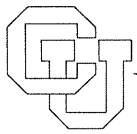
Real Time Coordinated Pair Systems

A. Ehrenfeucht, H. J. Hoogeboon G. Rozenberg*

CU-CS-259-83 September 1983



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

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

ABSTRACT

Coordinated table selective substitution systems (cts systems for short) were introduced in [R2] where it was demonstrated that they form a unifying framework for many types of grammars *and* automata (machines) considered in formal language theory. In this paper a submodel of the cps system model is considered: it is called *coordinated pair system (cp system* for short). It models very closely the work of a push-down automaton. We consider *real time cp systems*, i.e., cp systems in which each computation step of a "successful computation" generates one letter of the word defined by the computation. We demonstrate that real time cp systems constitute a normal form for the class of cp systems (as far as defined languages are concerned).

INTRODUCTION

The theory of selective substitution grammars (see, e.g., [R1], [K] and [KR]) provides a general framework for rewriting systems (grammars). It has turned out to be successful in the sense that, within this theory, several central notions of formal language theory were captured in a natural way and a number of interesting new notions (and results) have emerged.

This theory was expanded in [R2] to the theory of *coordinated table selective substitution systems (cts systems* for short) which provides a general framework for grammars *and* automata (machines). It is demonstrated in [R2] how quite many types of grammars and automata considered in the literature are easily modeled (formalized) within the framework of cts systems.

In the present paper a particular specialization of the cts system model to the case of two coordinates (one for "input" and another for "store") is considered. The model is called *coordinated pair system. (cp systems* for short). Roughly speaking a cp system G consists of

(1) two grammars G_1 and G_2 such that G_1 is a *right linear grammar* and G_2 is a *right boundary grammar* (which is like a right linear grammar in that it rewrites always the rightmost symbol of a sentential form but it does not distinguish between terminal and nonterminal symbols), and

(2) a set R of pairs of productions (called *rewrites*):, one from G_1 and one from G_2.

Then the rewriting in G consists of *coordinated* rewriting in G_1 and G_2 in the sense that at a given derivation step a production π_1 in G_1 may be used *and* a production π_2 in G_2 may be used *if* (π_1, π_2) is a rewrite in G. It is easily seen that cp systems model closely push-down automata, *however we deal with rewriting based on context-free productions rather than with transition func-*

tions in a rather involved machine model.

In this paper we consider *real time* cp systems that is cp systems in which at each step of a *computation* (where a computation is a sequence of coordinated rewritings) an "input symbol" is generated by G_1 (that is G_1 does not have chain productions). We demonstrate that the class of languages generated by real time cp systems and the class of languages generated by cp systems coincide.

Hence we get a "real-time normal form result" for (a model closely related to) push-down automata without using context-free grammars (hence without using Greibach normal form). The relationship between our method and the method for getting the same type of result for another general machine model presented in [G] is discussed in the last section.

0. PRELIMINARIES

We assume the reader to be familiar with the basic formal language theory and in particular with the theory of context-free grammars and the theory of push-down automata (see, e.g., [H] and [S]).

In order to fix the notation and the terminology for this paper, we will recall now a number of basic notions.

For a set Z, $\#Z$ denotes its cardinality.

For a word x, $|x|$ denotes its length, $mir(x)$ denotes its mirror image and, if x is nonempty, then $last(x)$ denotes the last letter of x; Λ denotes the empty word.

Let x, y be words. The *right quotient of x by y*, denoted x/y, is defined as follows: if y is a suffix of x, then $x/y = z$, where $zy = x$; otherwise x/y is not defined.

Let K, L be languages. The *right quotient of K by L*, denoted K/L, is defined by $K/L = \{z \mid zy \in K \text{ for some } y \text{ in } L\}$.

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

A *context-free grammar*, abbreviated *cf grammar*, is specified in the form $G = (\Sigma, P, S, \Delta)$ where Σ is its alphabet, P its set of productions, S its axiom and $\Delta \subseteq \Sigma$ its terminal alphabet. For $x, y \in \Sigma^*$ and $\pi \in P$ we write $x \overset{\pi}{\underset{G}{\Longrightarrow}} y$ if x directly derives y using π. Productions of the form $X \to Y$, where X and Y are nonterminal symbols (i.e., the elements of $\Sigma - \Delta$) are called *chain productions*.

A *right-linear grammar*, abbreviated *rl grammar*, is a context-free grammar $G = (\Sigma, P, S, \Delta)$, which has its productions in the set $(\Sigma - \Delta) \times \Delta^* ((\Sigma - \Delta) \cup \{\Lambda\})$.

A *derivation (in G)* is a sequence $\sigma = \sigma(0), \sigma(1), \ldots, \sigma(n)$, $n \geq 0$, of words from Σ^* such that, for every $1 \leq i \leq n$, $\sigma(i-1) \underset{G}{\overset{\pi_i}{\Longrightarrow}} \sigma(i)$, where π_i is a production from P. The sequence $\pi_1, \ldots, \pi_n$ is called the *control sequence of* σ and denoted by $cont(\sigma)$; if $n = 0$, then $cont(\sigma)$ is the empty sequence. If $\sigma(n) \in \Delta^*$, then both σ and $cont(\sigma)$ are called *successful*.

A *right-boundary grammar*, abbreviated *rb grammar*, differs from a right-linear grammar in the fact that it does not distinguish between terminal and nonterminal symbols. A rb grammar is specified in the form of a 3-tuple $G = (\Sigma, P, S)$, where Σ is its alphabet, $P \subseteq \Sigma \times \Sigma^*$ is its set of productions and $S \in \Sigma$ its axiom. As in the case of a rl grammar, productions are applied to the last occurrence in a word only. Thus, for $x, y \in \Sigma^*$ and $\pi = a \rightarrow w \in P$, x directly derives y (in G using π), written $x \underset{G}{\overset{\pi}{\Longrightarrow}} y$, if $x = za$ and $y = zw$ for some $z \in \Sigma^*$.

Two language generating "devices" are said to be *equivalent* if their languages differ at most by the empty word.

BASIC NOTIONS.

In this section the basic notion of this paper - a *coordinated pair system* - is introduced and several key notions pertinent to it are discussed.

Definition 1.1. A *coordinated pair system*, abbreviated *cp system*, is a triple $G = (G_1, G_2, R)$ such that

(1) $G_1 = (\Sigma_1, P_1, S_1, \Delta)$ is a rl grammar,

(2) $G_2 = (\Sigma_2, P_2, S_2)$ is a rb grammar, and

(3) $R \subseteq P_1 \times P_2$. ∎

Elements of R will be referred to as *rewrites* of G. If $\pi = (\pi_1, \pi_2)$ is a rewrite of G, then π_1 and π_2 are referred to as the *first* and *second production of* π respectively.

The set of all first (resp. second) productions of all rewrites in R is denoted by $proj_1(R)$(resp. $proj_2(R)$). In this paper we assume that $P_1 = proj_1(R)$ and $P_2 = proj_2(R)$.

Definition 1.2. Let $G = (G_1, G_2, R)$ be a cp system, where $G_1 = (\Sigma_1, P_1, S_1, \Delta)$ and $G_2 = (\Sigma_2, P_2, S_2)$.

(1) Let $x = (x_1, x_2)$, $y = (y_1, y_2) \in \Sigma_1^* \times \Sigma_2^*$. x *directly computes* y *(in* G*)*, denoted $x \underset{G}{\Longrightarrow} y$, if there exists a rewrite $\pi = (\pi_1, \pi_2) \in R$ such that $x_1 \overset{\pi_1}{\underset{G_1}{\Longrightarrow}} y_1$ and $x_2 \overset{\pi_2}{\underset{G_2}{\Longrightarrow}} y_2$; we write then $x \overset{\pi}{\underset{G}{\Longrightarrow}} y$ and we say that x *directly computes* y *(in* G*) using* π.

$\overset{*}{\underset{G}{\Longrightarrow}}$ denotes the reflexive and transitive closure of $\underset{G}{\Longrightarrow}$. If $x \overset{*}{\underset{G}{\Longrightarrow}} y$, then we say that x *computes* y *(in* G*)*.

(2) Let $(x, y) \in \Sigma_1^+ \times \Sigma_2^+$. A (x,y)*–computation (in* G*)* is a sequence

$\rho = \rho(0), \ldots, \rho(n)$ of elements from $\Sigma_1^* \times \Sigma_2^*$, $n \geq 0$, such that $\rho(0) = (x, y)$ and , for $1 \leq i \leq n$, $\rho(i-1) \underset{G}{\Longrightarrow} \rho(i)$. ρ is an *n-step* computation and, for $1 \leq i \leq n$, $\rho(i-1) \underset{G}{\Longrightarrow} \rho(i)$ is called the *i-th step of* ρ.

For each $0 \leq i \leq n$, $\rho(i)$ is referred to as a *snapshot of* ρ. If ρ is a (S_1, S_2)-computation in G, then $\rho(i)$ is a *snapshot (of G)*.

(3) Let $\rho = \rho(0), \ldots, \rho(n)$, $n \geq 1$, be a computation in G. The sequence $\pi_1, \pi_2, \ldots, \pi_n$ of rewrites from R such that, for $1 \leq i \leq n$, $\rho(i-1) \overset{\pi_i}{\underset{G}{\Longrightarrow}} \rho(i)$ is called the *control sequence of* ρ *and* denoted by $cont(\rho)$. If $\rho = \rho(0)$, then we define $cont(\rho)$ to be the empty sequence.

(4) Let $\rho = \rho(0), \ldots, \rho(n)$, $n \geq 0$, be a computation in G. ρ is a *successful computation* if $\rho(0) = (S_1, S_2)$ and $\rho(n) = (w, \Lambda)$ for some $w \in \Delta^*$. Then the *result of* ρ, denoted $res(\rho)$, is defined by $res(\rho) = w$.

(5) The *language of* G, denoted $L(G)$, is defined by $L(G) = \{res(\rho) \mid \rho$ is a successful computation in $G\}$. ■

The language of a cp system is referred to as a *cp language*.

In order to avoid separate considerations of "trivial" cases we will assume that every cp system considered in this paper "computes" at least one nonempty word.

2. MINIMIZING THE ROLE OF THE FIRST COMPONENT.

In this section two normal forms for cp systems are presented. The idea of the transformations considered in this section is to minimize the role of the first component in successful computations as much as possible.

We start with a definition that formalizes several ways of making the first component of a cp system "irrelevant". A cp system is called *simple* if the nonterminals of the first component do not influence the computations: one cannot distinguish "states" (there is one nonterminal only) and moreover, if it is possible to finish a computation on the second component then it is also possible (by choosing the appropriate rewrite) to end the computation on the first component.

We say that a cp system is *consistent* if its computations are such that both components "finish" at the same time; thus a snapshot of a computation has a terminal word on the first component if and only if its second component is empty.

Definition 2.1. Let $G = (G_1, G_2, R)$ be a cp system, where $G_1 = (\Sigma_1, P_1, S_1, \Delta)$, $G_2 = (\Sigma_2, P_2, S_2)$ and $\Gamma = \Sigma_1 - \Delta$.

(1) G is *1-minimal* if $\#\Gamma = 1$.

(2) G is *simple* if G is 1-minimal and moreover, for each $\pi_2 \in P_2$ and each $w \in \Delta^*$, $(S_1 \to w, \pi_2) \in R$ if and only if $(S_1 \to wS_1, \pi) \in R$.

(3) G is *consistent* if, for every snapshot (u_1, u_2) of G, $u_1 \in \Delta^*$ if and only if $u_2 = \Lambda$. ∎

Theorem 2.1. For each cp system G there exists an equivalent 1-minimal and consistent cp system.

Proof.

Let $G = (G_1 , G_2 , R)$ be a cp system, where

$G_1 = (\Sigma_1, P_1 , S_1 , \Delta)$, $G_2 = (\Sigma_2 , P_2 , S_2)$ and let $\Gamma = \Sigma_1 - \Delta$.

Then let $\overline{G} = (\overline{G}_1 , \overline{G}2 , \overline{R})$ be a cp system, where

$\overline{G}_1 = (\overline{\Sigma}_1 , \overline{P}_1 , \overline{S}_1 , \Delta)$ and $G_2 = (\overline{\Sigma}_2 , \overline{P}_2 , \overline{S}_2)$ with

$\overline{\Sigma}_1 = \Delta_1 \cup \{\overline{S}_1\}$, $\overline{S}_1 \notin \Delta_1$,

$\overline{\Sigma}_2 = \{[X, Y, Z] \mid X \in \Sigma_2 , Y \in \Gamma \cup \{\Delta\}$ and $Z \in \Gamma\}$,

$\overline{S}_2 = [S_2 , \Lambda , S_1]$ and

$\overline{R}$, $\overline{P}_1$ and $\overline{P}_2$ defined as follows.

For each rewrite $\pi = (\pi_1 , \pi_2)$ in R we construct the set $\overline{R}(\pi)$ of rewrites in the following way.

(1) If $\pi_1 = B \rightarrow wC$, $w \in \Delta^*$ and $C \in \Gamma$, then

(1.1) if $\pi_2 = D \rightarrow D_1 \cdots D_n$, $n \geq 1$, $D_i \in \Sigma_2$ for all $1 \leq i \leq n$, then $\overline{R}(\pi)$ consists of all rewrites

$(\overline{S}_1 \rightarrow w\overline{S}_1 , [D, X, B] \rightarrow [D_1 , X, X_1][D_2 , X_1 , X_2] \cdots [D_n , X_{n-1} , C])$

such that $X \in \Gamma \cup \{\Lambda\}$ and $X_1 , \ldots , X_{n-1} \in \Gamma$,

(1.2) if $\pi_2 = D \rightarrow \Lambda$, then $\overline{R}(\pi)$ consists of the single rewrite

$(\overline{S}_1 \rightarrow w\overline{S}_1 , [D, C, B] \rightarrow \Lambda)$.

(2) If $\pi_1 = B \rightarrow w$, $w \in \Delta^*$ and π_2 is as under (1.1), then $\overline{R}(\pi) = \emptyset$, otherwise, if π_2 is as in (1.2), then $\overline{R}(\pi)$ consists of the single rewrite $(\overline{S}_1 \rightarrow w, [D, \Lambda, B] \rightarrow \Lambda)$.

Then $\overline{R} = \bigcup_{\pi \in R} \overline{R}(\pi)$, $\overline{P}_1 = proj_1(R)$ and $\overline{P}_2 = proj_2(\overline{R})$.

From the construction of $\overline{G}$ it easily follows that $\overline{G}$ is both 1-minimal and consistent.

To show that $L(G) = L(\overline{G})$ we proceed as follows. (Given a letter $\sigma \in [X, Y, Z] \in \overline{\Sigma}_2$ we use $1(\sigma)$, $2(\sigma)$ and $3(\sigma)$ to denote X, Y and Z respectively).

(i) $L(G) \subseteq L(\overline{G})$.

Let ρ be a m-step successful computation, such that, for all $0 \leq i \leq m-1$, $\rho(i) = (w_i B_i\ ,\ D_{i,1} \cdots D_{i,n_i})$ with $w_i \in \Delta^*$, $B_i \in \Gamma$ and $D_{i,j} \in \Sigma_2$ for $1 \leq j \leq n_i$, and $\rho(m) = (w_m, \Lambda)$ with $w_m \in \Delta^*$.

We will construct a m-step computation $\overline{\rho}$ in $\overline{G}$ as follows.
First of all we require that, for each $0 \leq i \leq m-1$, $\overline{\rho}(i) = (w_i \overline{S}_1\ ,\ F_{i,1} \cdots F_{i,n_i})$ with $F_{i,j} \in \overline{\Sigma}_2$ for $1 \leq j \leq n_i$ and $2(F_{i,k+1}) = 3(F_{i,k})$ for all $1 \leq k \leq n_i - 1$.

Thus to specify $\overline{\rho}(i)$ it suffices to specify
$1(F_{i,1}), \ldots, 1(F_{i,n_i}), 3(F_{i,1}), \ldots, 3(F_{i,n_i})$ and $2(F_{i,1})$, which is done in the following way.
$1(F_{i,j}) = D_{i,j}$ for $1 \leq j \leq n_i$ and $2(F_{i,1}) = \Lambda$.
For each $1 \leq k \leq n_i$ let $q_k \geq i$ be the smallest integer such that in the (q_k+1)st computation step the occurrence $D_{q_k,k} = D_{i,k}$ is rewritten on the second component of $\rho(q_k)$. Then $3(F_{i,k}) = B_{q_k}$. Note that $q_{n_i} = i$.

Let $\overline{\rho}(m) = \rho(m)$.

From the definition of $\overline{R}$ it easily follows that $\overline{\rho}$ is successful. Thus $res(\overline{\rho}) = res(\rho) \in L(\overline{G})$ and consequently $L(G) \subseteq L(\overline{G})$.

(ii) $L(\overline{G}) \subseteq L(G)$. Consider a m-step successful computation $\overline{\rho}$ in $\overline{G}$. Note that by point (1.1) of the definition of $\overline{R}$, for each $0 \leq i \leq m-1$, if $\overline{\rho}(i) = (w_i \overline{S}_1\ ,\ F_{i,1} \cdots F_{i,n_i})$ where $F_{i,1}, \ldots, F_{i,n_i} \in \overline{\Sigma}_2$, then $2(F_{i,k+1}) = 3(F_{i,k})$ for all $1 \leq k < n_i$.

Now we construct a successful m-step computation ρ in G as follows: for $0 \leq i < m$, $\rho(i) = (w_i 3(F_{i,n_i}),\ 1(F_{i,1}) \cdots 1(F_{i,n_i}))$ and $\rho(m) = (w_m\ ,\ \Lambda)$.

From the observation above and the definition of $\overline{R}$ it easily follows that ρ is a computation in G (and moreover, for all $1 \leq i \leq m$, if $\overline{\rho}_{i-1} \underset{\overline{G}}{\Longrightarrow} \rho_i$ using a

rewrite $\overline{\pi}$ in $\overline{R}(\pi)$, then $\rho_{i-1} \underset{G}{\Longrightarrow} \rho_i$ using the rewrite π).

Consequently $L(\overline{G}) \subseteq L(G)$.

From (i) and (ii) it follows that $L(\overline{G}) = L(G)$ and so the theorem holds. ∎

Theorem 2.2. For each cp system G there exists an equivalent simple cp system.

Proof.

Let $G = (G_1 , G_2 , R)$ be a cp system, where $G_1 = (\Sigma_1 , P_1 , S_1 , \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$. By Theorem 2.1 we may assume that G is both minimal and consistent.

We will construct now a cp system $\tilde{G}$ which differs from G only in that productions and rewrites are changed in the following way.

From each $\pi \in R$, $\tilde{R}(\pi)$ is the set consisting of two rewrites, defined as follows.

If either $\pi = (S_1 \to w , \pi_2)$ or $\pi = (S_1 \to wS_1 , \pi_2)$ for some $w \in \Delta^*$, then $\tilde{R}(\pi) = \{(S_1 \to w, \pi_2) , (S_1 \to wS_2 , \pi_2)\}$.

Let $\tilde{R} = \bigcup_{\pi \in R} \tilde{R}(\pi)$ and let $\tilde{G} = (\tilde{G}_1 , \tilde{G}_2, \tilde{R})$, with $\tilde{G}_1 = (\Sigma_1 , proj_1(\tilde{R}), S_1, \Delta)$ and $\tilde{G}_2 = (\Sigma_2 , proj_2(\tilde{R}), S_2)$.

Then clearly $\tilde{G}$ is simple.

To see that $L(G) = L(\tilde{G})$ we proceed as follows.

(i) $L(G) \subseteq L(\tilde{G})$.

This is obvious because $\tilde{G}$ results by "expanding" G.

(ii) $L(\tilde{G}) \subseteq L(G)$.

Assume that there exists a word $x \in L(\tilde{G})$ such that $x \notin L(G)$. Thus there exists a m-step successful computation ρ in $\tilde{G}$ such that $res(\rho) = x$.

For some $1 \leq i \leq m$ $\rho(i-1) \underset{G}{\Longrightarrow} \rho(i)$ does not hold, because ρ is not a computation in G.

Assume that we have chosen the minimal i with the above property.

If $i < m$, then to obtain $\rho(i)$ from $\rho(i-1)$ in $\tilde{G}$ a rewrite $\pi = (S_1 \rightarrow wS_1 , \pi_2)$ in $\tilde{R}-R$ was applied. Thus (by the definition of $\tilde{R}$), the rewrite $(S_1 \rightarrow w, \pi_2)$ is an element of R, which contradicts the fact that G is consistent.

Similarly we get a contradiction in the case of $i = m$. Thus $L(\tilde{G}) \subseteq L(G)$.

Consequently $L(G) = L(\tilde{G})$ and so the theorem holds. ∎

We would like to make the following "side comment".

As a "by-product" of the above theorem we can prove that cp languages are context-free (since cp systems model so closely push-down automata this is also implied by the classic results of formal language theory).

The proof at this point is really easy because we have made the first component irrelevant and so one component only will suffice to get the language of a cp system - in this way we get the desired context-free grammar.

Corollary 2.3. If G is a cp system, then $L(G)$ is context-free.

Proof.

Let $G = (G_1 , G_2 , R)$, where $G_1 = (\Sigma_1 , P_1 , S_1 , \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$ be a cp system. We may assume that G is simple.

Let $H = (\Sigma, P, S, \Delta)$ be the context-free grammar such that
$\Sigma = \Sigma_2 \cup \Delta$, $S = S_2$ and

$P = \{X \to w\ mir(u) \mid R$ contains the rewrite $(S_1 \to wS_1\ ,\ X \to u)$

for some $w \in \Delta^*$, $X \in \Sigma_2$, $u \in \Sigma_2^*\}$.

It is easily seen that: for each successful computation ρ in G one can construct a derivation of $res(\rho)$ in H and conversely, for each leftmost derivation τ of a word w in $L(H)$ one can construct a successful computation ρ in G such that $res(\rho) = w$. Hence $L(G) = L(H)$ and the corollary holds. ∎

3. MAKING INTERNAL REWRITES GROWING.

In this, rather short, section we will show how one can transform a cp system in such a way that rewrites violating the real-time restriction cause the growth of the second component of the snapshot being rewritten.

Definition 3.1. Let $G = (G_1 , G_2 , R)$ be a cp system with $G_1 = (\Sigma_1 , P_1 , S, \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$.

(1) A rewrite $\pi = (A \to w, \pi_2)$ of G is called *internal* if $pres_\Delta(w) = \Lambda$; otherwise π is called *external*.

(2) An internal rewrite $\pi = (A \to w, X \to u)$ of G is *2-growing* if $|u| \geq 2$.

We say that G is 2-growing if each internal rewrite of G is 2-growing.

(3) G is called *2-nonblocking* if for each $A \in \Sigma_2$, $A \overset{*}{\underset{G_2}{\Longrightarrow}} \Lambda$. ∎

Remark. (1) If we deal with a simple cp system G and $A \overset{*}{\underset{G_2}{\Longrightarrow}} \Lambda$, then there exists a successful (S_1 , A)-computation in G (provided our assumption $P_2 = proj_2(R)$ holds). It is easily seen that 2-nonblocking cp system form a normal form for the class of simple cp system.

(2) If G does not have internal rewrites, then G is called a *real time* cp system. In this case, if ρ is a successufl computation with $u = res(\rho)$, then ρ is at most a $|u|$-step computation; hence u is computed (generated on the first coordinate) in real time. ∎

The main idea in the proof of the following lemma follows closely the classical proof that one can remove chain and erasing productions from context-free grammars (see, e.g., [S]).

Lemma 3.1. For each cp system G there exists an equivalent cp system G' that is simple, 2-nonblocking and 2-growing.

Proof.

Let $G = (G_1 , G_2 , R)$ be a cp system. According to Theorem 2.2 we may assume that G is simple. Moreover we assume that G is 2-nonblocking: it is easily seen that after the removal of rewrites which have "blocking" variables on the second component the resulting cp system is simple if the original cp system was simple.

Let $G_1 = (\Sigma_1 , P_1 , S_1 , \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$.

We divide the letters in Σ_2 into three categories.

Category 1 consists of all letters $A \in \Sigma_2$ satisfying the following property: if ρ is a successful (S_1 , A)-computation, then $cont(\rho)$ consists of internal rewrites only.

Category 2 consists of all letters $A \in \Sigma_2$ that are not of category 1, but for which there exists a successful (S_1 , A)-computation ρ such that $cont(\rho)$ contains only internal rewrites.

All letters that are neither of category 1 nor of category 2 belong to *category* 3.

Thus, categories 1, 2 and 3 contain letters that in successful computations are *always* (respectively *sometimes*, *never)* rewritten on the second component without contributing terminal symbols to the first component.

Now G' results from G by performing the following construction.

Step 1.

Erase all letters of category 1 from the right-hand sides of all second productions of all rewrites in R. Any internal rewrite resulting in this way such that its second production is an erasing production is removed. Note that in this way all rewrites with letters from category 1 on the left-hand side of their second production are also removed.

Let R_1 be the resulting set of rewrites.

Step 2. Each rewrite $(\pi_1 , X \to u)$ from R_1 is replaced by the set of rewrites of the form $(\pi_1 , X \to \overline{u})$, where $\overline{u} \in map_2(u)$ and map_2 is the finite substitution of Σ_2^* defined by:

$$map_2(a) = \begin{cases} \{a, \Lambda\} & \text{, if } a \text{ is a letter of category 2,} \\ a & \text{, otherwise.} \end{cases}$$

Any internal rewrite resulting in this way such that its second production is an erasing production is removed.

Let R_2 be the resulting set of rewrites.

Step 3.

We say that an internal rewrite is a *chain rewrite* if it is of the form $(S_1 \to S_1 , A \to B)$ for some $A, B \in \Sigma_2$. For every $A \in \Sigma_2$ let $chain(A)$ be the set of all $B \in \Sigma_2$ such that $(S_1 , A) \overset{*}{\Longrightarrow} (S_1 , B)$ using chain rewrites form R_2 only. Then for every $A \in \Sigma_2$ and every $B \in chain(A)$ we add to R_2 the rewrites $(\pi_1 , A \to w)$, where $(\pi_1 , B \to w) \in R_2$ is either an external rewrite or it is 2-growing.

Finally we remove from R_2 all chain rewrites; let R' be the resulting set of rewrites. Let $P_1' = proj_1(R')$, $P_2' = proj_2(R')$ and let Σ_2' be the set of letters occurring as left-hand side of productions in P_2'.

Then let $G' = (G_1' , G_2' , R')$, where $G_1' = (\Sigma_1' , P_1' , S_1 , \Delta)$ and $G_2' = (\Sigma_2' , P_2' , S_2)$.

It is easily seen that $L(G') = L(G)$ and so the lemma holds. ∎

4. REMOVING INTERNAL REWRITES.

In this section we prove the main result of this paper: internal rewrites may be removed form cp systems without changing the class of languages generated.

We start by introducing a number of important technical notions needed in the proof of our main theorem.

Definition 4.1. Let $G = (G_1 , G_2 , R)$ be a simple cp system where $G_1 = (\Sigma_1 , P_1 , S_1 , \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$.

(1) An *input segment (of G)* is a word $u \in \Delta^+$ such that R contains a rewrite $(S_1 \rightarrow uS_1 , \pi_2)$ for some production $\pi_2 \in P_2$.

(2) Let $A \in \Sigma_2$ and let u be an input segment. Then $Q_{A,u}$ is the set of all words w such that $(S_1 , A) \overset{*}{\underset{G}{\Longrightarrow}} (S_1 , x) \underset{G}{\Longrightarrow} (uS_1 , w)$ for some $x \in \Sigma_2^*$.

(3) Let $\mathbf{Q}(G) = \{\{S_2\}\} \cup \{Q_{A,u} \mid A \in \Sigma_2$ and u is an input segment $\}$.

Let $\hat{\mathbf{C}}(G)$ be the closure of $\mathbf{Q}(G)$ with respect to right quotients with letters from Σ_2 and let $\mathbf{C}(G) = \hat{\mathbf{C}}(G) - \{\emptyset\}$ ■

The following result follows easily from the above definitions.

Lemma 4.1. Let G be a simple 2-growing cp system.

(1) Each element of $\mathbf{Q}(G)$ is a regular language.

(2) $\mathbf{C}(G)$ is a finite family of regular languages. ■

We are now able to prove our main theorem.

Theorem 4.2. For each cp system G there exists an equivalent cp system G' that is simple and which has no internal rewrites.

Proof. (The reader may wish to read this proof together with Example 4.1 which illustrate the main constructions we use.)

Let $G = (G_1 , G_2 , R)$ be a cp system, where $G_1 = (\Sigma_1 , P_1 , S_1 , \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$. We assume that G is simple, 2-growing and 2-nonblocking; according to Lemma 3.1 we can do it without loss of generality.

Let f be a bijective mapping from $\mathbf{C}(G)-\{\{\Lambda\}\}$ onto a new alphabet Θ (elements of Θ are used to name sets from $\mathbf{C}(G)-\{\{\Lambda\}\}$). For convenience we define $f(\{\Lambda\}) = \Lambda$.

A new set of rewrites R' is defined as follows.

For each $Q \in \mathbf{C}(G)$, each $A \in \Sigma_2$ and each input segment u such that Q/A and $Q_{A,u}$ are nonempty, R' contains the rewrites

(I) $(S_1 \to uS_1 , f(Q) \to f(Q/A)f(Q_{A,u}))$ and $(S_1 \to u , f(Q) \to f(Q/A)f(Q_{A,u}))$,

(II) $(S_1 \to uS_1 , f(Q) \to f(Q_{A,u}))$ and $(S_1 \to u , f(Q) \to f(Q_{A,u}))$ if $\Lambda \in Q/A$,

(III) $(S_1 \to uS_1 , f(Q) \to f(Q/A))$ and $(S_1 \to u , f(Q) \to f(Q_{A,u}))$ if $\Lambda \in Q_{A,u}$, and

(IV) $(S_1 \to uS_1 , f(Q) \to \Lambda)$ and $(S_1 \to u , f(Q) \to \Lambda)$ if $\Lambda \in Q/A$ and $\Lambda \in Q_{A,u}$.

Now let $G' = (G_1' , G_2' , R')$ with $G_1' = (\Sigma_1 , P_1' , S_1 , \Delta)$ and $G_2' = (\Theta, P_2' , f(\{S_1\}))$, where $P_1' = proj_1(R')$, $P_2' = proj_2(R')$.

Note that Lemma 4.1 guarantees that G' is well-defined.

Clearly G' is simple and moreover G' does not have internal rewrites. Hence to prove the lemma it suffices to prove that $L(G') = L(G)$. This is done as follows.

(i) $L(G) \subseteq L(G')$.

Consider a successful n-step computation ρ in G and let for each $0 \le i < n$, $\rho(i) = (v_i S_i , w_i)$ and $\rho(n) = (v_n , w_n)$. Thus $w_n = \Lambda$.

For $1 \le i \le n$, the snapshot $\rho(i)$ is *external in* ρ if $|v_i| > |v_{i-1}|$. Moreover $\rho(0) = (S_1 , S_2)$ is considered to be an external in ρ. Then the subsequence of $\rho(i_0) , \rho(i_1), \ldots, \rho(i_m)$ of ρ consisting of all snpashots of ρ that are external in ρ is called the *external subsequence of* ρ and is denoted by $ext(\rho)$.

We observe immediately that $i_m = n$. This follows from the fact that in the last step of ρ a rewrite π is used such that its second production is an erasing production; since each internal rewrite of G is 2-growing $\rho(n)$ must be external in ρ.

Consider the j-th step in ρ, where $i_{k-1} < j < i_k$ for some $k \in \{1, \ldots, m\}$. Since in this step an internal rewrite is used and G is a 2-growing cp system, $|w_{j-1}| < |w_j|$.

So for each $k \in \{1, \ldots, m\}$ it is possible to find a word $w \in \Sigma_2^*$ such that $w_{i_k} = (w_{i_{k-1}} \backslash A) \cdot w$, where $A = last(w_{i_{k-1}})$. Note that w can be empty only if $i_k = i_{k-1}+1$.

Let $\tau = (v, w)$ or $\tau = (vS_1, w)$, for some $v \in \Delta^*$, $w \in \Sigma_2^*$ be a snapshot of ρ. A *parse* of τ is a pair $(u_1, \ldots, u_s ; z_1, \ldots, z_t)$ with $s, t \geq 0$, $u_1, \ldots, u_s \in \Delta^+$ and $z_1, \ldots, z_t \in \Sigma_2^+$, such that $v = u_1 \cdots u_s$ if $s \geq 1$, $v = \Lambda$ if $s = 0$, $w = z_1 \cdots z_t$ if $t \geq 1$ and $w = \Lambda$ if $t = 0$.

We will use ε to dencte the empty sequence.

Now, based on $ext(\rho)$, we construct a computation $\mu = \mu(0), \ldots, \mu(m)$ in G.

In order to see clearly how μ relates to our original computation ρ we will assign to each external snapshot τ of ρ a "canonical" parse $can(\tau)$; $\mu(l)$ and $can(\rho(i_l))$ are related in the following obvious way.

If $0 \leq l \leq m$, then $can(\rho(i_l)) = (u_1, \ldots, u_l ; z_1, \ldots, z_t)$ and $\mu(l) = (u_1 \cdots u_l S_1, f(Q) \cdots f(Q_t))$ for some $t > 0$ elements $Q_1, \ldots, Q_t$ of $\mathbf{C}(G)$ such that $z_j \in Q_j$ for $1 \leq j \leq t$.

Moreover $can(\rho(i_m)) = (u_1, \ldots, u_m ; \varepsilon)$ and $\mu(m) = (u_1 \cdots u_m, \Lambda)$.

Let $\mu(0) = (S_1, f(\{S_2\}))$ and $can(\rho(i_0)) = (\varepsilon ; S_2)$.

Assume now that for an l, $0 \leq l < m$, $\mu(l)$ and $can(\rho(i_l))$ are defined. Let $can(\rho(i_l)) = (u_1, \ldots, u_l ; z_1, \ldots. z_t)$ and $\mu(l) = (u_1 \cdots u_l S_1, f(Q_1) \cdots f(Q_t))$.

Then $can(\rho(i_{l+1})) = (u_1, \ldots, u_l, u_{l+1}; \alpha)$, where $u_1 \cdots u_l u_{l+1} = v_{i_{l+1}}$ and $\mu(l+1)$ is obtained from $\mu(l)$ by applying the rewrite π to $\mu(l)$.

Here α and π are defined case-wise as follows.

Let $A = last(z_t) = last(w_{i_l})$ and let z be such that $(w_{i_l} \backslash A) \cdot z = w_{i_{l+1}}$.

(1) If $z_t \backslash A \neq \Lambda$ and $z \neq \Lambda$, then $\alpha = (z_1, \ldots, z_{t-1}, z_t \backslash A, z)$ and $\pi = (s_1 \rightarrow u_{l+1} S_1, f(Q_t) \rightarrow f(Q_t \backslash A) f(Q_{A,u_{l+1}})$

(2) If $z_t \backslash A = \Lambda$ and $z \neq \Lambda$, then $\alpha = (z_1, \ldots, z_{t-1}, z)$ and $\pi = (S_1 \rightarrow u_{l+1} S_1, f(Q_t) \rightarrow f(Q_{A,u_{l+1}}))$.

(3) If $z_t \backslash A \neq \Lambda$ and $z = \Lambda$, then $\alpha = (z_1, \ldots, z_{t-1}, z_t \backslash A)$ and $\pi = (S_1 \rightarrow u_{l+1} S_1, f(Q_t) \rightarrow f(Q_t \backslash A))$.

(4) If $z_t \backslash A = \Lambda$ and $z = \Lambda$, then $\alpha = (z_1, \ldots, z_{t-1})$ and $\pi = (S_1 \rightarrow u_{l+1}, f(Q_t) \rightarrow \Lambda)$, if $l+1 = m$, $\pi = (S_1 \rightarrow u_{l+1} S_1, f(Q_t) \rightarrow \Lambda)$, otherwise.

It is easily seen that the given relation between $\rho(i_l)$, $can(\rho(i_l))$ and $\mu(l)$ generally holds.

Hence we conclude that μ is a well defined successful computation in G' and $res(\mu) = res(\rho)$.

Thus $L(G) \subseteq L(G')$.

(ii) $L(G') \subseteq L(G)$.

Let $\mu = \mu(0), \ldots, \mu(m)$ be a successful computation in G', where, for $0 \leq i < m$, $\mu(i) = (v_i S_1, f(Q_{i,1}) \cdots f(Q_{i,n_i}))$ with $v_i \in \Delta^*$, $n_i \geq 1$ and $Q_{i,j} \in \mathbf{C}(G) - \{\{\Lambda\}\}$ for $1 \leq j \leq n_i$ and $\mu(m) = (v_m, \Lambda)$, $v_m \in \Delta^+$.

We shall show that now there exists a successful computation ρ in G such that $res(\rho) = res(\mu)$. In order to do that, we assign to each $\mu(i)$, $0 \leq i \leq m$, a parse $p(i) = (\alpha_i ; \beta_i)$ of a snapshot in G.

As before we have the following relation between the parse $p(i)$ and the corresponding snapshot $\mu(i)$.

For each $0 \leq i \leq m$, $p(i) = (u_1 , \cdots , u_i ; z_{i,1} , \ldots , z_{i,n_i})$, where $z_{i,j} \in Q_{i,j}$ for $1 \leq j \leq n_i$ (here we take $n_m = 0$).

First we give the construction of the sequences α_l.

Let $\alpha_0 = \varepsilon$ (the empty sequence). We proceed inductively.

If $\alpha_l = (u_1 , \ldots , u_l)$, $0 \leq l < m$, then $\alpha_{l+1} = (u_1 , \ldots , u_l , u_{l+1})$, where $u_1 \cdots u_l u_{l+1} = \vartheta_l$.

The sequences β_l are defined inductively in a "bottom-up" fashion.

Thus we define $\beta_m = \varepsilon$.

For $1 \leq l \leq m$, β_{l-1} depends on β_l as well as on the production π_l used in the l-th step of the computation μ. Let $\beta_l = (z_1 , \ldots , z_t)$ for some $t \geq 0$.

We distinguish between the following cases; they correspond to points (I) through (IV) in the definition of R'. $Q \in \mathbf{C}(G)$, $A \in \Sigma_2$ and $u \in \Delta^+$ are assumed to be chosen appropriately.

(1) If $\pi_l = (S_1 \to uS_1 , f(Q) \to f(Q/A) f(Q_{A,u}))$, then $\beta_{l-1} = (z_1 , \ldots , z_{t-1} A)$.

(2) If $\pi_l = (S_1 \to uS_1 , f(Q) \to f(Q_{A,u}))$, then $\beta_{l-1} = (z_1 , \ldots , z_{t-1}, A)$.

(3) If $\pi_l = (S_1 \to uS_1 , f(Q) \to f(Q/A))$, then $\beta_{l-1} = (z_1 , \ldots , z_t A)$.

(4) If $\pi_l = (S_1 \to uS_1 , f(Q) \to \Lambda)$ or, in the case that $l = m$, if $\pi_l = (S_1 \to u , f(Q) \to \Lambda)$, then $\beta_{l-1} = (z_1 , \ldots , z_t , A)$.

Finally, for $0 \leq i < m$, we define $\rho(i) = (u_1 \cdots u_i S_1 , z_{i,1} \cdots z_{i,n_i})$ and $\rho(m) = (u_1 \cdots u_m , \Lambda)$.

Thus in particular we have $\rho(0) = (S_1 , S_2)$, because $\mu(0) = (S_1 , f(\{S_2\}))$ and consequently $p(0) = (\varepsilon ; S_2)$.

Now, using the definition of the sets $Q_{A,u}$, it can be shown that $\rho(i-1) \overset{*}{\underset{G}{\Longrightarrow}} \rho(i)$ for all $1 \leq i \leq m$. Hence $(S_1 , S_2) = \rho(0) \overset{*}{\underset{G}{\Longrightarrow}} \rho(m) = (v_m , \Lambda)$.

Consequently $res(\mu) \in L(G)$ and so $L(G') \subseteq L(G)$. Thus the theorem holds. ∎

Example 4.1. We will illustrate now the construction used in the proof of the above theorem.

Let $G = (G_1, G_2, R)$ be the cp system where $G_1 = (\{S, a, b\}, P_1, S, \{a, b\})$, $G_2 = (\{A,B\}, P_2, S)$ and R contains the following rewrites:

$(S \to S, A \to BA)$, $(S \to S, A \to AB)$, $(S \to S, B \to ABA)$,

$(S \to aS, A \to \Lambda)$, $(S \to aS, B \to A)$, $(S \to bS, B \to \Lambda)$.

Since we assume that G is simple, for each of the above mentioned rewrites $(S \to uS, \pi_2)$, R contains the rewrite $(S \to u, \pi_2)$.

The following transition diagram gives all sets $Q_{X,u}$ for $X \in \{A,B\}$ and $u \in \{a,b\}$:

Figure 1.

For example to find out what $Q_{A,a}$ is we consider the node A as the initial node, a as the final node and then the language of the *so* resulting finite automaton yields $Q_{A,a}$.

Hence it is easily seen that

$Q_1 = Q_{A,a} = L^*\{\Lambda,AA\}$,

$Q_2 = Q_{a,b} = L^*\{A\}$,

$Q_3 = Q_{B,a} = \{A\} \cup \{AB\}L^*\{\Lambda,AA\}$ and

$Q_4 = Q_{B,b} = \{\Lambda\} \cup \{AB\}L^*\{A\}$, where $L = \{B,AAB\}$.

Computing right quotients yields

$Q_1/A = L^*\{A\} = Q_2$, $Q_1/B = L^*\{\Lambda,AA\} = Q_1$,

$Q_2/A = L^* = T$, $Q_2/B = \emptyset$,

$Q_3/A = Q_4$, $Q_3/B = Q_3$,

$Q_4/A = \{AB\}L^* = U$, $Q_4/B = \emptyset$,

$T/A = \emptyset$, $T/B = Q_1$,

$U/A = \emptyset$ and $U/B = Q_3$.

In order not to complicate the notation we use Q_i , T and U to denote the languages Q_i , T and U as well as their names $f(Q_i)$, $f(T)$ and $f(U)$.
Let $f(\{A\}) = S_2$ and, as in the proof, $f(\{\Lambda\}) = \Lambda$.

Now using rule (I) from the description of R' in the proof above we get the following rewrite in R'. (In the list below we use (u, π_2) as shorthand for the two rewrites $(S_1 \to uS_1 , \pi_2)$ and $(S_1 \to u , \pi_2)$.)

$(a, Q_1 \to Q_2Q_1)$ $(a, Q_3 \to Q_4Q_1)$ $(a, S \to Q_1)$
$(b, Q_1 \to Q_2Q_2)$ $(b, Q_3 \to Q_4Q_2)$ $(b, S \to Q_2)$
$(a, Q_1 \to Q_1Q_3)$ $(a, Q_3 \to Q_3Q_3)$
$(b, Q_1 \to Q_1Q_4)$ $(b, Q_3 \to Q_3Q_4)$

$(a, Q_2 \to TQ_1)$ $(a, Q_4 \to UQ_1)$
$(b, Q_2 \to TQ_2)$ $(b, Q_4 \to UQ_2)$

$(a, T \to Q_1Q_3)$ $(a, U \to Q_3Q_3)$
$(b, T \to Q_1Q_4)$ $(b, U \to Q_3Q_4)$

The other rewrites (those constructed using rules (II) through (IV) of the description of R' in the proof of the theorem) can be obtained from the above rewrites by erasing in the second productions names of the languages that contain the empty word, e.g., from $(a, Q_2 \to TQ_1)$ one obtains the rewrites $(a, Q_2 \to T)$, $(a, Q_2 \to Q_1)$ and $(a, Q_2 \to \Lambda)$, where we use our "shorthand notation" again.

The following diagram illustrates a computation ρ of a^3ba^3ba in G together with parses of its external snapshots and a corresponding computation μ in G'.

Figure 2.

It is instructive to notice that if we are given a cp system G and according to the proofs of Theorem 2.1, Theorem 2.2, Lemma 3.1 and finally Theorem 4.2 we construct an equivalent cp system G' (which is simple and has no internal rewrites), then the following holds.

If a positive integer k bounds form above the length of any input segment of G (the meaning of the term "an input segment" for non-simple cp system should be clear) then also the length of any input segment of G' is bounded by k. Moreover without loss of generality we may assume that each (nonempty) input segment of our original cp system G has length 1.

Combining these observations with the proof of Theorem 4.2 we have the following final result.

Corollary 4.3. For each cp system G there exists an equivalent cp system $G' = (G_1 , G_2 , R)$ with $G_1 = (\Sigma_1 , P_1 , S_1 , \Delta)$ and $G_2 = (\Sigma_2 , P_2 , S_2)$ satisfying

(i) $\Sigma_1 = \Delta \cup \{S_1\}$,

(ii) $P_1 \subseteq \{S_1 \to aS_1 \mid a \in \Delta\} \cup \{S_1 \to a \mid a \in \Delta\}$,

(iii) for each $a \in \Delta$ and each $\pi_2 \in P_2$, $(S_1 \to aS_1 , \pi_2) \in R$ if and only if $(S_1 \to a , \pi_2) \in R$, and

(iv) $P_2 \subseteq \{Z \to XY \mid Z \in \Sigma_2 \text{ and } X, Y \in \Sigma_2 \cup \{\Lambda\}\}$. ∎

DISCUSSION.

From the technical point of view our construction can be seen as follows.

First we translate an arbitrary cp system into a simple cp system, i.e., a cp system with an "irrelevant" first component. Hence, in the terminology of [R2], we have obtained a cp system equivalent to a one coordinate cts system. (It is argued in [R2] that one coordinate cts systems correspond to grammars - hence we translate, using the standard triplet construction, cp systems into "grammars").

Then within the framework of simple cp systems we apply standard techniques to remove "chain-rewrites" and "Λ-rewrites".

To the so obtained cp systems we apply the "regular set construction" where taking right quotients corresponds to erasing a letter (a "pop" instruction in the terminology of push-downs).

As a matter of fact the last step is the only "involved" step used in our proof; the other two constructions are variations on rather standard techniques in formal language theory.

The same idea of "regular set construction" - the representation of a regular set by a single symbol - is already present in [G].

Goldstine's paper presents a general theory of automata which deals with "machines" more general than push-downs.
As an illustration of technical notions presented in his framework he provides a construction which for an arbitrary push-down automaton yields an equivalent real-time push-down automaton. *However* this construction goes through a number of intermediate steps where machines are obtained that are not push-down automata. So in this sense one cannot translate the techniques used by Goldstine into our framework.

Moreover in the construction form [G] another (apart from the one pointed out above) involved step is used - in this sense our construction seems to be simpler.

We hope that the proof we have presented illustrates that usefulness of the formalism of cp systems for dealing with push-down automata.

ACKNOWLEDGEMENTS.

This research was supported by NSF grant number MCS 83-05245.

REFERENCES.

[G] Goldstine, J., "Formal languages and their relation to automata: what Hopcroft and Ullman didn't tell us", in R. Book, ed., *Formal language theory. Perspectives and open problems*. Academic Press, New York, 1980.

[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.

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

[R1] Rozenberg, G., "Selective substitution grammars (towards a framework for rewriting systems), Part I: Definitions and Examples," *Elektron, Informationsverarbeit. Kybernetik*, v. 13, 455-463, 1977.

[R2] Rozenberg, G., "On coordinated selective substitutions: Towards a unified theory of grammars and machines," *Theoretical Computer Science*, to appear.

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

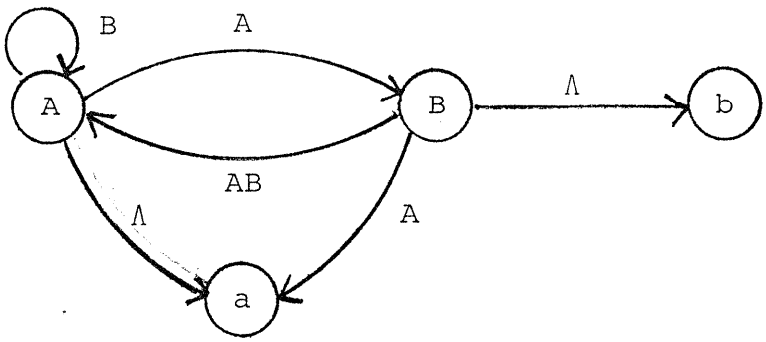


Figure 1

a computation ρ in G	parses of the external snapshots of ρ	the corresponding computation μ in G'
(S, A)	$(\varepsilon; A)$	(S_1 , S_2)
(S, AB)		
(aS, AA)	$(a; AA)$	(aS_1 , Q_1)
(aS, AAB)		
$(aS, AAABA)$		
$(aaS, AAAB)$	$(a,a; A,AAB)$	(a^2S_1 , Q_2Q_1)
$(aaS, AAAABA)$		
$(aaaS, AAAAB)$	$(...; A,AA,AB)$	$(a^3S_1 , Q_2Q_1Q_1)$
$(a^3bS, AAAA)$	$(...; A,AA,A)$	$(a^3bS_1 , Q_2Q_1Q_1)$
(a^3baS, AAA)	$(...; A,AA)$	(a^3baS_1 , Q_2Q_1)
(a^3ba^2S, AA)	$(...; A,A)$	$(a^3ba^2S_1 , Q_2Q_2)$
(a^3ba^3S, A)	$(...; A)$	$(a^3ba^3S_1 , Q_2)$
(a^3ba^3S, AB)		
(a^3ba^3bS, A)	$(...; A)$	$(a^3ba^3bS_1 , Q_2)$
(a^3ba^3ba, Λ)	$(...; \varepsilon)$	(a^3ba^3ba , Λ)

■

Figure 2.