uniform finite nonempty set of tables and $R \subseteq \bigtimes_{i=1}^{n} core(\mathbf{T}_i)$, $R \neq \emptyset$. For $1 \leq i \leq n$, $\mathbf{T}_i$ is referred to as the *i-th component of* H and elements of R are called *rewrites* of H; R is denoted by $rew(H)$. ■

The following terminology and notation will also be used (let H be as above).
n is called the *dimension* of H and denoted by $dim(H)$.
For $1 \leq i \leq n$, the alphabet common to all tables of the i-th component is called the *i-th alphabet of* H and denoted by $al_i(H)$.

Definition 1.2. Let $H = (\mathbf{T}_1, \ldots, \mathbf{T}_n, R)$ be a cts scheme.

(1) Let $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in \bigtimes_{i=1}^{n} (al_i(H))^*$. We say that x *directly computes* y *(in* H*)*, denoted $x \underset{H}{\Longrightarrow} y$, if there exists a $(U_1, \ldots, U_n) \in R$ such that $x_i \underset{U_i}{\Longrightarrow} y_i$, for $1 \leq i \leq n$.

(2) Let $x = (x_1, \ldots, x_n) \in \bigtimes_{i=1}^{n} (al_i(H))^*$. An *x-computation (in* H*)* is a sequence $\rho = \rho(0), \rho(1), \ldots, \rho(m), m \geq 1$, of elements of $\bigtimes_{i=1}^{n} (al_i(H))^*$ such that $\rho(0) = x$ and, for all $0 \leq i \leq m-1$, $\rho(i) \underset{H}{\Longrightarrow} \rho(i+1)$.

Each $\rho(i)$, $0 \leq i \leq m$, is called a *snapshot* of ρ.

(3) Let $\overset{*}{\underset{H}{\Longrightarrow}}$ denote the reflexive and the transitive closure of $\underset{H}{\Longrightarrow}$. If $x \overset{*}{\underset{H}{\Longrightarrow}} y$,

then we say that x *computes* y *(in* H*).* ■

We conclude this section by defining a number of subclasses of the class of cts schemes. These subclasses will turn out to be useful when we will demonstrate (in Section 3) how various examples of grammars and machines encountered in the literature fit into the framework of our theory. Also, the notation for (the specification of) schemes from these subclasses will be considerably simpler.

Definition 1.3. A cts scheme is *(selector) uniform*, abbreviated *ucts scheme*, if all of its components are selector uniform. ■

To simplify the notation, given a ucts scheme $H = (\mathbf{T}_1, \ldots, \mathbf{T}_n, R)$, R will be specified as a subset of $\bigtimes_{i=1}^{n} prod(\mathbf{T}_i)$; clearly such specifications are "correct" - the information about "missing" selectors is contained in the specifications of $\mathbf{T}_1, \ldots, \mathbf{T}_n$.

Definition 1.4. Let T be a table. T is *sequential* if $sel(T) = \Sigma_1^* \overline{\Sigma}_2 \Sigma_3^*$ for some $\Sigma_1, \Sigma_2, \Sigma_3 \subseteq al(T)$. A cts scheme H is *sequential*, abbreviated *scts scheme*, if every table appearing in any component of H is sequential. ■

Remark. In [KR2] selectors of the form $\Sigma_1^* \overline{\Sigma}_2 \Sigma_3^*$ are called *1-sequential* and their finite unions are called *sequential*. We have changed the terminology here because in cts systems one component consists of a number of tables so that various components of a sequential selector can be "spread among" tables of a given component and one can deal with 1-sequential components only.

To simplify the notation, given a *uscts* scheme $H = (\mathbf{T}_1, \ldots, \mathbf{T}_n, R)$ (i.e., a cts scheme that is both uniform and sequential) R may be specified as a subset of $\bigtimes_{i=1}^{n} (\bigcup_{T \in \mathbf{T}_i} prod(T))$ and, for each $1 \leq i \leq n$, $\mathbf{T}_i$ will be specified as the table $(al_i(H), \bigcup_{T \in \mathbf{T}_i} prod(T), K_i)$, where K_i is the selector common to all tables of the $i-th$ component of G.

Although, in general, from such a specification the original components of G can't be recovered any more, the "work of G" (that is the $\underset{G}{\Longrightarrow}$ relation) is "correctly" specified in this way.

2. ON cts SYSTEMS

cts systems result from cts schemes by adding a starting point (the axiom) for computations in G. These systems are formally defined in this section and then their extensions *(ects systems)*, very convenient for defining languages, are considered.

Definition 2.1. A *coordinated table selective substitution system*, abbreviated *cts system*, is a construct $G = (C_1, \ldots, C_n, R)$, $n \geq 1$, where, for each $1 \leq i \leq n$, $C_i = (\mathbf{T}_i, S_i)$, where S_i is a symbol from the alphabet of tables $\mathbf{T}_i$ and $(\mathbf{T}_1, \ldots, \mathbf{T}_n, R)$ is a cts scheme, called the *underlying scheme of* G. ▪

All the terminology and notation concerning cts schemes carry over to cts systems in the obvious way (through the underlying cts schemes). Additionally we will use the following terminology and notation (let G be as above).
For each $1 \leq i \leq n$, S_i is called the *i'th axiom of* G, denoted $ax_i(G)$, and $\mathbf{T}_i$ is called the i'th *table set of* G, denoted $tab_i(G)$. The n-tuple $(ax_1(G), ax_2(G), \ldots, ax_n(G))$ is called the *axiom of* G and denoted by $ax(G)$.
An *ax(G)-computation in* G is referred to simply as a *computation in* G; $COM(G)$ denotes the set of all computations in G.

cts systems are (may be) used to define languages. The notion of a snapshot plays the most crucial role here - it plays a role comparable to that of a sentential form in classical grammars.

Definition 2.2. Let G be a cts system.

(1) The *exhaustive set of* G, denoted $E(G)$, is defined by

$$E(G) = \{x \in \bigtimes_{i=1}^{n} (al_i(G))^* : \ x \text{ is a snapshot of a } \rho \in COM(G)\}$$

(2) The *i-th exhaustive language of* G, denoted $E_i(G)$, is defined by

$E_i(G) = \{x(i) : x \in E(G)\}$.

(3) $E_1(G)$ is called the *exhaustive language of G.* ■

In order to define different kinds of languages of *cts* systems (or in other words: "to squeeze" various kinds of languages out of exhaustive languages) one has to specify *success conditions* which allow one to classify a computation in a *cts* system as *successful* or *unsuccessful.*

In general a *success condition* for an n dimensional *cts* system G is an n-ary (success) predicate $\mathbf{S} \subseteq \bigtimes_{i=1}^{n} (al_i(G))^*$. Then a computation ρ in G (with a given $\mathbf{S}$) is called *successful* if $\mathbf{S}(last(\rho))$ is true, otherwise ρ is *unsuccessful.* A snapshot of a successful computation is called a *successful snapshot.* A success predicate $\mathbf{S}$ is called *(coordinate) independent* if there exist predicates $\mathbf{S}_1, \ldots, \mathbf{S}_n$, $\mathbf{S}_i \subseteq (al_i(G))^*$ for all $1 \leq i \leq n$, such that $\mathbf{S} = \bigtimes_{i=1}^{n} \mathbf{S}_i$; note that then for all $x = (x_1, \ldots, x_n) \in \bigtimes_{i=1}^{n} (al_i(G))^*$, $\mathbf{S}(x)$ is true if and only if $\mathbf{S}_i(x_i)$ is true for all $1 \leq i \leq n$.

We will deal with independent predicates only and moreover we will assume that if $\mathbf{S} = (\mathbf{S}_1 \times \mathbf{S}_2 \times \cdots \times \mathbf{S}_n)$ is a success predicate, then, for all $1 \leq i \leq n$, $\mathbf{S}_i$ is a regular language. We say then that we deal with *regular success conditions.*

Moreover (in the spirit of a very common language definition mechanism used in formal language theory) for each component of G we will specify its terminal alphabet (Δ) and often use these alphabets in the specifications of success conditions ($\mathbf{S}_i = \Delta^*$). When we integrate the specification of a *cts* system with the specification of a success condition given as above we arrive at the following definition.

Definition 2.3. An *extended coordinated table selective substitution system*, abbreviated *ects system*, is a construct $G = (D_1, \ldots, D_n, R)$, $n \geq 1$ where, for each $1 \leq i \leq n$, $D_i = (\mathbf{T}_i, S_i, \Delta_i)$, $((\mathbf{T}_1, S_1), \ldots, (\mathbf{T}_n, S_n), R)$ is a cts system, called the *underlying system of* G and denoted by $und(G)$, and Δ_i is a subset of $al_i(und(G))$. ■

All the terminology and notation concerning cts systems carry over to ects systems in the obvious way (through the underlying cts systems). In addition (for G as above):

Δ_i is the *i-th terminal alphabet of* G, denoted by $ter_i(G)$.

Note that if, for all $1 \leq i \leq n$, $ter_i(G) = al_i(G)$ then we really deal with cts systems.

In this paper we do not fix any specific "main" way of defining languages of ects systems. Rather, as an example we give now a definition that settles a way of defining languages that stems from a method very popular in the theory of automata. Note that the first component of an ects systems plays a special role here - it is interpreted as the "input" of the system.

Definition 2.4. Let G be an ects system, where $n = dim(G)$. The *empty store language of* G, denoted $L_e(G)$, is defined by

$L_e(G) = \{x \in (ter_1(G))^* :$ there exists a $y = (y_1, \ldots, y_n) \in E(G)$,

such that $y_1 = x$ and $y_i = \Lambda$ for all $2 \leq i \leq n\}$. ■

Note that the above definition implies that the regular success condition $\mathbf{S} = (\mathbf{S}_1 \times \cdots \times \mathbf{S}_n)$ we use is such that $\mathbf{S}_1 = (ter_1(G))^*$ and $\mathbf{S}_i = \{\Lambda\}$ for all $2 \leq i \leq n$.

If $G = ((\mathbf{T}_1, S_1, \Delta_1), R)$ is an ects system of dimension 1, such that $\#\mathbf{T}_1 = 1$, then G may be specified in the form $G = (\Sigma, h, S, \Delta, K)$, where $S = S_1$, $\Delta = \Delta_1$

and $\mathbf{T}_1 = \{(\Sigma, \hat{h}, K)\}$ with $\hat{h} = prod(T) \cap R$. Then G is really an *s-grammar* - the very basic construct of the theory of selective substitution grammars (see, e.g., [KR1]).

If, additionally, G is sequential then, in the terminology of [KR2], G becomes a $1S$ *grammar*.

If $G = ((\mathbf{T}_1, S_1, \Delta_1), R)$ is an ects system of dimension 1, then G may be specified in the form

$G = \{G_1, \ldots, G_m\}$, where, for each $1 \leq i \leq m$,

$G_i = (\Sigma, h_i, S_1, \Delta_1, K_i)$, with $\Sigma = al_1(G)$ and $R = \{(h_1, K_1), \ldots, (h_m, K_m)\}$.

Hence, in this case, G is really a finite set of s-grammars all of which have a common alphabet, terminal alphabet and an axiom. G is called a *table s−grammar*.

Moreover, if each of the G_i, $1 \leq i \leq m$, is sequential, then G is a $1S$ *table grammar*.

Even if not all but only some components of an ects system are sequential and uniform, we will specify these components as 1S grammars. Also if for an i−th component of an ects system G we have $al_i(G) = ter_i(G)$, then we may omit $ter_i(G)$ from the specification of G.

If G is a sequential and uniform ects system, hence a *euscts* system, $G = ((\mathbf{T}_1, S_1, \Delta_1), \ldots, (\mathbf{T}_n, S_n, \Delta_n), R)$, then G may be specified in the form $G = (G_1, \ldots, G_n, R')$, where

(1) for all $1 \leq i \leq n$,

$G_i = (\Sigma_i, h_i, S_i, \Delta_i, K_i)$, where $\Sigma_i = al_i(G)$, K_i is the selector common to all tables in $\mathbf{T}_i$ and $h_i = \bigcup_{T \in \mathbf{T}_i} prod(T)$, and

(2) $R' \subseteq \bigtimes_{i=1}^{n} h_i$.

Hence in this case each G_i is a $1S$ grammar and R' is a finite set of n-tuples $(\pi_1, \ldots, \pi_n)$, where, for each $1 \leq i \leq n$, π_i is a production from G_i.

3. EXAMPLES

In this section we will give examples of various types of ects systems. The main aim of this section is to demonstrate how various types of grammars and machines considered in the literature can be interpreted as (are instances of) ects systems. When demonstrating such an interpretation we will only occasionally talk about the interpretation of the languages of the grammars and the systems considered - the reason is that these interpretations are mostly obvious (by adjusting the type of success conditions).

Example 3.1.

Let G be an s-grammar such that $G = (\Sigma, h, S, \Delta, K)$, where $K = \Sigma^*(\bar{\Sigma}\backslash\bar{\Delta})\Sigma^*$, $S \in \Sigma\backslash\Delta$, and, for each $X \in \Delta$, $h(X) = \emptyset$.

Then G is interpreted as a *context-free grammar*. The standard success condition for specifying the language of G is Δ^*. ▪

Example 3.2.

Let $G = \{G_1, \ldots, G_m\}$ be a table s-grammar such that for each $1 \leq i \leq m$ the selector of G_i equals $\bar{\Sigma}^+$, where Σ is the alphabet of G.

Then G is interpreted as an ETOL *system* (with "partial" tables allowed) (see e.g., [RS]). ▪

Example 3.3.

Let G be an s-grammar such that $G = (\Sigma, h, S, \Delta, K)$, where $K = \Sigma^* \bar{\Sigma} \Sigma^*$.

Then G is interpreted as an EOS *grammar* (see, e.g., [KR1]) and so we refer to G as an EOS *grammar*. If $\Sigma = \Delta$, then G is an OS *grammar*. ▪

Example 3.4.

Let G be an s-grammar such that $G = (\Sigma, h, S, \Delta, K)$, where $S \in \Sigma\backslash\Delta$, $K = \Delta^*(\overline{\Sigma\backslash\Delta})$, h is such that, for each $X \in \Delta$, $h(X) = \emptyset$ and moreover: if $x \in h(X)$ for $X \in \Sigma\backslash\Delta$ and $x \in \Sigma^*$, then $x \in (\Delta \cup \{\Lambda\})\,((\Sigma\backslash\Delta) \cup \{\Lambda\})$.

Because of the obvious interpretation we refer to G as a *right-linear grammar*. The "usual" success condition for specifying the (so called *terminal*) language of G is Δ^*. ■

Example 3.5.

Let G be as in the example above, except that the additional restriction is posed: if $x \in h(X)$ for $X \in \Sigma\backslash\Delta$ and $x \in \Sigma^*$, then $x \in (\Delta \cup \{\Lambda\})\,(\Sigma\backslash\Delta)$.

Then G is referred to as a *strict right-linear grammar*. G may be interpreted, in the obvious way, as a *one-way finite automaton*.

The "usual" success condition is of the form $\Delta^*\Theta$ where Θ is a distinguished subset of $\Sigma\backslash\Delta$, *however* the language of G results from "successful" words in G by removing from them the last letter (the element of Θ). One could also allow erasing productions $X \to \Lambda$, $X \in \Sigma\backslash\Delta$ in G and then to have the success condition of the form Δ^*; clearly both models are "equivalent" (as far as languages - as defined above - are concerned).

Note that we allow chain productions $X \to Y$, where $X, Y \in \Sigma\backslash\Delta$, and so we allow the finite automaton to make "Λ-moves", i.e., to read an input symbol, change its state and do *not* advance to the right. If we forbid chain productions, then we get a finite automaton in the classical sense.

If additionally we require that "whenever $X \to aY$ and $X \to aZ$ are productions in h, then $Y = Z$", then we deal with a *deterministic one-way finite automaton*. ■

Example 3.6.

Let $G = (G_1, G_2, R)$ be a *usects* system such that G_1 is a strict right-linear grammar and G_2 is a right linear grammar with one nonterminal only.

Then G may be interpreted as a *one-way finite automaton with output* or, in a different terminology, as a *generalized sequential machine*.

The obvious interpretation is that the first coordinate acts as an input tape (and keeps the information about the current state) while the second component acts as an *output tape*. ■

Example 3.7.

We will discuss now the two-way finite automaton as a two dimensional cts system.

Let Δ and Γ be disjoint alphabets and let $\Theta_1 = \{[\$,a] : a \in \Delta\}$, $\Theta_2 = \{[a] : a \in \Delta\}$ and $\Theta = \Theta_1 \cup \Theta_2$; we assume that $\Theta_1 \cap \Theta_2 = \emptyset$ and $\Theta \cap (\Delta \cup \Gamma) = \emptyset$. Let $\Sigma_1 = \Delta \cup \Gamma$ and $\Sigma_2 = \Delta \cup \Theta \cup \{S_2\}$, where $S_2 \notin \Theta \cup \Delta \cup \Gamma$; let q_{in} be a distinguished element of Γ.

Let $K_1 = \Delta^* \bar{\Gamma}$, $K_{2,0} = \{\bar{S}_2\}$, $K_{2,1} = \Theta^* \bar{\Theta}$, $K_{2,2} = \Theta^* \bar{\Theta} \Delta^*$ and $K_{2,3} = \Theta^* \bar{\Delta} \Delta^*$.

Let for each $a \in \Delta$, $(core(T_{1,a,0}), core(T_{2,a,0}))$ be a rewrite such that

$T_{1,a,0} = (\Sigma_1, h_{1,a,0}, K_1)$ and $T_{2,a,0} = (\Sigma_2, h_{2,a,0}, K_{2,0})$,

where $h_{1,a,0}$ is a set of productions all of which are of the form $q_{in} \to aq'$ for $q' \in \Gamma$ and $h_{2,a,0}$ consists of the production $S_2 \to [\$,a]$.

Let $R_0 = \{(core(T_{1,a,0}), core(T_{2,a,0})) : a \in \Delta\}$.

Let for each $a \in \Delta$, $R_{1,a}$ be a finite set of rewrites $(core(T_{1,a}), core(T_{2,a}))$ such that

$T_{1,a} = (\Sigma_1, h_{1,a}, K_1)$ and $T_{2,a} = (\Sigma_2, h_{2,a}, K_{2,1})$,

where

$h_{1,a}$ is a set of productions of the form $q \to aq'$ for some $q, q' \in \Gamma$ and

$h_{2,a}$ is a set of productions either of the form $[b] \to [b][a]$ or of the form $[\$, b] \to [\$, b][a]$ for some $b \in \Delta$.

Let $R_1 = \bigcup_{a \in \Delta} R_{1,a}$.

Let R_2 be a finite set of rewrites $(core(T_1), core(T_2))$ such that $T_1 = (\Sigma_1, h_1, K_1)$ and $T_2 = (\Sigma_2, h_2, K_{2,2})$, where

h_1 is a set of productions of the form $q \to q', q$, with $q' \in \Gamma$ and

h_2 is a set of productions of the form $[b] \to b$ with $b \in \Delta$.

Let R_3 be a finite set of rewrites $(core(T_1), core(T_2))$ such that $T_1 = (\Sigma_1, h_1, K_1)$ and $T_2 = (\Sigma_2, h_2, K_{2,3})$, where

h_1 is a finite set of productions of the form $q \to q'$ with $q, q' \in \Gamma$ and

h_2 is a finite set of productions of the form $b \to [b], b \in \Delta$.

Let $R = R_0 \cup R_1 \cup R_2 \cup R_3$.

Let $\mathbf{T}_1$ be the set of all tables such that their cores occur as first components of rewrites in R and let $\mathbf{T}_2$ be the set of all tables such that their cores occur as second components of rewrites in R.

Finally let $G = ((\mathbf{T}_1, q_{in}), (\mathbf{T}_2, S_2), R)$.

Then G is easily interpreted as an (off line) *two-way finite automaton*. Various elements of the definition of G are then interpreted as follows.

Γ is the set of states, Δ is the input alphabet and q_{in} is the initial state of the automaton considered.

On the first component we store the portion of the input already read and the current state.

The work of the automaton is done (simulated) on the second coordinate.

Rewrites from R_0 correspond to the "starting condition": reading the first symbol in the initial state q_{in}.

A rewrite from $R_{1,a}$, $a \in \Delta$, corresponds to the reading of the next (new) input

symbol (which equals a) and moving to the right.

Rewrites from R_2 correspond to going to the left on the input (provided that at "this moment" one does not read the leftmost symbol of the tape) and rewrites from R_3 correspond to going to the right on the input (providing one is "inside" the already read portion of the input tape). ■

Example 3.8.

Let G be an s-grammar such that $G = (\Sigma, h, S, \Sigma, K)$, where $K = \Sigma^* \bar{\Sigma}$ (or $K = \bar{\Sigma}\Sigma^*$).

We will refer to G as a *right-boundary grammar* (or *left-boundary grammar*, respectively); note that it differs from the right-linear grammar essentially by the fact that it does not distinguish between terminal and nonterminal symbols. ■

Left-boundary (and by analogy right-boundary) grammars can be considered as a special case of Büchi regular canonical systems (see [Bü]).

Here is another way of interpreting two-way automata as cts systems equipped with a very natural *"global control"* condition (restriction) on their work.

Example 3.9. Let G be a right-boundary grammar in which one restricts the set of all computations as follows. A computation in G is *allowed* (is *legal*) only if any two snapshots of it are in a *prefix relation* (i.e., one of them is a prefix of the other one).

Then it is not difficult to see that G may be interpreted as a *two-way finite automaton.* ■

Example 3.10.

Let $G = (G_1, G_2, R)$ be a uniform and sequential ects system such that

G_1 is a right-linear grammar and

G_2 is a right-boundary grammar.

Then G is easily interpreted as a *push-down automaton*. If (w_1q, w_2) is a snapshot of G, then w_1 is the portion of the input already read, q is the current state of the automaton and w_2 is its stack (with the rightmost letter of w_2 representing the topmost element of the stack).

A number of direct analogies are immediately observable.

A *chain production* in G_1 corresponds to a Λ-move in the *pda* **A** (being "simulated"): if G_1 is *chain-free*, then **A** is Λ-free. If there is a *common bound k on* the number of consecutive applications of *chain-rules* in G_1, then **A** *operates with delay k*.

A production in G_2 of the form $X \to \Lambda$ corresponds to a *pop* and a production in G_2 of the form $X \to w$ with $|w| \geq 2$ corresponds to a *push*.

Acceptance by final state in **A** corresponds to the regular success condition $(\Delta_1^* \Theta_1, \Sigma_2^*)$, where $\Delta_1 = ter_1(G)$, Θ_1 is a distinguished subset of $al_1(G) \backslash ter_1(G)$ and $\Sigma_2 = al_2(G)$.

Acceptance by empty store in **A** corresponds to the regular success condition $(\Delta_1^* (\Sigma_1 \backslash \Delta_1), \{\Lambda\})$, where $\Sigma_1 = al_1(G)$.

Acceptance by both final state and empty store in **A** corresponds to the regular success condition $(\Delta_1^* \Theta_1 , \{\Lambda\})$. ∎

Note that if we require in the above example that $al_2(G) = \{S_2 , Z_2\}$, where $S_2 = ax_2(G)$ and $Z_2 \neq S_2$, and moreover we require that the only instructions available for S_2 in G_2 are of the form $S_2 \to S_2$, $S_2 \to S_2Z_2$ and $S_2 \to \Lambda$ and the only

instructions available for Z_2 in G_2 are of the form $Z_2 \to Z_2$, $Z_2 \to Z_2Z_2$ and $Z_2 \to \Lambda$ then we deal with *one counter automaton.* (This is the usual way of getting a one counter automaton from a push down automaton: require the stack alphabet to consist of one letter and a "bottom" symbol.) The extension of the above model to cover *n-counter automata* is obvious.

Example 3.11.

Here is another look at one counter automata.

Let $G = (G_1, (\mathbf{T}_2, S_2, \Delta_2), R)$ be an ects system such that

(1) G_1 is a right linear grammar,

(2) $al_2(G) = \{S_2, Z_2\}$, where $Z_2 \neq S_2$,

(3) if $K \in sel(\mathbf{T}_2)$, then $K = (al_2(G))^* \{\overline{S}_2 , \overline{Z}_2\}(al_2(G))^*$ and

(4) if $\pi \in prod(h)$, then either π is of the form $Z_2 \to \Lambda$ or π is of the form $S_2 \to \Lambda$ or π is of the form $S_2 \to x$ where x contains exactly one occurrence of S_2.

Then G is easily interpreted as a *one counter automaton.* Clearly if $dim(G) = n > 1$ and one sets restrictions on $com_i(G)$, for all $2 \leq i \leq n$, analogous to the restrictions on $com_2(G)$ above, then one gets a $(n-1)$ *counter automaton.* ■

Example 3.12.

Let $G = (G_1, G_2, G_3, R)$ be a uniform and sequential ects system such that G_1 is a right linear grammar, G_2 is a right-boundary grammar and G_3 is a left-boundary grammar.

Then G is easily interpreted as a *two-way push down automaton.* ■

Example 3.13.

We will discuss now Petri nets as cts systems.

Let Δ be a finite alphabet, $S \notin \Delta$ and let $\Omega = \{S_t : t \in \Delta\}$; we assume that

$\Omega \cap (\Delta \cup \{S\}) = \emptyset$.

Let $n > 1$ and let, for each $2 \leq i \leq n$, S_i, p_i be two different symbols; let $\Sigma_i = \{S_i, p_i\}$ and $\Sigma_1 = \Delta \cup \Omega \cup \{S\}$.

Let $\Theta = \{2, \ldots, n\}$ and let *in* and *out* be functions from Δ into 2^{Θ}.

Let $K_1 = \Delta^*(\{\bar{S}\} \cup \bar{\Omega})$ and let, for $2 \leq i \leq n$,

$K_{i,1} = \Sigma_i^* \{\bar{p}_i\} \Sigma_i^*$ and $K_{i,2} = \{\bar{S}_i\}\{p_i\}^*$.

Let for each $t \in \Delta$, $(T_{t,1,1}, T_{t,2,1}, \ldots, T_{t,n,1})$ be the rewrite such that $T_{t,1,1} = (\Sigma_1, \{S \to tS_t\}, K_1)$ and, for $2 \leq i \leq n$,

$$T_{t,i,1} = \begin{cases} (\Sigma_i, \{p_i \to \Lambda\}, K_{i,1}) \text{ if } i \in in(t), \\ (\Sigma_i, \{S_i \to S_i\}, K_{i,2}) \text{ if } i \notin in(t). \end{cases}$$

Let R_1 be the collection of all rewrites $(T_{t,1,1}, \ldots, T_{t,n,1})$ for all $t \in \Delta$.

Let for each $t \in \Delta$, $(T_{t,1,2}, T_{t,2,2}, \ldots, T_{t,n,2})$ be the rewrite such that $T_{t,1,2} = (\Sigma_1, \{S_t \to S\}, K_1)$ and, for $2 \leq i \leq n$,

$$T_{t,i,2} = \begin{cases} (\Sigma_i, \{S_i \to S_i p_i\}, K_{i,2}) \text{ if } i \in out(t), \\ (\Sigma_i, \{S_i \to S_i\}, K_{i,2}) \text{ if } i \notin out(t). \end{cases}$$

Let R_2 be the collection of all rewrites $(T_{t,1,2}, \ldots, T_{t,n,2})$ for all $t \in \Delta$.

Let $R = R_1 \cup R_2$.

Let, for each $1 \leq i \leq n$, $\mathbf{T}_i$ be the set of all tables occurring as i-th components in all rewrites of R.

Then let G be a cts system defined by $G = ((\mathbf{T}_1, S), (\mathbf{T}_2, S_2), \ldots, (\mathbf{T}_n, S_n), R)$.

G is interpreted as a *Petri net* (with weights equal one on all arcs), see, e.g., [B].

The interpretation of various elements of G is as follows.

Δ is the set of *transitions* of the net and the components $com_i(G)$, $2 \leq i \leq n$, are *places* of the net. For a $t \in \Delta$, $in(t)$ are all input places for t and $out(t)$ are all output places for t.

Each firing of a transition t is done in G in two phases.

Phase 1. t consumes tokens from its input places: the rewrite $(T_{t,1,1}, \ldots, T_{t,n,1})$ is used.

Phase 2. t spreads tokens to its output places: the rewrite $(T_{t,1,2}, \ldots, T_{t,n,2})$ used. This phase follows always immediately after Phase 1.

Hence the firing sequences of the net are generated (to the left of S) on the first component. Note that various "standard" ways of defining the language of a Petri net can be easily accommodated by defining various success conditions. ▪

A very different way of interpreting Petri nets as cts systems is discussed in [AR].

4. ON THE RELATIVE POWER OF SELECTORS

In this section we will investigate and compare two different uses of (classes of) selectors:

(1) *"direct"* - i.e. on the first coordinate of an ects system (where the words of the language are computed) and

(2) *"indirect"* - i.e. on other than the first coordinate of an ects system (where only "auxiliary" computations take place).

Roughly speaking, the former mode corresponds to the use of selectors in grammars - where they *directly* influence the derivation process -, while the latter mode corresponds to the use of selectors in automata - where they determine the type of storage access and hence influence the computation of the language only *indirectly* (through the storage of an automaton).

Perhaps three most popular (from the grammatical point of view) classes of selectors are the following ones.

(1) Right-boundary (i.e. of the form $\Sigma^* \overline{\Sigma}$, where Σ is the alphabet involved): it underlies right-linear grammars. (Although in Example 3.4 the form of the selector used is $\Delta^*(\overline{\Sigma}\backslash\overline{\Delta})$, it is easily seen that, with the given form of productions, one can also use the selector $\Sigma^* \overline{\Sigma}$).

(2) OS (i.e. of the form $\Sigma^* \overline{\Sigma}\Sigma^*$, where Σ is the alphabet involved): it underlies context-free grammars. (Again, a remark analogous to the one made under (1) but concerning Example 3.1 can be made).

(3) OL (i.e. of the form $\overline{\Sigma}^+$, where Σ is the alphabet involved): it underlies ETOL systems.

In this section we will investigate and compare the use of the above three classes of selectors in (uniform) ects systems of dimension 1 (hence in direct mode) and in (uniform) ects systems of dimension 2 on the second component

(hence in indirect mode). In the latter case we will assume that the first component of an ects system is a right linear grammar - such an assumption seems to be reasonable, because right-linear grammars on the first component correspond to the standard input tape in most kinds of automata encountered in the literature.

Throughout this section we will consider the standard "empty store" way of defining languages of ects systems (see Definition 2.4).

The classes of languages obtained by using the classes of right-boundary, OS and OL selectors in uniform ects systems of dimension 1 will be denoted by $\mathbf{L}^{(1)}(RB)$, $\mathbf{L}^{(1)}(OS)$ and $\mathbf{L}^{(1)}(OL)$ respectively.

The classes of languages obtained by using the classes of right-boundary, OS and OL selectors in uniform ects systems of dimension 2 (where the first component is a right-linear grammar) will be denoted by $\mathbf{L}^{(2)}(RB)$, $\mathbf{L}^{(2)}(OS)$ and $\mathbf{L}^{(2)}(OL)$ respectively.

Moreover we will use $\mathbf{F}(REG)$, $\mathbf{F}(CF)$ and $\mathbf{F}(ETOL)$ to denote the classes of regular, context-free and ETOL languages. $\mathbf{F}(PN)$ denotes the class of languages that are accepted by Petri nets (that is labeled marked Petri nets with final zero marking) - see, e.g., [B].

We begin by comparing classes $\mathbf{L}^{(1)}(RB)$, $\mathbf{L}^{(1)}(OS)$ and $\mathbf{L}^{(1)}(OL)$.

Theorem 4.1. $\mathbf{L}^{(1)}(RB) \subsetneq \mathbf{L}^{(1)}(OS) \subsetneq \mathbf{L}^{(1)}(OL)$.

Proof.

As we have indicated already (see Example 3.8) a right-boundary grammar is a (special case of) Büchi's regular canonical system, see [Bü]. Consequently it follows from [Bü], see also [KR2], that $\mathbf{L}^{(1)}(RB) \subseteq \mathbf{F}(REG)$. On the other hand it is obvious that $\mathbf{F}(REG) \subseteq \mathbf{L}^{(1)}(RB)$ and consequently $\mathbf{L}^{(1)}(RB) = \mathbf{F}(REG)$.

It is easily seen that $\mathbf{L}^{(1)}(OS) = \mathbf{F}(CF)$ and $\mathbf{L}^{(1)}(OL) = \mathbf{F}(ETOL)$ (see Example 3.1 and Example 3.2 respectively).

Since it is well known that $\mathbf{F}(REG) \subsetneq \mathbf{F}(CF) \subsetneq \mathbf{F}(ETOL)$, the theorem follows. ∎

We move now to compare classes $\mathbf{L}^{(2)}(RB)$, $\mathbf{L}^{(2)}(OS)$ and $\mathbf{L}^{(2)}(OL)$. First we define formally the ects systems involved.

Definition 4.1. Let $G = (D_1, D_2, R)$ be a uniform and sequential ects system such that D_1 is a right-linear grammar and D_2 is a right-boundary grammar. Then G is a (RL, RB) *system*. ∎

$\mathbf{L}^{(2)}(RB)$ denotes the class of languages of the form $L_e(G)$, where G is a (RL, RB) system.

Definition 4.2. Let $G = (D_1, D_2, R)$ be a uniform and sequential ects system such that D_1 is a right-linear grammar and D_2 is a OS grammar. Then G is a (RL,OS) *system*. ∎

$\mathbf{L}^{(2)}(OS)$ denotes the class of languages of the form $L_e(G)$, where G is a (RL, OS) system.

Definition 4.3. Let $G = (D_1, D_2, R)$ be a uniform ects system such that D_1 is a right-linear grammar and $ter_2(G) = al_2(G)$. If the uniform selector of all tables from D_2 is of the form $(\overline{al_2(G)})^+$, then G is a (RL, TOL) *system*. ∎

$\mathbf{L}^{(2)}(OL)$ denotes the class of all languages of the form $L_e(G)$, where G is a (RL, TOL) system.

A (RL, TOL) system G will be specified in the form $G = (G_1, G_2, R)$, where

(i) G_1 is a right-linear grammar,

(ii) $G_2 = (\Sigma_2, H_2, S_2, K_2)$ is a TOL system with Σ_2 its alphabet, H_2 the set of its sets of productions, S_2 its axiom and K_2 its selector, and

(iii) R is a set of pairs of the form (π, h), where π is a production from G_1 and h is a set of productions from H_2.

We establish now the language computing power of (RL, TOL) systems.

Lemma 4.1. $\mathbf{L}^{(2)}(OL) \subseteq \mathbf{F}(REG)$.

Proof.

Let $G = (G_1, G_2, R)$ be a (RL; TOL) system, where $G_1 = (\Sigma_1, h_1, S_1, \Delta_1, K_1)$ and $G_2 = (\Sigma_2, H_2, S_2, K_2)$.

Clearly, without loss of generality we may assume that for each $h \in H_2$ there exists a $\pi \in h_1$ such that $(\pi,h) \in R$ and for each $\pi \in h_1$ there exists an $h \in H_2$ such that $(\pi, h) \in R$.

Let $\Theta_1 = \{ < \pi > : \pi \in h_1\}$ and $\Theta_2 = \{[h] : h \in H_2\}$.

Let α be the homomorphism of Θ_2^* into H_2^* defined by: $\alpha([h]) = h$ for each $h \in H_2$; as usual we assume that each word $g_1 \cdots g_n \in H_2^*$ with $g_1, \ldots, g_n \in H_2$ is (can be seen as) the composition of substitutions $g_1, \ldots, g_n$ from H_2 (each set of productions form H_2 can be considered as a finite substitution).

Let γ be the finite substitution of Θ_2^* into Θ_1^* defined by: $\gamma([h]) = \{ < \pi > : (\pi,h) \in R\}$.

Let $\hat{G}_1 = (\hat{\Sigma}_1, \hat{h}_1, \hat{S}_1, \hat{\Delta}_1, \hat{K}_1)$ be the right-linear grammar defined by:

$\hat{\Sigma}_1 = (\Sigma_1 \backslash \Delta_1) \cup \Theta_1$, $\hat{S}_1 = S_1$, $\hat{\Delta}_1 = \Theta_1$, $K_1 = \Theta_1^*(\overline{\Sigma}_1 \backslash \overline{\Delta}_1)$ and $\hat{h}_1$ is defined as follows:

(1) for all $X, Y \in \Sigma_1 \backslash \Delta_1$,

$X \rightarrow < \pi > Y \in \hat{h}_1$ if and only if $\pi = (X \rightarrow y\ Y) \in h_1$ for some $y \in \Delta_1 \cup \{\Lambda\}$,

(2) for all $X \in \Sigma_1 \backslash \Delta_1$,

$X \to < \pi > \in \hat{h}_1$ if and only if $\pi = (X \to y) \in h_1$ for some $y \in \Delta_1 \cup \{\Lambda\}$,

(3) $\hat{h}_1$ contains only productions as defined under (1) and (2) above.

Let β be the homomorphism of Θ_1^* into Δ_1^* defined by:

for each $< \pi > \in \Theta_1$, $\beta(< \pi >) = y$, where $\pi = (X \to yY)$ for some $X \in \Sigma_1 \backslash \Delta_1$, $Y \in (\Sigma_1 \backslash \Delta_1) \cup \{\Lambda\}$ and $y \in \Delta_1 \cup \{\Lambda\}$.

Now let

$M_2 = \{z \in \Theta_2^* : \Lambda \in (\alpha(z))(S_2)\}$,

$M_1 = \gamma(M_2)$ and

$M = M_1 \cap L(\hat{G}_1)$.

It is easily seen that $L_s(G) = \beta(M)$ ..... (*)

On the other hand it was proved in [GR] that given any TOL system H and any regular set L over the alphabet of H, the set of U of all sequences of sets of productions from H such that for each $u \in U$, $u(S) \cap L \neq \emptyset$, where S is the axiom of H , is regular. (The functional notation $u(S)$ is used to denote the set of all words that can be obtained by applying to S the sequence of finite substitutions u.)

This directly implies that M_2 is regular. Since $\mathbf{F}(REG)$ is closed under finite substitutions and intersections, $\beta(M)$ is regular. Consequently, the lemma follows now from (*). ■

Theorem 4.2. $\mathbf{L}^{(2)}(OL) = \mathbf{F}(REG)$.

Proof.

This follows immediately from Lemma 4.1 and from the obvious observation that $\mathbf{F}(REG) \subseteq \mathbf{L}^{(2)}(OL)$. ■

It is rather evident (see Example 3.10) that (RL, RB) systems correspond to push-down automata. Hence we have the following result.

Theorem 4.3 $\mathbf{L}^{(2)}(RB) = \mathbf{F}(CF)$. ▪

The following result is established in [AR].

Theorem 4.4. $\mathbf{L}^{(2)}(OS) = \mathbf{F}(PN)$. ▪

Theorem 4.5.

(1) $\mathbf{L}^{(2)}(OL) \subsetneq \mathbf{L}^{(2)}(RB)$.

(2) $\mathbf{L}^{(2)}(OL) \subsetneq \mathbf{L}^{(2)}(OS)$.

(3) $\mathbf{L}^{(2)}(OS)$ is incomparable with $\mathbf{L}^{(2)}(RB)$.

Proof.

ad.(1) This follows from Theorem 4.2, Theorem 4.3 and the well-known fact that $\mathbf{F}(REG) \subsetneq \mathbf{F}(CF)$.

ad.(2) This follows from Theorem 4.2, Theorem 4.4 and the well-known fact (see, e.g., [B]) that $\mathbf{F}(REG) \subsetneq \mathbf{F}(PN)$.

ad.(3) This follows from Theorem 4.3, Theorem 4.4 and the well-know fact (see, e.g., [B]) that $\mathbf{F}(CF)$ and $\mathbf{F}(PN)$ are not comparable. ▪

Comparing Theorem 4.1 with Theorem 4.5 leads one to rather intriguing observations. The relative strength of a particular type (class) of selectors depends rather *drastically* on the (direct or indirect) mode of use.

For example, while OL selectors are "stronger" than RB selectors when used in *direct* (i.e., "grammatical") mode, they are "weaker" than RB selectors when used in *indirect* (i.e. in "machine storage") mode !

But this "flip over" of computing strength when switching from direct to indirect mode *is not a rule*. For example, while OL selectors are stronger than OS selectors when used in direct mode, they are incomparable with OS selectors when used in indirect mode.

5. DISCUSSION

In this paper we have introduced ects systems and motivated them by demonstrating how various types of grammars and automata discussed in the literature can be seen as special instances of this model.

There exist in the literature several approaches (formalisms) to a general "abstract automata theory" (see, e.g., [E], [G], [Go] and [Sc]). Typically in these approaches an automaton is considered as a program operating on a data type. The automaton data type (or storage type) consists of a set **C** of configurations (or snapshots) together with a set of binary relations on **C** (the operations on the data type to be used by the program). Our formalism seems to be both more general and more restricted than this abstract automata type of formalism.

It is more general because it models both grammars and automata. As a matter of fact, our model consists only of a data type - the operations on the data type are (viewed as) transition (derivation) relations.

It is more restricted because a configuration of a data type is always a tuple of strings and an operation on the data type is always a tuple of selective substitutions (one for each element of a configuration). Hence considered data types are of the sort usually encountered in grammars. However, many types of automata can still be modeled in our framework mainly because:

(i) the finite state control and input (and/or output) of the automaton can be modeled by string(s) with selective substitution to model the state transitions and reading of the input (writing of the output),

(ii) storage configurations of an automaton (such as, e.g., pushdown or multicounter) can often be modeled by strings with selective substitutions modeling operations on these configurations.

Grammars rewriting synchronously n-tuples of words rather than single words have been considered in the literature already (see, e.g., [Kr] or [W] for a more restricted type). However

(i) in these models there is no distinction between various roles (input, storage, output) of different components: the (pieces of) words of the language defined are computed simultaneously on all components (and then the words of the language are obtained by, e.g., catenating all words obtained on all components at the same time), and

(ii) no means (like our selective substitutions) are provided for defining different kinds of rewritings for different components.

Clearly before the true value of our model (ects systems) can be asserted, an extensive research must be carried out so that a number of basic questions can be answered. Here are examples of some problems (and problem areas) that we think are worthwhile to pursue.

(1) Various extensions of the basic ects model can be considered. For example, allowing each component to have a set (language) of axioms rather than a single axiom (of length 1) seems to be natural when the modeling of transducers is considered.

(2) Various language defining mechanisms should be compared within this model.

(3) The influence of various parameters of an ects system (e.g., the number of tables within a single component, the number of components, the type of selectors, etc.) on its "power" (e.g., the language defining power) should be investigated. What "trade offs" hold? What "normal forms" do we get?

(4) Finding and analyzing "natural" classes of selectors. From the examples given it is rather clear that sequential selectors and in particular right (or left) boundary selectors are very natural.

(5) Investigate the influence of types of selectors and properties of rewrites on

the decision problems or closure properties concerning various classes of languages associated with cts systems.

(6) While our basic model provides a unifying framework for both grammars and automata it also suggests a division line between these constructs: dimension 1 and dimension bigger than one. Hence two very central research topics from formal language theory may be now formulated in a quite general way as follows. Let $\mathbf{K}_1$ and $\mathbf{K}_2$ be classes of selectors.

(6.1) *Grammar synthesis problem.*

Let G be an arbitrary ects system of dimension $n \geq 1$ such that its selectors are in $\mathbf{K}_1$. Does there exist an "equivalent" ects system H of dimension $m = 1$ such that its selectors are in $\mathbf{K}_2$? What price in the "complexity" is paid when such a translation is possible?

(6.2) *Automaton synthesis problem.*

Let G be an arbitrary ects system of dimension $n = 1$ such that its selectors are in $\mathbf{K}_2$. Does there exist an "equivalent" ects system H with selectors in $\mathbf{K}_1$ of dimension $m \geq 1$? What price in the "complexity" is paid when such a translation is possible?

(7) How to formalize the notion of "weak" and "strong" coordination between components of an ects system?

A first step in this direction would be to allow a selector over Σ to be a subset of $(\Sigma \cup \bar{\Sigma})^*$, rather than a subset of $(\Sigma \cup \bar{\Sigma})^* \, \bar{\Sigma} (\Sigma \cup \bar{\Sigma})^*$. (As a matter of fact in this way we come back to the original notion of a selector, see, e.g., [R] and [K]). Hence a selector word may be now a word over Σ^* and "applying it" to a word under rewriting means "doing nothing". Then, e.g., if $\pi = (T_1, T_2, T_3)$ is a rewrite of an ects system G such that $sel(T_2) = (al_2(G))^*$ and for $i \in \{1,3\}$ each word in T_j contains at least one occurrence of a "barred letter", then π coordinates the rewriting on the first and the third coordinate of G without "influencing" the second coordinate of G.

(8) One can also easily accommodate the notion of concurrency within the theory of ects systems. A first approach in this direction can be sketched as follows. (We assume that the notion of a selector is defined as under (7) above.) Let $G = (D_1, \ldots, D_n, R)$, $n \geq 2$, be an ects system and let for each $1 \leq i \leq n$, $\hat{\mathrm{T}}_i$ be the set of all tables from $tab_i(G)$ that do not appear at the i'th coordinate of any rewrite of R. For $x, y \in (al_i(G))^*$ we write $x \overset{(*)}{\underset{i;G}{\Longrightarrow}} y$ if either $x = y$ or $x \overset{*}{\Longrightarrow} y$ using tables from $\hat{\mathrm{T}}_i$. Then given $u = (u_1, \ldots, u_n)$, $v = (v_1, \ldots, v_n) \in \bigtimes_{i=1}^{n} (al_i(G))^*$, $u \overset{c}{\underset{G}{\Longrightarrow}} v$ if and only if there exists a $z = (z_1, \ldots, z_n) \in \bigtimes_{i=1}^{n} (al_i(G))^*$ such that $z \underset{G}{\Longrightarrow} v$ and, for each $1 \leq i \leq n$, $u_i \overset{(*)}{\underset{i;G}{\Longrightarrow}} z_i$; in this way $u \overset{c}{\underset{G}{\Longrightarrow}} v$ corresponds to a "concurrent computation step" in G.

In this way one gets models of concurrency corresponding closely to the models from [CH] and [M].

(9) A natural way to generalize our model is to coordinate *directly* productions used in various components by considering selectors over words built up from (the names of) productions rather than over the words build up from the given alphabet. Such systems should be investigated.

We are currently working on a number of the above (and other) topics and hope to report on the results in the future.

ACKNOWLEDGEMENTS.

The author is very obliged to A. Ehrenfeucht and the Leiden TIC group (IJ.J. Aalbersberg, J. Engelfriet, D. Janssens, H.C.M. Kleijn and H.J. Hoogeboom) for very useful comments concering the first draft of this paper. The author gratefully acknowledges the support of NSF grant MCS 83-05245.

REFERENCES

[AR] Aalbersberg, IJ.J and Rozenberg, G., "cts systems and Petri nets", Techn. Rep., Institute of Applied Mathematics and Computer Science, University of Leiden, 1984.

[B] Brauer, W., ed., *Net theory and applications*, *Lecture Notes in Computer Science*, v. 84, Springer Verlag, Berlin, Heidelberg, 1980.

[Bü] Büchi, R., "Regular canonical systems", *Archiv. fur Mathematische Logik und Grundlagenforschung*, v.6, 91-111, 1964.

[CH] Campbell, R.H. and Haberman, A.N., "The specification of process synchronization by path expressions", *Lecture Notes in Computer Science*, Springer Verlag, v. 16, 89-102, 1974.

[E] Eilenberg, S., *Automata, Languages and Machines*, Academic Press, London-New York, 1974.

[G] Ginsburg, S., *Algebraic and automata - theoretic properties of formal languages*, North-Holland Publishing Co., Amsterdam, 1975.

[GR] Ginsburg, S. and Rozenberg, G., "TOL schemes and control sets", *Information and Control*, v.27, 109-125, 1974.

[Go] Goldstine, J., "Automata with data storage", in *Proceedings of a Conference on Theoretical Computer Science*, Waterloo, 239-246, 1977.

[H] Harrison, M., *Introduction to formal language theory*, Addison-Wesley, Reading, Massachusetts, 1978.

[K] Kleijn, H.C.M., "Selective substitution grammars based on context-free productions," Ph.D. Thesis, Department of Mathematics, University of Leiden, The Netherlands, 1983.

[KR1] Kleijn, H.C.M. and Rozenberg, G., "Context-free like restrictions on selective rewriting," *Theoretical Computer Science*, v. 16, 237-269, 1981.

[KR2] Kleijn, H.C.M. and Rozenberg, G., "Sequential, continuous and parallel grammars," *Information and Control*, v. 48, 221-260, 1981.

[Kr] Kral, J., "On multiple grammars", *Kybernetika*, v.5, 60-85, 1969.

[M] Mazurkiewicz, A., "A complete set of assertions on distributed systems", Tech. Rep., University of Aarhus, Dept. of Comp. Science, 1979.

[R] Rozenberg, G., "Selective substitution grammars (towards a framework for rewriting systems), Part I: Definitions and Examples", *Elektron. Informationsverarbeit Kybernetik* 13, 455-463, 1977.

[RS] Rozenberg, G. and Salomaa, A., *The mathematical theory of L systems*, Academic Press, London, New York, 1981.

[S] Salomaa, A., *Formal languages*, Academic Press, London, New York, 1973.

[Sc] Scott, D., "Some definitional suggestions for automata theory", *Journal of Computer and System Sciences*, v.1, 187-212, 1967.

[W] Wood, D., "Properties of n-parallel finite state languages", *Utilitas Mathematica* 4, 103-113, 1973.