WEB GRAMMARS

John L. Pfaltz
Azriel Rosenfeld

Computer Science Center
University of Maryland
College Park, Maryland

Classes of "phrase-structure grammars" are defined whose "languages" consist, not of strings of symbols, but of directed graphs with symbols at their vertices ("webs"). Examples of such "web grammars" are given, having languages consisting of trees, of two-terminal series-parallel networks, and of "triangular" networks. It is shown that if the graphs permitted in a "context-sensitive" web grammar are required to be acyclic, and the parsing rules are assumed to be graph homomorphisms, then any sub-graph which is parsed by a rule must be "convex", and any rule is a composite of rules each of which parses a subgraph having just two points.

Foreword

Since the early 1960's, considerable effort has been devoted to the problem of developing formal "picture languages" whose "sentences" are pictures of various types. This work (see^ for a recent survey) can be thought of as generalizing the concepts and methods of mathematical linguistics from conventional languages, in which sentences are strings formed by concatenating symbols, to "languages" in which "sentences" are formed by combining symbols in more general ways. The present paper describes a method of defining "languages" whose "sentences" are directed graphs with symbols at their vertices. The writers are indebted to R. A. Kirsch and D. E. Knuth for suggesting this area of research. The support of the Information Systems Branch, Office of Naval Research, under Contract Nonr-5144(00), is gratefully acknowledged.

Graphs and Webs

A directed graph G is an ordered pair (P_G,E_G), where

> P_G is a finite set, whose elements (denoted by lower-case letters) are called the points of G
>
> E_G is a finite set of ordered pairs of elements of P_G, which are called the edges of G.

We say that there is an edge from p to q if (p,q) is in E_G; we say that there is a loop at p if (p,p) is in E_G. A string $(a_1,\ldots,a_n)$ can be regarded as a directed graph, in which the set of points is $\{a_1,\ldots,a_n\}$, and the set of edges is $\{(a_1,a_2),(a_2,a_3),\ldots,(a_{n-1},a_n)\}$.

$H = (P_H,E_H)$ is called a subgraph of $G = (P_G,E_G)$ if

> P_H is a subset of P_G
>
> E_H consists of just those pairs in E_G whose terms are both in H

Thus any subset P of P_G defines a unique subgraph, which we call the subgraph on P.

Two points p,q of the graph G are said to be connected if there exists a sequence of points $p = p_0,p_1,\ldots,p_n = q$ such that either (p_{i-1},p_i) or (p_i,p_{i-1}) is in E_G, $1 \le i \le n$. A subgraph is called connected if every pair of its points is connected. The set of points of G which are connected to any given point of G is called a connected component of G. Readily, the connected subgraphs of a string are just its substrings; these are the subgraphs defined on sets of consecutive points of the string.

By a path of length n (where $n \ge 0$) from p to q in the graph G is meant a sequence of points $p = p_0,p_1,\ldots,p_n = q$ such that $p_{i-1} \ne p_i$ and (p_{i-1},p_i) is in E_G, $1 \le i \le n$. [Note that, in contrast to the definition of "connected", here the edges must all go in the same direction; the requirement that consecutive points be distinct is needed in order to be able to speak meaningfully about the length of the path. A sequence in which the edges need not be in the same direction is sometimes called a walk.] A path of length ≥ 2 from a point to itself is called a cycle; note that by this definition, a loop is not a cycle. A graph in which there exist no cycles is called acyclic. Readily, any string is acyclic.

A point of a directed graph is called a least point, if there is a path from it to every other point; a greatest point, if there is a path from every other point to it. A point is called minimal, if there is no path from any other point to it; maximal, if there is no path from it to any other point. Readily, in an acyclic graph there can be at most one least point and one greatest point, since if there were two, they would be on a cycle.

The terminology "least", "greatest", "minimal" and "maximal" can be justified if we look at graphs from the point of view of relations. In any graph G, the set of edges E_G can be thought of as defining a relation on the set of points P_G (indeed, by definition, a relation R on a set S is just a set of ordered pairs of elements of S). By the transitive closure $\bar{R}$ of the relation R is meant the set of pairs (x,y) of elements of S such that there exist $x = x_0, x_1, \ldots, x_n = y$, $n \geq 0$, for which (x_{i-1}, x_i) is in R, $1 \leq i \leq n$. Note that if n = 0, the last part of the requirement is vacuous -- in other words, if y = x, we automatically have (x,x) in $\bar{R}$, so that $\bar{R}$ is always reflexive; and readily, $\bar{R}$ is transitive, i.e. if (x,y) and (y,z) are in $\bar{R}$, so is (x,z).

Given any graph G, let Π_G be the relation on P_G consisting of the set of pairs (p,q) for which there is a path from p to q. Readily, Π_G is just the transitive closure of E_G. Moreover, let G be acyclic; it is easily seen that this is equivalent to saying that for any p,q in P_G, if there are paths from both p to q and q to p, then we must have p = q. In other words, G is acyclic if and only if the relation Π_G is weakly antisymmetric; and since it is reflexive and transitive, this makes it a partial order relation. [In particular, if G is a string, Π_G is a total order relation, since for every pair of distinct points (a_i, a_j) we evidently have either (a_i, a_j) or (a_j, a_i) in Π_G, depending on whether $i < j$ or $j < i$. This last observation shows that acyclic graphs are a natural generalization of strings.]

Let V be a finite set, which we shall call the vocabulary (or "alphabet"); the elements of V will be called symbols. By a V-labelling of the graph G will be meant a function Λ from P_G into V. By a web W over the vocabulary V will be meant an ordered pair (G,Λ), where G is a graph and Λ is a V-labelling of G. Note that since a string is a special case of a graph, a string of symbols from V is a special case of a web over V.

If G' is a subgraph of G and Λ is a V-labelling of G, then the restriction Λ' of Λ to G' is a V-labelling of G'; we say that $W' = (G',\Lambda')$ is a subweb of $W = (G,\Lambda)$.

Web Grammars

In a conventional phrase-structure "string grammar", rewriting rules of the form $\alpha := \beta$ are used to replace one string by another. Such a rule is completely determined by specifying the pair of strings (α,β); any string $\omega = \varphi\alpha\Psi$ which contains α as a substring can then be immediately rewritten as $\varphi\beta\Psi$. The definition of "rewriting rules" for webs is more complicated; if we want to replace the subweb α of the web ω by another subweb β, it is necessary to specify how to "embed" β in ω in place of α. This can be done in many different ways; for example, one can specify that there be edges between given points of β and any points of $\omega - \alpha$ having various properties (e.g., having given labels, having given numbers of incoming or outgoing edges, being on edges to or from particular points of α in the original ω, etc. etc.) Any such specification of the edges between β and its "host web" will be called an embedding of β.

Formally, we can now define a web rewriting rule as a triple (α,β,E), where α and β are webs, and E is an embedding of β. It is important to emphasize that the definition of an embedding must not depend on the host web ω, since we want to be able to replace α by β in any web containing α as a subweb. Thus any properties of points of the host web which are used in defining the embedding must be well defined for an arbitrary ω. Note that in string grammars, the same (implicit) embedding is used for all rewriting rules $\alpha := \beta$, namely "Put 'edges' from the symbol of the host string which just preceded α to the leftmost symbol of β, and from the rightmost symbol of β to the symbol of the host string which just followed α".

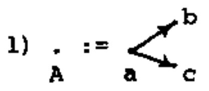
By a web grammar $\mathcal{G}$ we shall mean a triple of the form (V,I,R), where V is a vocabulary; I is a set of "initial" webs; and R is a set of web rewriting rules. As usual, we assume that V consists of two disjoint parts, a "nonterminal vocabulary" V_N and a "terminal vocabulary" V_T. By the language $\mathcal{L}_{\mathcal{G}}$ of the web grammar $\mathcal{G}$ we mean the set of webs which can be derived from the initial webs by repeated application of the rewriting rules, and whose labels all belong to V_T.

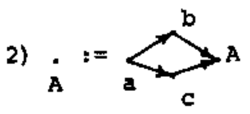
The following is a very simple example of a web grammar:

$V = \{A,a,b,c\}$ (where we have denoted the elements of V_N by capital letters, those of V_T by lower case letters);

I consists of the one-point web with label A;

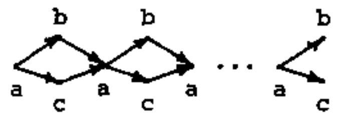
R consists of the following rules:

1) 

2) 

where for both rules, the embedding is $E = \{(p,a) \mid (p,A)$ an edge in the host web$\}$.

It is easily verified that the language of this grammar is just the set of all webs of the form



Some less trivial examples will be given in later sections.

An important special case of a web grammar is that in which the terminal vocabulary consists of only a single symbol. In this case, every point of every web in the language has the same label, so that we can ignore the labels and identify the webs with their underlying graphs. This type of web grammar will be called a graph grammar, and its language will be called a graph language. [In the string case too one can restrict consideration to strings involving only a single symbol; this is still a nontrivial case, since not every set of natural numbers can be the set of lengths of the strings in a one-symbol string language if the sets I and R are required to be finite.]

The general definitions of a web rewriting rule and a web grammar which have been given above are too broad to be of much practical use (just as, analogously, there is little that one can say about string grammars in which arbitrary string rewriting rules are allowed). However, one can define special types of web grammars which are analogous to the various important types of string grammars -- in particular, to context-sensitive and context-free string grammars.

We shall call the web rewriting rule (α,β,E) context-sensitive if there exists a point a of α such that $\alpha - \{a\}$ is a subweb of β -- in other words, the rule "rewrites" only a single point of α -- and all edges between points of the host web and points of $\alpha - \{a\}$ are in E. In particular, the rule (α,β,E) will be called context-free if α has only a single point; note that in this case the definition of a context-sensitive rule is trivially satisfied, since $\alpha - \{a\}$ is empty. It is readily verified that context-sensitive (-free) string rewriting rules, if we regard them as web rewriting rules, satisfy these definitions.

A web grammar will be called context-sensitive (-free) if all its rewriting rules are context-sensitive (-free). The simple example of a web grammar which was given above is context-free, since only one-point webs are rewritten. Other examples of both context-free and context-sensitive web grammars will be given in the next two sections.

It is instructive to compare our definition of a web grammar with the definition of a "plex grammar" given by Feder[2]. We can think of Feder's NAPEs ("n-attaching-point entities") as webs in which one point is labelled with the name of the NAPE, and the others with the identifiers of its attaching points. Feder's "joint lists", which describe how sets of NAPEs are interconnected, correpond to the edges (which need not be directed) internal to the subwebs a and 8 in a web rewriting rule; while his "tie point list" corresponds to the embedding E of B in the host web. (He also makes use of a "tie point list" for a, so that he can allow the rewriting rule to be applied only if the edges between *a* and the host web satisfy given conditions.) Feder's system can be regarded as a generalization of Shaw's (e.g.,1); Shaw deals only with entities having just two attaching points ("head" and "tail"). Thus the web grammars defined in this paper — if their webs are regarded as embedded in the plane — can be thought of as equivalent to a general class of "picture grammars".

<u>Examples of Context-Free Web Languages</u>: <u>Trees and Two-Terminal Series-Parallel Networks</u>

In this section we give context-free graph grammars for two important classes of directed graphs: directed trees with least points, and directed two-terminal series-parallel networks (without multiple edges)•

By a <u>directed tree</u> is meant a connected directed graph whose number of edges is just one less than its number of points. It can be shown that a directed graph with least point is a directed tree if and only if there is a <u>unique</u> path from the least point to each point.

<u>Theorem 3.1</u> The following context-free graph grammar has as its language the set of all directed trees which have least elements:

$V = \{A, a\}$

$I = \{\overset{a}{\cdot}\ ,\ a \longrightarrow A\}$

R consists of the following rules:

(R_1) $\overset{a}{\cdot} := a \longrightarrow A$

$E = \{(p,a) \mid (p,A) \text{ an edge in the host web}\}$

(R_2) $\overset{A}{\cdot} := \begin{array}{c} A \\ \cdot \\ A \\ \cdot \end{array}$

$E = \{(p,A) \mid (p,A) \text{ an edge in the host web}\}$

(R_3) $\overset{A}{\cdot} := \overset{a}{\cdot}$

$E = \{(p,a) \mid (p,A) \text{ an edge in the host web}\}$

<u>Proof</u>; We first verify that the webs generated by this grammar all have least point and all have one fewer edges than points. This is cleat if the initial web was ^{a} , since none of the rules apply to it. Similarly, the initial web a._A has these properties; we proceed by induction on the number of times that the rules are applied to it. Readily, under any application of a rule, the "a" in the initial web remains the least point, since if there was a path from it to every point before the rule was applied, the definition of the embedding insures that there is still a path from it to every point afterwards. Finally, it is easily verified that each rule always adds the same number of edges as it does points, so that the former number always remains one less than the latter.

Conversely, we shall show that any directed tree G with least point can be generated by the grammar. Note first that in any such G, if t is a point such that the path from the least point to t is as long as possible, then t is maximal, so that G always has a maximal point. We shall show that, given any maximal point p, G can be generated in such a way that the last rule used is R_3, applied to p to change its label from A to a. This is clear if G has only one or two points; we proceed by induction on the number of its points.

Let r be a point of G such that the path from the least point to it is as long as possible, and let q be the next-to-last point on this path, so that (q,r) is an edge. Suppose first that there is no $s \neq r$ such that (q,s) is also an edge; then in the subgraph of G from which r has been omitted, q is a maximal element, so that by induction hypothesis this subgraph can be generated by the grammar in such a way that the last rule used is R_3, applied to change the label of q from A to a. But then we can obtain G itself by first applying R_1 to q to "attach" r to it, and then applying R_3 to change the label of r from A to a.

Finally, suppose that there are other points $r = r_1, r_2, \ldots, r_k$, $k \geq 2$, such that (q,r_i) is an edge. Then by induction hypothesis, we can generate the subgraph from which these points have been omitted, and in which q is evidently a maximal element. To obtain G, we then simply replace the final application of R_3 to q by an application of R_1, followed by k - 1 applications of R_2, followed by k applications of R_3.//

It is not difficult to show that if we replace I in the above grammar by $\{\overset{A}{\cdot}\}$, we obtain as language the set of all "forests" of such trees -- in other words, directed graphs each of whose connected components is such a tree.

It is well known that trees have a natural string representation as nests of parentheses. Now the string language consisting of nests of parentheses can be accepted by a pushdown automaton, but not by a finite-state automaton; thus its grammar is context-free, but not finite-state. In contrast, the simple context-free grammar given in Section 2 evidently *is* finite-state. It would be of interest to define and study classes of automata which would be "equivalent" to the various classes of web grammars -- context-free, context-sensitive, etc. -- considered in this paper.

The set of *directed two-terminal series-parallel networks* can be defined as the set of directed graphs having least and greatest points which can be "derived" from the graph $\overset{x\quad y}{\longrightarrow}$ by repeatedly performing the following operations:

a) *Serial composition*: Let G_1 and G_2 be TTSPN's with least and greatest points x_1, y_1 and x_2, y_2, respectively; then their *serial composition* is the graph G_s for which

$$P_{G_s} = P_{G_1} \cup P_{G_2} - \{x_2\}$$

$$E_{G_s} = E_{G_1} \cup E_{G_2} - \{(x_2,z) \mid (x_2,z) \in E_{G_2}\}$$

$$\cup \{(y_1,z) \mid (x_2,z) \in E_{G_2}\}$$

In short: G_s is obtained by identifying x_2 with y_1. Readily, G_s has x_1 as least point and y_2 as greatest point.

b) *Parallel composition*: Similarly, the *parallel composition* G_p of G_1 and G_2 is defined by

$$P_{G_p} = P_{G_1} \cup P_{G_2} - \{x_2, y_2\}$$

$$E_{G_p} = E_{G_1} \cup E_{G_2} - \{(x_2,z) \mid (x_2,z) \in E_{G_2}\}$$

$$-\{(z,y_2) \mid (z,y_2) \in E_{G_2}\}$$

$$\cup \{(x_1,z) \mid (x_2,z) \in E_{G_2}\}$$

$$\cup \{(z,y_1) \mid (z,y_2) \in E_{G_2}\}$$

In short: G_p is obtained by identifying x_2 with x_1 and y_2 with y_1; readily, it has x_1 and y_1 as least and greatest points.

It can be shown that a directed graph G with distinct least and greatest points x,y is a TTSPN if and only if any two given points p and q of G always appear in the same order on any walk (="undirected path") from x to y which passes through them.

A directed graph is called a <u>basis graph</u> if whenever (r,s) and (s,t) are edges, (r,t) is not an edge. It is called <u>semi-transitive</u> if whenever (r,t) is an edge, then (r,s) and (s,t) are edges for any point s on any path from r to t. (In both of these definitions, we assume that r,s,t are all distinct.) We shall call G <u>basic</u> if it is both a basis graph and semi-transitive. Readily, G is basic if and only if whenever there is a path of length ≥ 2 from p to q, there is no edge from p to q. All of these properties evidently pass to subgraphs.

<u>Theorem 3.2</u> The following context-free graph grammar has as its language the set of all basic TTSPN's:

$V = \{A, a\}$

$I = \{\, \overset{a}{\bullet} \longrightarrow \overset{a}{\bullet} \;,\; \overset{a}{\bullet} \longrightarrow \overset{A}{\bullet} \longrightarrow \overset{a}{\bullet} \,\}$

R consists of the rules

(R_1) $\overset{A}{\bullet} := \overset{A_{(1)}}{\bullet} \longrightarrow \overset{A_{(2)}}{\bullet}$

$E = \{(p, A_{(1)}) \mid (p,A)$ an edge in the host web$\}$
$\cup \{(A_{(2)}, p) \mid (A,p)$ an edge in the host web$\}$*

(R_2) $\overset{A}{\bullet} := \begin{matrix} \overset{A}{\bullet} \\ \overset{A}{\bullet} \end{matrix}$

$E = \{(p,A) \mid (p,A)$ an edge in the host web$\}$
$\cup \{(A,p) \mid (A,p)$ an edge in the host web$\}$

(R_3) $\overset{A}{\bullet} := \overset{a}{\bullet}$

E same as in (R_2)

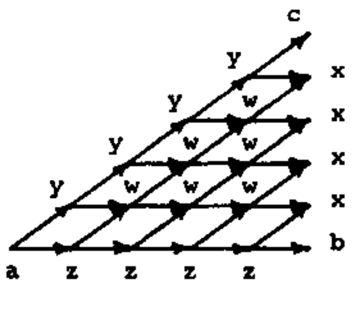
*We have attached subscripts to the two A's to avoid having to write "E is the set of edges from p to the point of β which is the first term of an edge, for all p such that there was an edge from p to the sole point of α in the host web, together with the set of edges to p from the point of θ which ..."; the A's are still the same label.

<u>Proof</u>: It is easily verified, using induction, that any web generated by this grammar is basic, has least and greatest points and has the "unique order" property, making it a basic TTSPN. Conversely, if G is a basic TTSPN having six or fewer points, it is readily generated by the grammar; we proceed by induction on the number $n \geq 7$ of points in G.

Since G is a TTSPN, it is the composition (serial or parallel) of two TTSPN's, call them G_1 and G_2, which are basic since they are subgraphs of G, and at least one of which, say G_1, has $n_1 \geq 4$ points. By induction hypothesis, G_1 is generated by the grammar; and since each application of a rewriting rule adds at most one point, there is a step G_1' in a "derivation" of G_1 from $\overset{a}{\bullet} \longrightarrow \overset{A}{\bullet} \longrightarrow \overset{a}{\bullet}$ at which there are exactly $n_1 - 1 \geq 3$ points. By the first part of the proof, G_1' is a basic TTSPN. Let G' be the web obtained by composing G_1' (rather than G_1) with G_2; thus G' is a TTSPN, and readily it is basic since G is. Since G' has n - 1 points, by induction hypothesis it is generated by the grammar; and if we now apply to it the same sequence of rewriting rules which was used to get G_1 from G_1', we clearly obtain G.//

<u>Context-Sensitive Web Languages: "Triangles"</u>

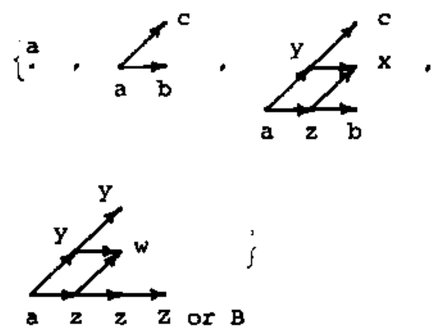
In this section we describe a context-sensitive web grammar which generates "directed triangles" of the form



(Note that these are directed graphs, not really triangles; but there is a natural way of "embedding" them in the plane -- as in the above picture -- so that the points of the graph coincide with the points of a triangle. The grammar given below should be compared to the right-triangle grammar given by Kirsch[3].)

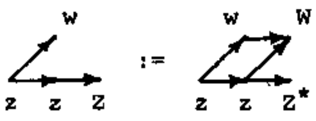
$V = \{B, Z, Z^*, W, W^*, X, a, b, c, w, x, y, z\}$

I =



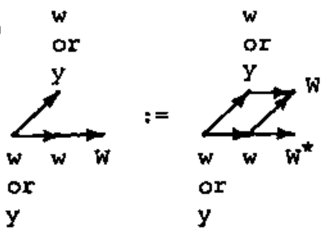
R consists of the rules

(R_1)

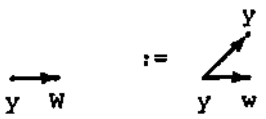


(where the embedding of the "new" vertex, labelled W, is as shown; the other vertices in the rewritten web are attached in the same way as were the corresponding vertices before the rewriting -- and similarly in the remaining rules)

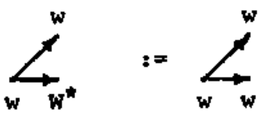
(R_{2-3})



(R_4)

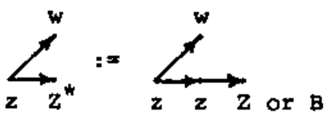


(R_5)

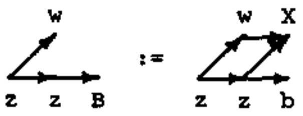


(i.e., W^* is simply relabeled w)

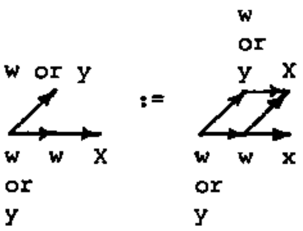
(R_{6-7})



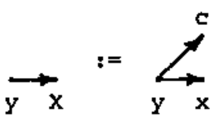
(R_8)



(R_{9-10})



(R_{11})



Readily, at any stage in the derivation of a terminal web using this grammar, exactly one of the rules can be applied. The grammar changes the initial Z to Z^*, and builds up a "column" of W^*'s, topped by a W, "on top" of it. When the column reaches the proper height, its top and next-to-top points become labelled y and w, and the W^*'s are then successively changed to w's until the bottom is reached. At this point, the Z^* can be changed to a z and a new Z (or a B) added "to the right" of it. If a Z is added, the process repeats; if a B (or if we started with the initial web having the B instead of the Z), the grammar changes it to a b and builds up a column of x's, topped by an X, on top of it, until the proper height is reached, at which the label c is used. Note that in each of these rules, only a single vertex is rewritten, so that this is indeed a context-sensitive web grammar.

The following interesting variation on the above grammar generates truncated "Pascal's triangles" having the binomial coefficients as labels. [This grammar requires an "unbounded" vocabulary, and makes use of rules in which the label of a "new" point is obtained by adding up the labels of neighboring "old" points.]

V = {B,Z,Z^*, positive integers, primed positive integers, and starred positive integers} (where the primed and starred integers are understood to be nonterminal)

$$I = \left\{ 1 \; , \; \begin{tikzcd} & 1 \\ 1 \arrow[ur] \arrow[r] & 1 \end{tikzcd} \; , \; \begin{tikzcd} & 1 & \\ 1 \arrow[ur] \arrow[r] & 1 \arrow[r] & Z \end{tikzcd} \right\}$$

R consists of the following rules (where the letters h,i,j,k,m,n,r, s,t stand for arbitrary positive integers):

(R_1)
$$\begin{tikzcd} & m & \\ 1 \arrow[ur] \arrow[r] & 1 \arrow[r] & Z \end{tikzcd} := \begin{tikzcd} & m \arrow[r] & (m+1)' \\ 1 \arrow[ur] \arrow[r] & 1 \arrow[ur] \arrow[r] & Z^* \text{ or } B \end{tikzcd}$$

(R_2)
$$\begin{tikzcd} & k & \\ h \arrow[ur] \arrow[r] & i \arrow[r] & j' \end{tikzcd} := \begin{tikzcd} & k \arrow[r] & (i+k)' \\ h \arrow[ur] \arrow[r] & i \arrow[ur] \arrow[r] & j^* \end{tikzcd}$$

(provided i $\neq$ 1)

(R_3)
$$\begin{tikzcd} 1 \arrow[r] & j' \end{tikzcd} := \begin{tikzcd} & 1 \\ 1 \arrow[ur] \arrow[r] & j \end{tikzcd}$$

(R_4)
$$\begin{tikzcd} & s \\ r \arrow[ur] \arrow[r] & t^* \end{tikzcd} := \begin{tikzcd} & s \\ r \arrow[ur] \arrow[r] & t \end{tikzcd}$$

(R_5)
$$\begin{tikzcd} & n \\ 1 \arrow[ur] \arrow[r] & Z^* \end{tikzcd} := \begin{tikzcd} & n & \\ 1 \arrow[ur] \arrow[r] & 1 \arrow[r] & Z \end{tikzcd}$$

(R_6)
$$\begin{tikzcd} & n \\ 1 \arrow[ur] \arrow[r] & B \end{tikzcd} := \begin{tikzcd} & n \\ 1 \arrow[ur] \arrow[r] & 1 \end{tikzcd}$$

Here again, starred integers are built up on top of the B (or Z), topped by a primed integer, until the proper height is reached, at which point the stars are successively removed and the B changed to a 1 (or the Z changed to 1 and another Z added).

In both of the above grammars, "contexts" are used which involve more than just the immediate "neighbors" of the point being rewritten -- that is, more than just the points having edges between them and the rewritten point. We can obtain a grammar using only such "immediate contexts" if we "hang" an extra point from the "bottoms" of the triangles (which, incidentally makes the "triangles" TTSPN's). For conciseness, we use the following notation in describing the embeddings: L(x) = the set of points joined by edges to x; R(x) = the set of points to which x is joined by an edge.

V = {T,T',A,B,C,t, and positive integers with subscripts a,b, or c}

$$I = \left\{ \begin{tikzcd} 1_c \arrow[r] & t \end{tikzcd} \; , \; \begin{tikzcd} & 1_a \arrow[dr] & \\ 1_c \arrow[ur] \arrow[dr] & & t \\ & 1_b \arrow[ur] & \end{tikzcd} \; , \; \begin{tikzcd} & & 1_a \arrow[ddr] & \\ & 1_a \arrow[ur] \arrow[dr] & & \\ 1_c \arrow[ur] \arrow[dr] & & 2_c \arrow[r] & T' \\ & 1_b \arrow[ur] \arrow[dr] & & \\ & & 1_b \arrow[uur] & \end{tikzcd} \right\}$$

R consists of the following rules (where we have described, rather than drawing, the contexts of the points which are rewritten):

Rule		Contextual condition	Embedding
(R_1)	$T := t$	NONE	L(t) = L(T)
(R_2)	$T := \begin{tikzcd} T' \arrow[r] & T \end{tikzcd}$	L(T) contains no points having nonterminal labels	L(T') = L(T)
(R_3)	$T' := \begin{matrix} 1_a \\ A \end{matrix}$	NONE	L(A) = L(T') $L(1_a) = \{1_a\} \subseteq L(T')$

[Note: In rules (R_3-R_9), $R(X) = T$ for all X in the right members]

	Rule	Contextual condition	Embedding
(R_4)	$\underset{\cdot}{A} := \begin{array}{l} \cdot n_a \\ \cdot \\ A \end{array}$	L(A) contains at least two points whose labels have "a" subscripts; here n_a is the sum of the two smallest of these labels, call them i_a and j_a, where $i_a < j_a$	$L(A) = L(A) - \{i_a\}$ $L(n_a) = \{i_a, j_a\}$
(R_5)	$\underset{\cdot}{A} := \begin{array}{l} \cdot n_a \\ \cdot \\ C \end{array}$	L(A) contains just one point i_a whose label has an "a" subscript, and a point j_c whose label has a "c" subscript; here $n_a = i_a + j_c$	$L(C) = L(A) - \{i_a\}$ $L(n_a) = \{i_a, j_a\}$
(R_6)	$\underset{\cdot}{A} := \begin{array}{l} \cdot n_c \\ \cdot \\ B \end{array}$	L(A) contains just one point i_a whose label has an "a" subscript, and no point whose label has a "c" subscript; here n_c is the sum of i_a and the largest label j_b of a point in L(A) having a "b" subscript	$L(B) = L(A) - \{i_a\}$ $L(n_c) = \{i_a, j_b\}$
(R_7)	$\underset{\cdot}{C} := \begin{array}{l} \cdot n_b \\ \cdot \\ B \end{array}$	NONE; n_b is the sum of the two largest labels $i_b < j_b$ in L(C) having "b" subscripts	$L(B) = L(C) - \{j_b\}$ $L(n_b) = \{i_b, j_b\}$
(R_8)	$\underset{\cdot}{B} := \begin{array}{l} \cdot n_b \\ \cdot \\ B \end{array}$	L(B) contains at least two points whose labels have "b" subscripts; n_b is the sum of the two largest of these, call them $i_b < j_b$	$L(B) = L(B) - \{j_b\}$ $L(n_b) = \{i_b, j_b\}$
(R_9)	$\underset{\cdot}{B} := \underset{\cdot}{1_b}$	$L(B) = \{1_b\}$	$L(1_b) = L(B)$

Acyclic Web Languages

In all of the examples given earlier, the web languages consist entirely of acyclic webs. [As indicated earlier, from the point of view of relations, acyclic graphs are a natural generalization of strings, since the "path" relation on a string is a total order relation, while on an acyclic graph it is a partial order relation.] In this section we show that for such languages, if we impose a certain natural restriction on the types of embeddings which are allowed, we can give a simple graph-theoretic characterization of the subwebs which can result from rewriting a single point. We can also show that any such subweb can be obtained by repeatedly rewriting one-point subwebs as one- or two-point subwebs; note that all the rewriting rules in our examples were indeed of this form.

The subgraph H of the graph G is called[4] convex in G if any path between two points of H lies completely in H.

Proposition 5.1 G is acyclic if and only if every subgraph on a single point is convex in G.

Proof: Evidently, in any graph, a subgraph on a single point is convex if the point does not lie on a cycle.//

Readily, the convex subgraphs of a string are just its substrings.

Let G and G' be graphs. A mapping φ from P_G onto $P_{G'}$ is called a graph homomorphism if

1) (p,q) in E_G implies $(\varphi(p),\varphi(q))$ in $E_{G'}$

2) (p',q') in $E_{G'}$ implies (a,b) in E_G for some a,b such that

$\varphi(a) = p'$ and $\varphi(b) = q'$.

<u>Proposition 5.2</u> <u>A homomorphic preimage of a convex subgraph is convex</u>.

<u>Proof</u>: Let φ: $P_G \to P_{G'}$ be a homomorphism, let H' be convex in G', and suppose that there were a path in G between two points p_1, p_2 of $\varphi^{-1}(H')$ which contained a point q not in $\varphi^{-1}(H')$. Then there would be a path in G' between the points $\varphi(p_1)$ and $\varphi(p_2)$ of H' containing the point $\varphi(q)$, which is not in H', contradicting the convexity of H'.//

Readily, a homomorphic image or preimage of a string need not be a string.

Let $\xi A \eta := \xi \beta \eta$ be a context-sensitive string rewriting rule which "expands" the single point A into the substring β. When we use this rule in reverse, to "parse" sentences in the string language, we are "contracting" the substring β into a single point. It is easily verified that if we regard the strings in question as graphs, this "contraction" defines a graph homomorphism of the original string onto the rewritten string. This observation suggests that in context-sensitive web grammars too, where the rewriting rules expand points into subwebs, it is natural to require that the corresponding "contractions" be graph homomorphisms. If we impose this requirement, and also require that the web be acyclic both before and after the contraction -- which is certainly true in the string case -- we shall show immediately below that the subwebs into which points are expanded must always be convex.

To see what is implied when we require that the contraction of a subgraph onto a point be a graph homomorphism, let G be a graph, and define the <u>left (right) neighborhood</u> of the subgraph H of G as the set of points of G not in H from (to) which there is an edge to (from) a point of H. Let φ be a function from G onto G' which is one-to-one, and edge-preserving outside H, and which maps H onto a single point p' (where if there is an edge in H, there is a loop at p'). Then readily, φ is a homomorphism if and only if it preserves left and right neighborhoods -- in other words, if the neighborhoods of the subgraph on {p'} in G' are just the images of the neighborhoods of H in G. [Note that if G is acyclic, the left and right neighborhoods of any one-point subgraph must be disjoint. In the string case, the left (right) neighborhood of a substring is evidently just the point immediately to the left (right) of its leftmost (rightmost) element.] Thus requiring that the reverse of a web rewriting rule be a homomorphism is equivalent to requiring that the embedding part of the rule be neighborhood-preserving.

We shall call the subgraph H of the acyclic graph G <u>contractable</u> if there exists a homomorphism φ of G onto an acyclic graph G' which maps H onto a single point, and is one-to-one on the rest of G; any such φ will be called a <u>contraction</u> of H.

<u>Theorem 5.3</u> <u>H is contractable if and only if it is convex</u>.

<u>Proof</u>: Let φ be a contraction of H onto the vertex y'. Since G' is acyclic, the subgraph of y' is convex; hence by Proposition 5.2, so is its preimage H.

Conversely, let H be convex, let φ be a homomorphism of G which takes H onto a single vertex y' and is one-to-one on the rest of G, and suppose that G' were not acyclic. If there were a cycle in G' not passing through y', then since φ is one-to-one outside H, this cycle would be the image of a cycle in G, contradicting the fact that G is acyclic. On the other hand, let $\varphi(p_0), \ldots, \varphi(p_n)$ be a cycle in G' with $\varphi(p_j) = y'$; then $\varphi(p_{j-1})$ and $\varphi(p_{j+1})$ are different from y', so that p_{j-1} and p_{j+1} are not in H. But there is an edge from p_{j-1} to a vertex r_j of H and from a vertex s_j of H to p_{j+1}, so that the path $s_j, p_{j+1}, \ldots, p_n = p_0, p_1, \ldots, p_{j-1}, r_j$ contradicts the convexity of H.//

We recall that a subgraph of a string is convex if and only if it is connected. Thus we are still dealing here with a generalization of the situation which is usually considered in string grammars -- namely, that the point a can be rewritten only as a connected (="continuous") substring; "discontinuous constituent" grammars are usually not allowed.

By Theorem 5.3, if we want to "parse" an acyclic graph to determine whether it could have been generated by our grammar, we need only examine its convex subgraphs.

Proposition 5.4 *Let G be an acylic graph, and let H be a convex subgraph of G which has at least two points; then there exist points p,q in H such that the subgraph on* {p,q} *is convex in* G.

Proof: This is clear if H has just two points; we proceed by induction. Let r be a maximal point of H; then the subgraph on $P_H - \{r\}$ is readily convex in G, so that by induction hypothesis we can choose the required p,q from $P_H - \{r\}$.//

Proposition 5.5 *A contraction of* H *takes convex subgraphs containing* H *into convex subgraphs*.

Proof: This follows immediately from the definition of convexity and the fact that a contraction is one-to-one outside H.//

These two propositions combine to give us

Theorem 5.6 *Any contraction is a composits of contractions each of which contracts a subgraph having exactly two points*.

Proof: To contract the convex subgraph H, first choose two points from it such that the subgraph on them is convex (Proposition 5.4), and contract this subgraph; the image of H under this "partial contraction" is still convex (Proposition 5.5), so that we can repeat the process until all of H has been contracted.//

A similar argument can be used to show that any homomorphism of an acyclic graph onto an acyclic graph is a composite of such "two-point" contractions. These results suggest that it may be possible to prove, for context-sensitive web grammars, an analog of Kuroda's result[5] that any context-sensitive string grammar is equivalent to one whose rewriting rules involve only strings of lengths ≤ 2.

References

[1] W. F. Miller and A. C. Shaw, Linguistic methods in picture processing - a survey, *Proc. Fall Joint Computer Conf.*, December 1968, 279-290.

[2] J. Feder, Linguistic specification and analysis of classes of line patterns, Ph.D. Dissertation, Department of Electrical Engineering, New York University, January 1969.

[3] R. A. Kirsch, Computer interpretation of English text and picture patterns, *I.E.E.E. Trans. EC-13*, August 1964, 363-376.

[4] J. L. Pfaltz, Convexity in graphs: Mathematical preliminaries to a theory of graph grammars, AD673420, July 1968 (abridged versions of parts of this report have been accepted for publication in the Journal of Combinatorial Theory and the Proceedings of the American Mathematical Society).

[5] S.-Y. Kuroda, Classes of languages and linear-bounded automata, *Info. Control 7*, June 1964, 207-223.