R. R. PUCELLA

An analysis of Lambek's production machines

AN ANALYSIS OF LAMBEK'S PRODUCTION MACHINES (*)

by R. R. Pucella ([1])

Abstract. – *Lambek's production machines may be used to generate and recognize sentences in a subset of the language described by a production grammar. We determine in this paper the subset of the language of a grammar generated and recognized by such machines.*

Résumé. – *Les machine à productions de Lambek peuvent être utilisées pour produire et reconnaître des phrases faisant partie d'un certain sous-ensemble du language décris par une grammaire de productions. Nous déterminons dans cet article le sous-ensemble du language d'une grammaire qui est produit et reconnu par de telles machines.*

1. INTRODUCTION

The focus of this paper is the mechanical generation and recognition of sentences from a production grammar [4, 8], which are known in mathematics as semi-Thue systems and in linguistics as rewriting systems or generative grammars. The latter, linguistics, is an important area of application for production grammars. They were used to study French and Latin conjugation [5, 6] and kinship terminology in English [7] and other languages [11, 1, 2, 3]. Production grammars were also provided for subsets of English and French [8, 13] and used in a naive approach to syntactic translation [13].

To generate and recognize sentences in languages defined by a production grammar, Lambek combined two pushdown automata into a single machine [9] and gave examples of the execution of the machine on simple sentences taken from a grammar describing a subset of English.

Our previous work [13] indicates that Lambek's production machines generate and recognize a subset of the language of a grammar — in other

words, they do not generate or recognize sentences not in the language. This paper analyzes the machines in order to determine exactly which subsets of the language are generated and recognized. The sublanguage generated is generally a proper subset of the language, which we call the leftmost language. Correspondingly, the sublanguage recognized, also generally a proper subset of the language, may be seen as a dual to the leftmost language.

2. PRODUCTION GRAMMARS

We review in this section the fundamental material needed in the paper. We assume the reader is acquainted with the theory of formal languages, so that only a short overview of the notation is necessary.

A production grammar is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{V}_i, \mathcal{V}_t, \mathcal{P})$ where $\mathcal{V}$ (the vocabulary or alphabet) is a finite set, $\mathcal{V}_i$ and $\mathcal{V}_t$ (the initial and terminal vocabularies) are subsets of $\mathcal{V}$, and $\mathcal{P}$ (the productions) is a finite or at least recursive set of pairs (Γ, Δ) with Γ and Δ strings of elements of $\mathcal{V}$. We usually represent an element (Γ, Δ) of $\mathcal{P}$ as $\Gamma \longrightarrow \Delta$. An element of $\mathcal{V}_t$ is called a *terminal* symbol, while an element of $\mathcal{V} - \mathcal{V}_t$ is called a *nonterminal* symbol. A string of elements of $\mathcal{V}$ will typically be denoted by a greek letter, and individual elements of $\mathcal{V}$ by capital roman letter.

From any production grammar $\mathcal{G} = (\mathcal{V}, \mathcal{V}_i, \mathcal{V}_t, \mathcal{P})$ one obtains the *dual* grammar of $\mathcal{G}$ by taking $\mathcal{G}^{-1} = (\mathcal{V}, \mathcal{V}_t, \mathcal{V}_i, \mathcal{P}^{-1})$ where $\mathcal{P}^{-1}$ is the set of all pairs (Δ, Γ) such that $(\Gamma, \Delta) \in \mathcal{P}$.

A production $\Gamma \longrightarrow \Delta$ is *applicable* to a string σ of element of $\mathcal{V}$ if σ is of the form $\sigma_1 \Gamma \sigma_2$. The *application* of $\Gamma \longrightarrow \Delta$ to σ is the string $\sigma_1 \Delta \sigma_2$. A production $\Gamma \longrightarrow \Delta$ is *leftmost applicable* to a string σ if σ is of the form $\sigma_1 \Gamma \sigma_2$ and for any production $\Gamma' \longrightarrow \Delta'$, if σ is of the form $\gamma_1 \Gamma' \gamma_2$, then $|\Gamma| \leq |\Gamma'|$ and $|\sigma_1| \leq |\gamma_1|$.

We define the leftmost reduction relation on strings of elements of $\mathcal{V}$ as follows: let $\sigma_1 \longrightarrow \sigma_2$ if a production of $\mathcal{G}$ is leftmost applicable to σ_1 and σ_2 is the application of the production to σ_1. A *sentence* is a string of terminal symbols in $\mathcal{V}_t$. The *leftmost language* of a grammar $\mathcal{G}$ is the set of all sentences that can be derived via $\overset{*}{\longrightarrow}$ starting from symbols in $\mathcal{V}_i$. If we define a reduction relation using the notion of applicability instead of leftmost applicability, the set of sentences that can be derived is called the *language* of the grammar. For emphasis, we sometimes refer to the language as the *full language* of the grammar. It is clear that the leftmost language of a grammar is a subset of the full language. The following grammar shows

that the inclusion may be proper:

$$S \longrightarrow ABC$$
$$AB \longrightarrow x$$
$$BC \longrightarrow y$$
$$C \longrightarrow z$$
$$A \longrightarrow w$$

The full language of this grammar is $\{xz, wy\}$, and the leftmost language is $\{wy\}$.

We assume in this paper that all grammars under consideration are *well-formed*, in the sense that all reduction sequences ultimately lead to sentences — string of terminal symbols. This among other things implies that there is at least one production for each initial symbol in $\mathcal{V}_i$. We shall also assume, as it is usually done, that there is no empty production and that no terminal appears on the left side of a production.

Let us now present three transformations one needs to perform on a grammar $\mathcal{G}$ to make it suitable for treatment by the machine we introduce in the next section. A requirement of the transformations is that they preserve the leftmost language of the untransformed grammar.

The first transformation takes a grammar $\mathcal{G}$ with initial vocabulary $\mathcal{V}_i$ and produces a new grammar $\mathcal{G}'$ with a unique initial symbol, say S (this symbol must be a new symbol not originally in $\mathcal{V}$). The transformation simply consists of adding a new production for every initial symbol of $\mathcal{G}$. For example, if $\mathcal{V}_i = \{A,B,C\}$, we add the productions

$$S \longrightarrow A$$
$$S \longrightarrow B$$
$$S \longrightarrow C$$

and let the new initial vocabulary be $\mathcal{V}_i = \{S\}$. It is clear that the leftmost language of $\mathcal{G}$ is preserved by this transformation. The second transformation is the process of normalization. A production $\Gamma \longrightarrow \Delta$ is called *normal* if both Γ and Δ have length 1 or 2. A normal grammar is a grammar in which every production is normal. Normalization produces a normal grammar from a grammar, while preserving the leftmost language of the grammar. The transformation consists in iterating the following production replacements (the symbol N is always taken to be a new symbol not in $\mathcal{V}$ at every

production replacement):

$$\Gamma \longrightarrow AB\Delta \Rightarrow \Gamma \longrightarrow N\Delta$$
$$N \longrightarrow AB$$
$$AB\Gamma \longrightarrow \Delta \Rightarrow N\Gamma \longrightarrow \Delta$$
$$AB \longrightarrow N$$

For the last production replacement, the same symbol N must be used for all productions with the same left side, e.g. $AB\Gamma$. To see why the leftmost language of the original grammar is preserved, consider the two cases that arise: if $\Gamma \longrightarrow AB\Delta$ is leftmost applicable, so is $\Gamma \longrightarrow N\Delta$, and once applied, by leftmost reduction and since no other production may involve the newly introduced symbol N, the next production to apply must be $N \longrightarrow AB$; similarly, if $AB\Gamma \longrightarrow \Delta$ is leftmost applicable, so is $AB \longrightarrow N$, and once applied, the leftmost applicable productions include $N\Gamma \longrightarrow \Delta$ (again, since the newly introduced symbol N cannot appear in other productions not of the form $N\Gamma \longrightarrow ...$).

The next transformation we consider isolates the generation of terminal symbols into their own production. Assuming the grammar under consideration is normal, iterate the following productions replacement (the symbols N,N_1,N_2 are taken to be new symbols not in $\mathcal{V}$ for every replacement, and the symbols t,t_1,t_2 are taken to be terminal symbols):

$$\Gamma \longrightarrow At \Rightarrow \Gamma \longrightarrow AN$$
$$N \longrightarrow t$$
$$\Gamma \longrightarrow tA \Rightarrow \Gamma \longrightarrow NA$$
$$N \longrightarrow t$$
$$\Gamma \longrightarrow t_1 t_2 \Rightarrow \Gamma \longrightarrow N_1 N_2$$
$$N_1 \longrightarrow t_1$$
$$N_2 \longrightarrow t_2$$

It is clear that this transformation preserves the leftmost language of the original grammar.

Please note that the first transformation applied to a grammar $\mathcal{G}$ has the same effect as the last transformation when one considers the dual grammar $\mathcal{G}^{-1}$, namely to isolate the production of the (then terminal) symbol S.

The last transformation has the following interesting (and useful) consequence:

LEMMA 2.1: *Given $\mathcal{G}$ a grammar to which the last transformation above has been applied. If a terminal symbol is produced after leftmost applications*

of productions, then every symbol to the left of that terminal symbol will also be a terminal symbol.

Proof: By the last transformation applied to the given grammar, since a terminal is produced, then the leftmost applicable production must have been of the form N $\longrightarrow$ t with t the produced terminal symbol. Assume that there are nonterminals to the left of that terminal. Since no new nonterminal has been introduced, no terminal may be used on the left of a production, and the grammar is assumed to be well-formed, there must exist a production applicable to nonterminals on the left of the terminal. But this contradicts the fact that the production N $\longrightarrow$ t was leftmost.   □

3. PRODUCTION MACHINES

Lambek describes in [9] a machine that allows us to generate and recognize sentences from a production grammar. A *production machine* [9, 10] corresponds roughly to a combinaison of two pushdown automata. It consists of three potentially infinite tapes subdivided into squares. The middle tape is the input/ouput tape, the top and bottom tapes are storage tapes. Only one square in each taped is *scanned* at any given point in time. The two storage tapes can move in either direction, whereas the input/output tape moves only from right to left. The tapes are positioned so that all three scanned squares are aligned (see Figure 1).
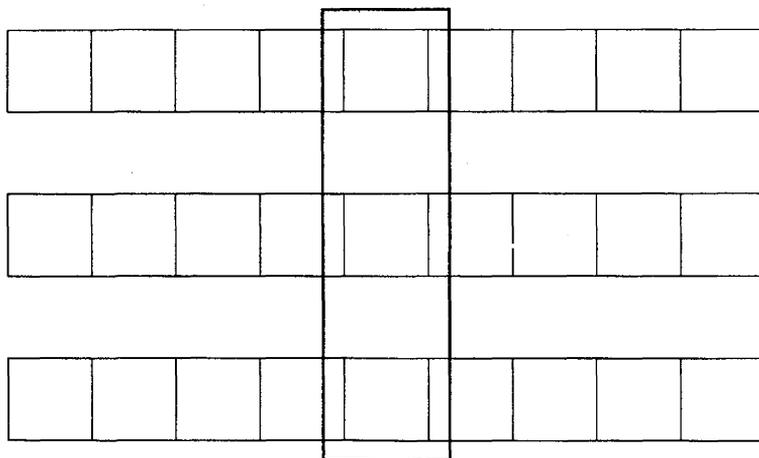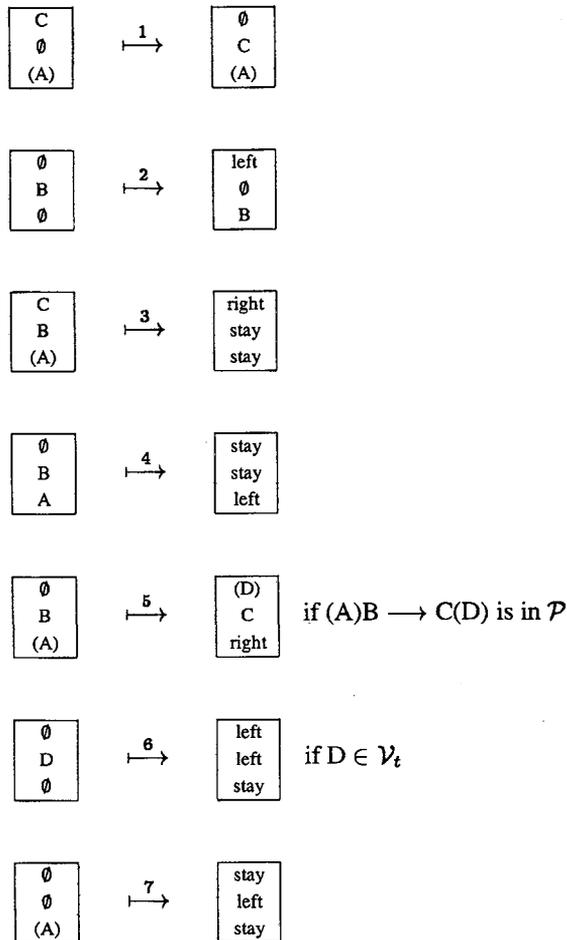

Figure 1. – Production machine

Seven moves are defined for production machines, parametrized by a given grammar $\mathcal{G}$. The moves involve the scanned squares of the tapes:

$$
\begin{array}{c} \text{C} \\ \emptyset \\ \text{(A)} \end{array} \overset{1}{\longmapsto} \begin{array}{c} \emptyset \\ \text{C} \\ \text{(A)} \end{array}
$$

$$
\begin{array}{c} \emptyset \\ \text{B} \\ \emptyset \end{array} \overset{2}{\longmapsto} \begin{array}{c} \text{left} \\ \emptyset \\ \text{B} \end{array}
$$

$$
\begin{array}{c} \text{C} \\ \text{B} \\ \text{(A)} \end{array} \overset{3}{\longmapsto} \begin{array}{c} \text{right} \\ \text{stay} \\ \text{stay} \end{array}
$$

$$
\begin{array}{c} \emptyset \\ \text{B} \\ \text{A} \end{array} \overset{4}{\longmapsto} \begin{array}{c} \text{stay} \\ \text{stay} \\ \text{left} \end{array}
$$

$$
\begin{array}{c} \emptyset \\ \text{B} \\ \text{(A)} \end{array} \overset{5}{\longmapsto} \begin{array}{c} \text{(D)} \\ \text{C} \\ \text{right} \end{array} \quad \text{if (A)B} \longrightarrow \text{C(D) is in } \mathcal{P}
$$

$$
\begin{array}{c} \emptyset \\ \text{D} \\ \emptyset \end{array} \overset{6}{\longmapsto} \begin{array}{c} \text{left} \\ \text{left} \\ \text{stay} \end{array} \quad \text{if D} \in \mathcal{V}_t
$$

$$
\begin{array}{c} \emptyset \\ \emptyset \\ \text{(A)} \end{array} \overset{7}{\longmapsto} \begin{array}{c} \text{stay} \\ \text{left} \\ \text{stay} \end{array}
$$

The $(\cdot)$ notation indicates that the scanned square may or may not be empty, and $\emptyset$ represents an empty square. A mention of "left", "right", "stay" means that the corresponding tape should be moved left, right or stay in the current position. We use the expression "move $\overset{5}{\longmapsto}$ via production P" to explicitely state which production is involved in the move.

The machine may be used either to generate sentences from the grammar or to recognize sentences in the grammar. Those two activities involve different subsets of the general moves presented above, and different starting

and ending states for the machine. We will therefore speak of production machines as though there were two types of machines: the generative machine $\mathcal{M}_g(\mathcal{G})$ corresponding to a grammar $\mathcal{G}$ and the recognitive machine $\mathcal{M}_r(\mathcal{G})$ corresponding to a grammar $\mathcal{G}$.

The generative machine of $\mathcal{G}$ has the following initial and terminal states:

$$\text{Initial:} \quad \begin{bmatrix} S \\ \emptyset \\ \emptyset \end{bmatrix} \qquad \text{Terminal:} \quad <\text{sentence}> \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

The machine is defined with respect to the grammar $\mathcal{G}$, and the moves that should be attempted in order are the following: 5, 6, 1, 2, 3, 4. We say that a sentence σ is *producible* by $\mathcal{M}_g(\mathcal{G})$ if the machine starts in the initial state and ends up in a state

$$\sigma \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

The recognitive machine of $\mathcal{G}$ has the following initial and terminal states:

$$\text{Initial:} \quad \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} <\text{sentence}> \qquad \text{Terminal:} \quad \begin{bmatrix} \emptyset \\ \emptyset \\ S \end{bmatrix}$$

The machine is defined with respect to the dual grammar $\mathcal{G}^{-1}$ and the moves that should be attempted in order are the following: 5, 7, 1, 2, 3, 4. We say that a sentence σ is *recognizable* by $\mathcal{M}_r(\mathcal{G})$ if it ends in the terminal state after starting in a state

$$\begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \sigma$$

We refer the reader to [9] for sample executions of the machine to generate and recognize sentences in a simple grammar for the English language.

One look at the moves of a production machine shows that the machine is fundamentally nondeterministic. Indeed, move $\overset{5}{\longmapsto}$ is used in a nondeterministic way if more than one production with a left side of (A)B is present in the grammar. For a generative production machine, this allows the machine to generate different sentences. For a recognitive machine, this introduces a complexity: possibly only one nondeterministic choice of production to apply next leads to the terminating state of the machine,

as some examples in [9] show. Hence, a recognitive production machine must consider concurrently all the possible applications of move 5 and terminate when one leads to the terminating state. A sentence σ is therefore recognizable if one of the concurrent consideration of an application of move $\overset{5}{\longmapsto}$ of the recognitive production machine reaches the terminal state.

4. GENERATION

We analyze in this section the generative production machine $\mathcal{M}_g(\mathcal{G})$ of a given grammar $\mathcal{G}$. We show that the language generated by $\mathcal{M}_g(\mathcal{G})$ is exactly the leftmost language of $\mathcal{G}$: a sentence σ is producible by $\mathcal{M}_g(\mathcal{G})$ if and only if σ is in the leftmost language of $\mathcal{G}$. Without loss of generality, we may assume that the grammar $\mathcal{G}$ under consideration is a normal grammar with a unique initial symbol S and with a unique production corresponding to the generation of every terminal symbol. As we saw earlier, any grammar may be transformed into such a grammar defining the same leftmost language.

The idea underlying the proof is straightforward. Given a grammar $\mathcal{G}$ and a generative production machine $\mathcal{M}_g(\mathcal{G})$, we show that the graph corresponding to the leftmost reduction relation is isomorphic to a graph corresponding to the moves of the machines. Therefore, a string in the leftmost language of $\mathcal{G}$ obtained by leftmost reductions may be generated by the machine following the moves specified by the isomorphism, and vice-versa.

The main operational tool we use is a transition graph. Given a set D, a subset I of D and a non-transitive relation $<$ over D, define a family of subset of D by the equations

$$S_0 = I$$
$$S_{n+1} = \{b : a < b \text{ for some } a \in S_n\}$$

The transition graph of $<$ generated by I is the graph with nodes in $\cup_{n=0}^{\infty} S_n$ and an edge between $a, b \in \cup_{n=0}^{\infty} S_n$ if and only if $a < b$. Define a layer of the transition graph T over $<$ generated by I to be the set of all element of the graph at a certain distance of an element of the initial subset, $lay_i(T) = \{a : \exists a_0, \ldots, a_{i-1} \in T \text{ such that } a_0 \in I \text{ and } a_0 < \cdots < a_{i-1} < a\}$. If T is defined by the above equations for S_0 and S_{n+1}, it is not hard to see that $lay_i(T) = S_i$.

For a given grammar $\mathcal{G}$ with initial symbol S, the leftmost reduction relation over strings in $\mathcal{V}^*$ lead to the transition graph of $\longrightarrow$ generated by $\{S\}$, which we will denote by $\mathcal{L}$. It is this transition graph that we will

show is isomorphic to a transition graph derived from the moves of the generative machine.

Taking the $\longmapsto$ relation over the states of the machine also leads to a transition graph, but it is easily seen to be much larger than the transition graph $\mathcal{L}$, since for every production application (which corresponds to a move $\overset{5}{\longmapsto}$), there are other administrative moves that the machine needs to perform. However, the key consideraton is the following: all the moves the machine makes are deterministic, except for move $\overset{5}{\longmapsto}$, since there might be many applicable productions at that point. If the grammar is well-formed, the following lemma is easily seen to hold:

LEMMA 4.1 (Determinacy): *Given a state s of $\mathcal{M}_g(\mathcal{G})$ which allows a move* $\overset{5}{\longmapsto}$ *to a state s'. There exists unique states and moves*

$$s \overset{5}{\longmapsto} s' \overset{m_1}{\longmapsto} s_1 \overset{m_2}{\longmapsto} \cdots \overset{m_k}{\longmapsto} s_k$$

such that $m_1, \ldots, m_k \neq 5$ and state s_k allows either no moves or a move $\overset{5}{\longmapsto}$.

We define a reduction relation $\overset{c}{\longmapsto}$ between states of $\mathcal{M}_g(\mathcal{G})$ that allow either a move $\overset{5}{\longmapsto}$ or no move at all: in the statement of the above lemma, if $s \overset{5}{\longmapsto} s'$ via production P, we say that $s \overset{c}{\longmapsto} s_k$ via production P. This is well-defined (by the above lemma) and can be seen as a collapse of the $\longmapsto$ transitions. The following result is a reformulation of lemma 4.1:

COROLLARY 4.2: *Given s a state of $\mathcal{M}_g(\mathcal{G})$. If $s \overset{c}{\longmapsto} s_1$ via production P and $s \overset{c}{\longmapsto} s_2$ via production P, then $s_1 = s_2$.*

Let $\mathcal{T}$ be the transition graph of $\overset{c}{\longmapsto}$ generated by the machine state $\begin{array}{|c|} \hline \emptyset \\ s \\ \emptyset \\ \hline \end{array}$

We now show that $\mathcal{L}$ is isomorphic to $\mathcal{T}$. Let us first define a mapping between strings of elements of $\mathcal{V}$ and states of $\mathcal{M}_g(\mathcal{G})$. This function will be the isomorphism we are looking for.

DEFINITION 4.3: *Given a grammar $\mathcal{G} = (\mathcal{V}, \mathcal{V}_i, \mathcal{V}_t, \mathcal{P})$, and σ a string of elements of $\mathcal{V}$. Suppose σ is of the form $t_1 \ldots t_p n_1 \ldots n_q P_1 P_2 m_1 \ldots m_r$, where $t_1, \ldots, t_p$ are prefixing terminal symbols, $n_1, \ldots, n_q, P_1, P_2, m_1, \ldots, m_r$ are nonterminal symbols and the leftmost applicable production of $\mathcal{P}$ to σ, if any, is of the form $P_1 P_2 \longrightarrow \ldots$ (P_1 might be empty). Define the function F by*

$$F(\sigma) = \begin{array}{c c} & \begin{array}{|c|} \hline \emptyset \\ t_1 \cdots t_p \quad P_2 \\ n_1 \cdots n_q \quad P_1 \\ \hline \end{array} \end{array} \; m_1 \cdots m_r$$

or (if no production is applicable to σ)

$$F(\sigma) = \quad \sigma \begin{array}{|c|} \hline \emptyset \\ \emptyset \\ \emptyset \\ \hline \end{array}$$

The symbols P_1 (if any) and P_2 are said to be in application position.

LEMMA 4.4: *F is injective.*

Proof: Given $\sigma, \sigma' \in \mathcal{L}$. Assume $F(\sigma) = F(\sigma')$. Then $\sigma = t_1 \ldots t_p \sigma_1$ and $\sigma' = t_1 \ldots t_p \sigma'_1$, with σ_1, σ'_1 strings of nonterminals. If no symbols are in application position, then by the definition of F both σ, σ' are strings of terminals, and by the above $\sigma = \sigma'$. If P_1 and P_2 are in application position (P_1 might be empty), then $\sigma_1 = \sigma_2 P_1 P_2 \sigma_3$ and $\sigma'_1 = \sigma'_2 P_1 P_2 \sigma'_3$ and again by the definition of F, $\sigma_2 = \sigma'_2$ and $\sigma_3 = \sigma'_3$. Thus $\sigma = \sigma'$ and F is injective.

LEMMA 4.5: *Given $\sigma, \sigma' \in \mathcal{L}$, then $\sigma \longrightarrow \sigma'$ implies $F(\sigma) \overset{c}{\longmapsto} F(\sigma')$.*

Proof: Given $\sigma, \sigma' \in \mathcal{L}$. Assume σ is of the form $\tau A_1 \cdots A_n$. Four cases arise, depending on the form of the production applicable to σ (there must be one).

1. $A_1 \longrightarrow t$ with t a terminal symbol, and σ' is of the form

$$\tau t A_2 \cdots A_n$$

2. $A_1 A_2 \longrightarrow t$ with t a terminal symbol, and σ' is of the form

$$\tau t A_3 \cdots A_n$$

3. $A_k \longrightarrow \Gamma$ for some k, and σ' is of the form

$$\tau A_1 \cdots A_{k-1} \Gamma A_{k+1} \cdots A_n$$

4. $A_k A_{k+1} \longrightarrow \Gamma$ for some k, and σ' is of the form

$$\tau A_1 \cdots A_{k-1} \Gamma A_{k+2} \cdots A_n$$

It is straightforward to show that in all those cases, $F(\sigma) \overset{c}{\longmapsto} F(\sigma')$. $\qquad\square$

LEMMA 4.6: *Given $\sigma, \sigma' \in \mathcal{L}$, then $F(\sigma) \overset{c}{\longmapsto} F(\sigma')$ implies $\sigma \longrightarrow \sigma'$.*

Proof: Assume $F(\sigma) \overset{c}{\longmapsto} F(\sigma')$ via production $\Gamma \longrightarrow \Delta$. By definition of F, $\Gamma \longrightarrow \Delta$ is leftmost applicable to σ. Let $\sigma \longrightarrow \sigma''$ via production

$\Gamma \longrightarrow \Delta$. By lemma 4.5, $F(\sigma) \stackrel{c}{\longmapsto} F(\sigma'')$ via production $\Gamma \longrightarrow \Delta$. By corollary 4.2, $F(\sigma') = F(\sigma'')$, and by lemma 4.4, $\sigma' = \sigma''$ and thus $\sigma \longrightarrow \sigma'$. ∎

LEMMA 4.7: $F(\mathcal{L}) = \mathcal{T}$.

Proof: We show by induction on i that $\forall i\ F(lay_i(\mathcal{L})) = lay_i(\mathcal{T})$, which clearly implies the statement of the lemma.

The base case of the induction is trivial, since $F(S) = \begin{bmatrix} \emptyset \\ s \\ \emptyset \end{bmatrix}$

For the induction step, we first show $F(lay_{i+1}(\mathcal{L})) \subset lay_{i+1}(\mathcal{T})$. Given $\sigma \in lay_{i+1}(\mathcal{L})$. Thus, there exists a $\sigma' \in lay_i(\mathcal{L})$ such that $\sigma' \longrightarrow \sigma$. By the induction hypothesis, $F(\sigma') \subset lay_i(\mathcal{T})$. By lemma 4.5, $F(\sigma') \stackrel{c}{\longmapsto} F(\sigma)$, and by definition of transition graph $\mathcal{T}$, $F(\sigma) \in lay_{i+1}(\mathcal{T})$.

We next show $lay_{i+1}(\mathcal{T}) \subset F(lay_{i+1}(\mathcal{L}))$. Let $s \in lay_{i+1}(\mathcal{T})$. Thus there exists a $s' \in lay_i(\mathcal{T})$ with $s' \stackrel{c}{\longmapsto} s$ via production $\Gamma \longrightarrow \Delta$. By the induction hypothesis, there exists a $\sigma' \in lay_i(\mathcal{L})$ such that $F(\sigma') = s'$. Let σ be the application of $\Gamma \longrightarrow \Delta$ to σ'. By lemma 4.5, $F(\sigma') \stackrel{c}{\longmapsto} F(\sigma)$, and thus $s' \stackrel{c}{\longmapsto} F(\sigma)$. By corollary 4.2, $F(\sigma) = s$ and thus $s \in F(lay_{i+1}(\mathcal{L}))$. This completes the induction and the proof. ∎

LEMMA 4.8: *F is an isomorphism of graphs from $\mathcal{L}$ to $\mathcal{T}$.*

Proof: By lemmas 4.4 and 4.7, F is a bijective function from $\mathcal{L}$ to $\mathcal{T}$. By lemmas 4.5 and 4.6, F is a transition graph isomorphism. ∎

PROPOSITION 4.9: *Given a grammar $\mathcal{G}$, a sentence σ is producible by $\mathcal{M}_g(\mathcal{G})$ if and only if σ is in the leftmost language of $\mathcal{G}$.*

Proof: ($\Rightarrow$) Given $\sigma = t_1 \cdots t_n$ a string in the leftmost language of $\mathcal{G}$. Thus there exists a chain in $\mathcal{L}$ from S, the initial symbol of $\mathcal{G}$, to σ representing the leftmost reductions derivation of σ. By the isomorphism of lemma 4.8, there exists a chain in $\mathcal{T}$

$$\begin{bmatrix} \emptyset \\ s \\ \emptyset \end{bmatrix} \stackrel{c}{\longmapsto} \cdots \stackrel{c}{\longmapsto} \sigma \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

Since

$$\begin{bmatrix} s \\ \emptyset \\ \emptyset \end{bmatrix} \stackrel{1}{\longmapsto} \begin{bmatrix} \emptyset \\ s \\ \emptyset \end{bmatrix}$$

and extending (uniquely, by lemma 4.1) the $\xmapsto{c}$ transitions, we get a sequence of machine moves

$$\begin{bmatrix} S \\ \emptyset \\ \emptyset \end{bmatrix} \xmapsto{1} \cdots \xmapsto{6} \sigma \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

and thus σ is producible by $\mathcal{M}_g(\mathcal{G})$.

($\Longleftarrow$) Given $\sigma = t_1 \cdots t_n$ a string producible by $\mathcal{M}_g(G)$. There exists machine moves

$$\begin{bmatrix} S \\ \emptyset \\ \emptyset \end{bmatrix} \xmapsto{1} \begin{bmatrix} \emptyset \\ S \\ \emptyset \end{bmatrix} \xmapsto{5} \cdots \xmapsto{6} \sigma \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix}$$

Starting from $\begin{bmatrix} \emptyset \\ S \\ \emptyset \end{bmatrix}$ and collapsing the $\longmapsto$ transitions into $\xmapsto{c}$ transitions, we get a chain in $\mathcal{T}$. By the isomorphism of lemma 4.8, we get a chain in $\mathcal{L}$

$$S \longrightarrow \cdots \longrightarrow \sigma$$

and thus σ is in the leftmost language of $\mathcal{G}$. $\qquad\square$

5. RECOGNITION

Fundamentally, the recognitive machine $\mathcal{M}_r(\mathcal{G})$ is similar to the generative one: it defines essentially the same moves (except that the move producing terminals is replaced by a move that accept the next symbol from the input/output tape), and it uses the dual of the grammar under consideration.

One may again derive an isomorphism in the manner described in the previous section, connecting the moves of the recognitive machine to the leftmost reduction relation defined on the dual of the grammar. One needs to extend the definition of transition graphs to use strings of terminals as the initial set. The extension is fairly trivial, and is left as an exercise.

The language generated by $\mathcal{M}_g(\mathcal{G})$ is the leftmost language of $\mathcal{G}$, the one obtained by allowing only leftmost reductions. Correspondingly, the language recognized by $\mathcal{M}_r(\mathcal{G})$ is a dual to the leftmost language, characterized as those sentences that can be recognized via leftmost reductions in the dual grammar.

It is clear that the recognized language is a subset of the full language of the grammar. The following grammar shows that the recognized language is in general a proper subset of the full language, and need not be equal to the generated language:

$$S \longrightarrow AG$$
$$F \longrightarrow C$$
$$G \longrightarrow BC$$
$$E \longrightarrow AB$$
$$BC \longrightarrow z$$
$$A \longrightarrow x$$

The full language generated by this grammar is $\{xz\}$. The leftmost language of this grammar is also $\{xz\}$. However, trying to recognize the string xz via leftmost reductions in the dual grammar leads to a unique derivation

$$xz \longrightarrow Az \longrightarrow ABC \longrightarrow EC \longrightarrow EF$$

and thus the string is not recognized by the machine.

6. CONCLUSION

We provide in this paper an analysis of the production machines described by Lambek in [9, 10]. We determine the subset of the full language of a grammar that is both generated and recognized by the machines. The generated language corresponds to the subset of the full language one obtains by applying leftmost reductions, and is in general a proper subset of the full language. Conversely, the recognized language corresponds to the subset of the full language one obtains by applying leftmost reductions in the *dual* grammar, and is also in general a proper subset of the full language. Moreover, the generated and recognized language need not agree.

The generative version of production machines can in fact be reguarded as implementing a generalized version of a Markov algorithm [12, 14]. A Markov algorithm on a production grammar $\mathcal{G}$ consists of repeatedly applying a leftmost applicable production to a string, and if more than one production is leftmost applicable, the first production (given an ordering of the productions) is applied. As such, the algorithm is fully deterministic. In contrast, while a generative production machine also applies leftmost applicable productions, the choice of which production to apply if more than one is applicable is non-deterministic.

Let us mention a possible extension of the description of the production machines that would allow for the generation and recognition of the full language. Recognition is the easiest to extend: when the machine verifies all the possible choices of production in parallel when a move $\overset{5}{\longmapsto}$ is applicable, one adds the parallel choice of *not* applying any production, and passing on to the next possible move of the machine. One can extend generation in the same way, by adding a nondeterministic choice of not applying a move $\overset{5}{\longmapsto}$ when it is possible to do so. This extension has a caveat: generation may fail to produce a sentence.

An important class of grammars does not satisfy the criteria set forth for generation and recognition via production grammars: translation grammars, which take strings of initial symbols as initial states. For example, the initial symbols could be words of English, and terminal symbols words in French, and the grammar would translate English into French. The production machines presented in this paper can be modified easily to handle such grammars.

REFERENCES

1. M. BHARGAVA and J. LAMBEK, A production grammar for Hindi kinship terminology, *Theoretical Linguistics*, 1983, *10*, pp. 227-245.

2. M. BHARGAVA and J. LAMBEK, A production grammar for Sanskrit kinship terminology, *Theoretical Linguistics*, 1992, *18*, pp. 45-60.

3. M. BHARGAVA and J. LAMBEK, Lounsbury's analysis of Trobriand kinship terminology, *Theoretical Linguistics*, 1995, *21*, pp. 241-253.

4. J. E. HOPCROFT and J. D. ULLMAN, *Formal languages and their relation to automata*, Addison Wesley, Reading Mass., 1969.

5. J. LAMBEK, A mathematician looks at French conjugation, *Theoretical Linguistics*, 1975, *2*, pp. 203-214.

6. J. LAMBEK, A mathematician looks at Latin conjugation, *Theoretical Linguistics*, 1979, *6*, pp. 221-234.

7. J. LAMBEK, A production grammar for English kinship terminology, *Theoretical Linguistics*, 1986, *13*, pp. 19-36.

8. J. LAMBEK, Grammar as mathematics, *Canadian Mathematical Bulletin*, 1989, *32*, (3), pp. 257-273.

9. J. LAMBEK, Production grammars revisited, *Linguistic Analysis*, 1993, *23*, pp. 1-21.

10. J. LAMBEK, Programs, grammars and arguments: a personal view of some connections between computation, language and logic. In *Proceedings of the Annual IEEE Symposium on Logic in Computer Science*, 1993.
11. J. LAMBEK and M. LAMBEK, The kinship terminology of Malagasy speakers in Mayotte, *Anthropological Linguistics*, 1981, *23*, pp. 154-182.
12. A. A. MARKOV, Theory of algorithms, *Translations of the American Mathematical Society Series 2*, 1960, *15*.
13. R. R. PUCELLA, Production grammars, machines and syntactic translation. Technical Report SOCS-96.6, McGill University, November 1996.
14. A. SALOMAA, Formal languages and power series. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, chapter 3, pages 103–132. The MIT Press / Elsevier, 1990.