The multidimensional forest languages

by

Thomas Eugene O'Neil

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1985

TABLE OF CONTENTS

LIST OF DEFINITIONS AND THEOREMS

CHAPTER 1.
INTRODUCTION

The top two levels of the Chomsky language hierarchy play a major role in the translation of programming languages. Lexical and syntactic analyzers can be automatically generated given regular and context-free grammars. A large part of the work of syntactic analysis, however, lies outside the realm of context-free languages. Syntax rules such as those regarding type agreement, scope of names, and parameter correspondence cannot be expressed in a context-free grammar. Such rules are enforced during translation by ad hoc methods.

Despite the limitations of context-free languages, their use in syntactic analysis provides a framework for the code-generation phase of translation. Derivation trees are constructed during syntactic analysis. Syntax-directed translation uses these trees for code generation. The underlying theory characterizes translation as an operation on trees (Aho and Ullman 1977). The code produced by a syntax-directed translation of a program in a high-level language can be seriously suboptimal, and complete translation may require one or more optimization phases.

This is the state of the art in translation: regular and context-free grammars lay the foundation; ad hoc methods complete the process. This approach is considered satisfactory for translation of the high-level languages of today, and more sophisticated translation methods would probably not do much better. So why search for better translation methods? There is no need to -- as long as the high-level languages of today are adequate. But the tremendous expense of developing and maintaining reliable software leads us to believe that our high-level languages are not adequate. There is an urgent need for general-purpose, very-high-level languages which can be efficiently translated. So far no such languages have appeared. Compared to the whirlwind development of computer hardware over the last thirty years, the progress in programming languages has been disappointingly slow. Our languages under-utilize the capacity of the machines. Programmers and general users are burdened with mountains of detail, low-level algorithms, and inflexible syntax rules. The fact is that our so-called "high-level" languages are still inflexible and unnatural, and the limitations of these languages may be a direct result of the inadequacy of our theory of translation.

Regular and context-free grammars have been stretched as far as they will go. More powerful theoretical tools are needed for translation of higher-level languages. The next level in the Chomsky hierarchy is the class of context-sensitive languages. But this class is too broad and complex to be of use. Research has been continuing since the late 1960s to find reasonable extensions of the context-free languages and to discover non-context-free rewriting systems which generate context-free languages. This thesis presents results which contribute to that area of research.

Ginsburg and Greibach (1966) have investigated certain rewriting systems with context-dependent rules that delete symbols. These grammars were found to produce languages that are context-sensitive, and under some circumstances even context-free. Book (1972), Baker (1974), and Aggarwal and Heinen (1979) pursued this line of research to discover that rewriting systems can take into account constant-size, non-overlapping amounts of context and still produce context-free languages. While such rewriting systems do not give us a new class of languages, they do allow us to write simple grammars for languages whose context-free grammars would be formidable.

In 1968, Michael Fischer described macro languages, a class of languages larger than context-free and smaller than context-sensitive (Fischer 1968). Fischer showed a macro grammar which had the power to enforce declared type consistency on the right- and left-hand sides of assignment operators in a programming language. The grammar failed, however, to prevent multiple declarations of identifiers. Macro languages were observed to have two incomparable subclasses: inside-out (IO) and outside-in (OI). Both these subclasses contain interesting non-context-free languages, but neither subclass contains all the interesting new languages. The grammar mentioned above which enforced type consistency was a quoted macro grammar, one which selectively used both outside-in and inside-out rewriting rules. The results which Fischer obtained for IO and OI macro languages were not shown to apply to quoted macro languages. It remains unknown whether quoted macro grammars produce languages which are not context sensitive.

In the early 1970s, several people defined and investigated grammars and finite automata on trees (Rounds 1969 and 1970, Engelfriet 1975b). Recognizable sets of trees were defined to be those sets accepted by finite tree automata. It was found that the frontier operation applied to a recognizable set of trees produced a context-free language of strings. Similarly context-free grammars on trees produce sets whose frontiers yield macro languages of strings.

Tree automata have been generalized to define tree transducers -- tree automata with output (Baker 1975, Engelfriet 1975b). Tree-transducers applied to recognizable sets of trees can produce sets of trees whose frontiers form languages outside the class of context-free languages, but well within the class of context-sensitive languages. Tree automata and transducers have been used to formally describe syntax-directed translation (Baker 1975). This formalism covers only translation of the context-free portion of a language's syntax. Tree-transducers might be more beneficial if a programming language could be characterized as a transduction or composition of transductions so that the formal model would encompass the non-context-free features of the language. This approach, however, may not be feasible because tree transducers are not closed under composition (Baker 1979). Another unsettling feature of transducers is the presence of two incomparable subclasses, top-down and bottom-up (Engelfriet 1975a). This dichotomy is analogous to the problem of the OI and IO subclasses of macro languages (Engelfriet and Schmidt 1977, 1978).

Multidimensional trees have recently been introduced by Strawn (1982) and Baldwin (1983). These data structures are logical extensions of trees to higher dimensions. The 1-dimensional structures correspond to strings of symbols, the 2-dimensional structures correspond to trees, and the definition of higher dimensional structures is based on a generalization of the relationship between strings and trees. Just as a frontier operation can be applied to a tree to give a string, so can frontier operations be applied to higher-dimension structures to give strings.

The frontiers of multidimensional trees (md trees) form an infinite hierarchy of string languages properly contained within the class of context-sensitive languages. The first three levels of the hierarchy correspond to the regular, context-free, and IO macro languages respectively.

This thesis presents a modification of multidimensional trees which results in a hierarchy of string languages related to that discovered by Strawn and Baldwin. Most attention is focused on the second and third levels of this hierarchy and their relationship with other known classes of languages. It is hoped that multidimensional trees will provide a unifying framework for study of classes of languages between context-free and context-sensitive, and that further study will lead to the discovery of automatic and efficient methods for translating higher-level programming languages.

Chapter 2 presents some basic definitions and summarizes some results obtained by Strawn and Baldwin. Chapter 3 shows the correspondence of 1-dimensional tree-frontier languages and the regular languages. Chapter 4 explores the relationship between 2-dimensional tree-frontier languages and the context-free languages. Chapter 5 examines the 3-dimensional tree-frontier languages and their relationship with the context-sensitive languages. Chapter 6 relates the 3-dimensional tree frontier languages to the IO and OI macro languages. Chapter 7 contains a concluding summary and suggestions for further work.

CHAPTER 2.

MULTIDIMENSIONAL TREES AND FORESTS

The basic data structure investigated in this thesis is the multidimensional forest. The basic operation on multidimensional forests is the frontier operation, which can convert forests to strings. Grammars can be written to produce forest languages, and each forest language has an associated string language called its string yield.

Defining Trees and Forests

Multidimensional forests are structures containing nodes and arcs. The nodes are labeled with symbols: terminals, non-terminals, or auxiliary symbols called selectors. The arcs are labeled with natural numbers. The selectors and non-terminals are elements of indexed sets.

DEFINITION 2-1. An *indexed set* I is a set whose elements are pairs $<i, x>$ consisting of an integer index i and a symbol or string x. A subscripted reference to an indexed set I_n represents a subset containing every element of I whose index is n.

In order to specify multidimensional forests in one or two dimensions, an n-dimensional forests is represented as an n-ary tree. The correspondence between n-d forests and n-ary trees is a generalization of the correspondence between conventional forests and binary trees (Horowitz and Sahni 1976). A 3-d forest, for example, can be represented as a structure in which each node has at most three arcs, labeled 1, 2, and 3.

Every multidimensional forest has a dimension n and a degree k. If n is the largest label on any arc of a forest β, and k is the smallest label on any arc emanating from the root of β, then β is an n-dimensional, k-degree forest.

It should be noted that not every n-ary tree represents an n-dimensional forest. The informal rules below characterize multidimensional forests. These rules are useful for determining whether a given n-ary tree represents a multidimensional forest over a set of terminals Σ and a set of selectors Ξ.

1) Elements of Σ can label any nodes.

2) If a selector $<r, p>$ in Ξ labels a node, then the label on each arc emanating from the node can be at most $r-1$.

3) If the arc pointing to a node has label r, then the label on each arc emanating from the node must be at least $r-1$.

The two definitions below refer to one another to formally define multidimensional trees and forests. We will adopt the notational convention that $A\,[_n B\,]$ represents the set $\{\, a\,[_n b\,] \mid a \in A$ and $b \in B \}$. If B is the empty set, then the brackets will be elided, leaving $A\,[_n \Phi\,] = A$.

DEFINITION 2-2. Let Σ be a finite set of symbols and let Ξ be a finite indexed set of auxiliary symbols. Then the set of *n-dimensional trees* over Σ and Ξ, $H_n(\Sigma, \Xi)$, $n > 0$, is defined

$$H_0(\Sigma, \Xi) = \Sigma, \text{ and}$$

$$H_n(\Sigma, \Xi) = \Sigma \bigcup \Sigma\,[_n H_n^{n-1}(\Sigma, \Xi)\,] \bigcup \Xi_n\ .$$

DEFINITION 2-3. Let Σ be a finite set of symbols and let Ξ be a finite indexed set of auxiliary symbols. Then the set of *n-dimensional forests* over Σ and Ξ is defined

$$H_n^0(\Sigma, \Xi) = H_n^1(\Sigma, \Xi),$$
$$H_n^k(\Sigma, \Xi) = H_k(H_n^{k+1}(\Sigma; \Xi), \Xi) \text{ for } 0<k<n \text{, and}$$
$$H_n^n(\Sigma, \Xi) = H_n(\Sigma, \Xi).$$

These definitions are consistent with those developed by Baldwin (1983) and Strawn (1982). The set of 1-dimensional trees over Σ and Φ, where Φ is the empty set, corresponds to the conventional notion of the set of strings over Σ. The set of 2-dimensional trees over Σ and Φ corresponds to the conventional notion of the set of ordered trees over Σ represented as binary trees.

Figure 1 contains two 3-ary trees of which only one represents a valid 3-dimensional tree. The second tree is not a 3-d forest since the node d has a 3-arc pointing to it and a 1-arc emanating from it. This figure also shows a non-standard graphic representation of n-ary trees. This representation is preferred since it is easily printed and it corresponds directly to the multidimensional forest definition. The labeled arcs simply replace the labeled brackets.

The selectors in multidimensional trees and forests mark the places where substitution will occur during the frontier operation. Two universal selector sets are defined below: the set of standard selectors and the set of extended selectors.

DEFINITION 2-4. The set of *standard selectors* Ξ^s is an indexed set of elements $<n, x>$ where $n > 0$ and $x \in \{1, 2, \ldots, n-1\}^*$.

$r\ [_3\ a\ [_3\ b\ [_2\ d\ [_2\, x\ [_1\ y]][_1 e\ [_2\ f\ [_1\ y]]]]][_2 b\ [_2\ c\ [_1\ y]]]]$

```
r---3
    |
    a---3----------------------------2
        |                            |
        b---2                        b---2
            |                            |
            d---2-------1                c---1
                |       |                    |
                x---1   e---2                y
                    |       |
                    y       f---1
                                |
                                y
```

a valid 3-d tree

$r\ [_3\ a\ [_3\ d\ [_2\, x\ [_1\ y]][_1\ e\ [_1\ f\ [_1\ y]]]][_2 b\ [_2\ c\ [_1\ y]]]]$

```
r---3
    |
    a---3------------------------2
        |                        |
        d---2-------1            b---2
            |       |                |
            x---1   e---1            c---1
                |       |                |
                y       f---1            y
                            |
                            y
```

not a valid 3-d tree

FIGURE 1. 3-ary tree examples

DEFINITION 2-5. The set of *extended selectors* Ξ^e is an indexed set of elements $<n, x>$ where $n > 0$ and $x \in \{1, 2, \ldots, n\}^*$.

The standard selectors are those used in the work of Strawn (1982) and Baldwin (1983). The extended selectors are introduced here for the first time, and the rest of this thesis is devoted to assessing the impact of this seemingly minor change.

The Frontier Operation

The frontier operation reduces the dimension of a forest. It involves the selection and substitution of subtrees. The selectors in a forest mark the places where substitution can occur and specify what subtrees will be moved or copied. Each selector has two components: an index and a path. The index specifies the dimension of the forest to which the selector can be applied as well as the dimension of the tree which will be selected. The path is a sequence of arc labels used by the selection function to pick out a subtree for substitution. A path is traced in a forest by starting at the root node and traversing the arcs whose labels match those in the path. For example, the path of selector $<3, 3321>$ can be traced to the node e in the valid 3-d tree of figure 1. The tree selected is just e. The subforest $f\ [_1\ y]$ is dropped, since the arc pointing to it is not a 3-arc.

DEFINITION 2-6. The selection function $sel(<n, p>, \beta) : \Xi_n \times H_n^1(\Sigma, \Xi) \rightarrow H_n(\Sigma, \Xi)$ is defined

$sel(<n, \lambda>, \alpha) = \alpha$ for $\alpha \in H_n(\Sigma, \Xi)$ and λ the empty string,

$sel(<n, \lambda>, \alpha[_k\, \gamma]) = sel(<n, \lambda>, \alpha)$ for $k < n$,

$\alpha \in H_n^{k+1}(\Sigma, \Xi)$, and $\gamma \in H_n^{k-1}(\Sigma, \Xi)$.

$sel\,(<n\,, ki>,\ \alpha[_j\,\gamma]\,) = sel\,(<n\,.\,ki>,\ \alpha)$ for $\alpha \in H_n^{j+1}(\Sigma, \Xi)$,

$\gamma \in H_n^{j-1}(\Sigma, \Xi)$, $j < k$,

$sel\,(<n\,, ki>,\ \alpha[_k\,\gamma]\,) = sel\,(<n\,, i>,\ \gamma)$ for $\alpha \in H_n^{k+1}(\Sigma, \Xi)$

and $\gamma \in H_n^{k-1}(\Sigma, \Xi)$, and

$sel\,(<n\,, p>,\ \beta)$ is otherwise undefined.

There are situations in which the selection function is not defined. Let β be the valid 3-d forest of figure 1. $sel\,(<3, 321>,\ \beta)$ is not defined because the node b has no 1-arc. This is called a path error. If the y in β is a selector $<1, \lambda>$, then $sel\,(<3, 3221>,\ \beta)$ is also undefined. In this case, we successfully trace 3221 to y, but $y = <1, \lambda>$ is not an element of $H_3(\Sigma, \Xi)$ as required.

The substitution function $subs_n\,(\alpha,\ \beta)$ operates on two n -dimensional forests. It simply replaces each selector $<n\,, p>$ in α with $sel\,(<n\,, p>,\ \beta)$.

DEFINITION 2-7. The substitution function $subs_n\,(\alpha,\ \beta)$: $H_n^1(\Sigma, \Xi) \rightarrow H_n^1(\Sigma, \Xi)$ is defined

$subs_n\,(a\,, \beta) = a$ for $a \in \Sigma \bigcup \Xi_j$, $j < n$,

$subs_n\,(x\,, \beta) = sel\,(x\,, \beta)$ for $x \in \Xi_n$, and

$subs_n\,(\alpha[_m\,\gamma]\,, \beta) = subs_n\,(\alpha,\ \beta)\,[_m\, subs_n\,(\gamma,\ \beta)\,]$

for $m \leqslant n$, $\alpha \in H_n^{m+1}(\Sigma, \Xi)$, and $\gamma \in H_n^{m-1}(\Sigma, \Xi)$.

A call to the frontier function $fr_n\,(\alpha)$ reduces the dimension of α to n. If the dimension of α exceeds $n+1$, then $fr_{n+1}(\alpha)$ is performed first. If the dimension of α is less than $n+1$, then the frontier operation has no effect. Any nodes with arcs labeled $n+1$ are removed by fr_n. A node with an $n+1$-arc but no n -arc is simply eliminated. A node with both an $n+1$-arc and an n -arc is processed by

```
#---3-------2
    |       |
    #---2   #---2
        |       |
        x---1   #---2-------1
            |       |       |
            y       a---1   #---2-------1
                        |       |       |
                        a---1   b---1   #---2
                            |       |       |
                            z       b---1   c---1
                                        |       |
                                        z       c---1
                                                    |
                                                    z
```

3-d forest with selectors x = <2, 211>, y = <2, 2>, z = <1, λ>

```
#---2
    |
    #---2-------1
        |       |
        c---1   #---2
            |       |
            c---1   a---1
                |       |
                z       a---1
                            |
       2-d frontier         z
```

```
c---1
    |
    c---1
        |
        a---1
            |
            a---1
                |
 1-d frontier   z
```

FIGURE 2. A 3-d forest and its frontiers

frontiering the subforests and calling the substitute function: $fr_n(\#[_{n+1}\beta][_n\gamma]) = subs_n(fr_n(\beta), fr_n(\gamma))$. The frontier function $fr_n(\alpha)$ is undefined if α is a selector, $<r, p>$, and $r > n$.

DEFINITION 2-8. The frontier function $fr_n(\alpha): H_m^k(\Sigma, \Xi) \rightarrow H_n^k(\Sigma, \Xi)$ is defined when $m > n+1$ as $fr_n(\alpha) = fr_n(fr_{n+1}(\alpha))$, and when $m \leqslant n+1$ as

$fr_n(b) = b$ for $b \in \Sigma \bigcup \Xi_i$ where $i \leqslant n$,

$fr_n(b[_{n+1}\beta]) = fr_n(\beta)$ for $b \in \Sigma \bigcup \Xi$ and $\beta \in H_m^n(\Sigma, \Xi)$,

$fr_n(b[_{n+1}\beta][_n\gamma]) = subs_n(fr_n(\beta), fr_n(\gamma))$
for $b \in \Sigma \bigcup \Xi$, $\beta \in H_m^n(\Sigma, \Xi)$, and $\gamma \in H_m^{n-1}(\Sigma, \Xi)$,

$fr_n(\beta[_r\gamma]) = fr_n(\beta)[_r fr_n(\gamma)]$
for $k \leqslant r \leqslant n$, $\beta \in H_m^{r+1}(\Sigma, \Xi)$, and $\gamma \in H_m^{r-1}(\Sigma, \Xi)$, and

$fr_n(\alpha)$ is otherwise undefined.

Figure 2 contains an example of a 3-d forest and its 2-d and 1-d frontiers. This figure also illustrates that the frontier operation has the power to delete subforests. The subtree $\#[_2 b[_1 b[_1 z]]]$ is unselected during the 2-d frontier, and so it is eliminated. Deletion will also occur during $fr_n(\alpha)$ if $\alpha = \#[_{n+1}\gamma][_n\delta]$ and γ contains no n-dimensional selectors. In that case, $fr_n(\#[_{n+1}\gamma][_n\delta]) = fr_n(\gamma)$.

The frontier operation also has the power to make multiple copies of subtrees. This happens when a substitution is made into a subtree that has multiple occurrences of the same selector. Since the frontier operation has both copying power and deleting power, the result of a frontier operation can be either larger or smaller than the original forest.

Grammars and Languages

Sets of multidimensional forests can be generated by rewriting systems called regular multidimensional forest grammars. These grammars are regular because of the restricted placement of non-terminal symbols in the replacement rules.

DEFINITION 2-9. A *regular n-dimensional forest grammar* is a formal system $<\Sigma, \Xi, N, R, S>_n^k$, $n > 0$, $1 \leqslant k \leqslant n$, where

Σ is a finite set of terminal symbols,

Ξ is a finite set of selectors with indices in $\{1, 2, \ldots, n\}$,

N is a finite indexed set of non-terminal symbols with indices in $\{1, 2, \ldots, n\}$,

R is a finite set of replacement rules of the form $A \rightarrow \beta$ where $A \in N_i$ and $\beta \in H_n^i(\Sigma, \Xi \bigcup N)$ for $1 \leqslant i \leqslant n$, and

S is the start symbol in N_k.

A non-terminal in a forest grammar generates a set of forests by repeated application of the replacement rules. The intermediate structures, which contain a mix of terminals, selectors, and non-terminals, are called *structural forms*. The process of applying the replacement rules is a derivation. A derivation step is the application of a single rule to obtain one structural form from another.

DEFINITION 2-10. A structural form α of a forest grammar $<\Sigma, \Xi, N, R, S>_n^k$ *directly derives* another structural form β, $\alpha \Rightarrow \beta$, if and only if R contains a rule $A \rightarrow \gamma$ and β can be obtained from α by replacing an occurrence of A with γ.

DEFINITION 2-11. A structural form α of a forest grammar *derives* another structural form β, $\alpha \Rightarrow^* \beta$, if $\alpha = \beta$ or there are structural forms $\alpha_1, \ldots, \alpha_m$, $m \geqslant 0$, such that $\alpha \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_m \Rightarrow \beta$.

DEFINITION 2-12. The *language* generated by a forest grammar $G = <\Sigma, \Xi, N, R, S>_n^k$ is defined as $L(G) = \{\beta \mid S \Rightarrow^* \beta \text{ and } \beta \in H_n^k(\Sigma, \Xi)\}$.

The language generated by a grammar is thus defined as the set of forests each of which contains no non-terminals and can be derived from the start symbol. The frontier function can be applied to forests generated by a grammar to give sets of lower-dimensioned forests. These lower-dimensioned sets are the yields of the grammar. The 0-dimensional yield will be defined to establish a direct relationship between 1-dimensional trees and strings.

DEFINITION 2-13. The string function $str(\alpha): H_1(\Sigma, \Xi) \bigcup \{\lambda\} \to \Sigma^*$ is defined

$str(\lambda) = \lambda$,

$str(a) = a$ for $a \in \Sigma$,

$str(x) = \lambda$ for $x \in \Xi$,

$str(x[_1\,\beta]) = str(\beta)$ for $x \in \Xi$, and

$str(a[_1\,\beta]) = a \cdot str(\beta)$ for $a \in \Sigma$.

where $\cdot$ represents string catenation.

DEFINITION 2-14. The *string frontier* of a forest $\beta \in H_n^k(\Sigma, \Xi)$ is defined as $sfr(\beta) = str(fr_1(\beta))$.

DEFINITION 2-15. The *m-dimensional yield* of a grammar $G = <\Sigma, \Xi, N, R, S>_n^k$ is defined as

$Y_0(G) = \{\alpha \mid \beta \in L(G) \text{ and } str(fr_1(\beta)) = \alpha\}$, and

$Y_m(G) = \{\alpha \mid \beta \in L(G) \text{ and } fr_m(\beta) = \alpha\}$ for $1 \leqslant m \leqslant n$.

Some major results of Baldwin (1983) can be summarized in terms of the above definitions. Baldwin established the existence of an infinite hierarchy of string languages which he called the algebraic hierarchy: $ALG_n^1 \subset ALG_{n+1}^1$ for $n \geqslant 1$, and $ALG_n^1 \subset CS$, where CS represents the class of context-sensitive languages. ALG_n^1 is the class of languages obtained by taking the 1-d yield of n-dimensional forest grammars which use only standard selectors. More formally, $L \in ALG_n^1$ if and only if $L = Y_1(G)$ for some grammar $G = <\Sigma, \Xi, N, R, S>_n^k$ where $1 \leqslant k \leqslant n$ and $\Xi \subset \Xi^s$. We know from Baldwin's work that ALG_1^1 is the class of regular languages, ALG_2^1 is the class of context-free languages, and ALG_3^1 is the class of IO macro languages.

Language classes corresponding to the algebraic hierarchy can be defined which allow the use of extended selectors. These new classes will be represented as $mYnF$ for $n \geqslant 1$. $mYnF$ is an acronym for the m-dimensional Yields of regular sets of n-dimensional Forests.

DEFINITION 2-16. A language L belongs to the language class $mYnF$, $n \geqslant 1$ and $m \geqslant 0$, if and only if $L = Y_m(G)$ for some grammar $G = <\Sigma, \Xi, N, R, S>_n^k$, where $1 \leqslant k \leqslant n$ and $\Xi \subset \Xi^e$.

In the chapters that follow, we explore the relationship of the first three levels of the algebraic hierarchy and the corresponding $0YnF$ language classes.

CHAPTER 3.

ONE-DIMENSIONAL FOREST YIELD LANGUAGES

The string languages which are yields of regular sets of 1-dimensional forests, 0Y1F, can be shown to be equivalent to the class of regular languages. A method for converting a 1-d forest grammar to a regular grammar is given in this chapter.

One-Dimensional Normal Form

A 1-d forest grammar can be made to look like a regular string grammar by putting it in 1-dimensional normal form.

DEFINITION 3-1. A grammar $G = <\Sigma, \Xi, N, R, S>_1^1$ is in 1*-dimensional normal form*, 1DNF, if and only if every rule has the form

1) $A \rightarrow a[_1 B]$ for A, $B \in N$, and $a \in \Sigma$,

2) $A \rightarrow a$ for $A \in N$, and $a \in \Sigma$, or

3) $A \rightarrow \lambda$ for $A \in N$ and λ the empty string.

Note that a 1DNF grammar makes no use of selectors. This is appropriate because the selectors in a 1-d forest are nothing more than endmarkers which will be removed when the forest is converted to a string. The conversion of an arbitrary 1-d forest grammar to 1DNF is described in two steps. The first step is to add non-terminals to the grammar so that the right-hand side of each rule contains at most one terminal and one non-terminal.

LEMMA 3-2. If G is a grammar $<\Sigma, \Xi, N, R, S>_1^1$ then there is a grammar $G' = <\Sigma', \Xi', N', R', S'>_1^1$ such that (1) $L(G) = L(G')$ and (2) if

$A \rightarrow a[_1 \beta] \in R'$ then $\beta \in N'$.

PROOF. Construct G' from G by introducing new non-terminals C_i as required. Set $\Sigma' = \Sigma$, $\Xi' = \Xi$, $S' = S$, and construct R' and N' according to the algorithm below. Assume $A \in N'$ and $a \in \Sigma'$.

```
Put all the rules in $R$ in $R'$.
Set $N' = N$.
Set $i = 0$.
Repeat
    Add 1 to $i$.
    Find $A \rightarrow a[_1 \alpha]$ in $R'$ where $\alpha$ is not in $N'$.
    Invent new non-terminal $C_i$ and put it in $N'$.
    Replace $A \rightarrow a[_1 \alpha]$ in $R'$ with $A \rightarrow a[_1 C_i]$.
    Add $C_i \rightarrow \alpha$ to $R'$.
Until no rule $A \rightarrow a[_1 \alpha]$ can be found where $\alpha$ is not in $N'$.
```

It should be clear that this construction does not affect the language generated by the grammar. We have only introduced more non-terminals and rules to make the derivations longer. So $L(G) = L(G')$. QED.

The next step in converting a 1-d forest grammar to normal form is to remove the selectors. This construction is described in the proof of lemma 3-3 below.

LEMMA 3-3. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_1^1$ such that if R contains a rule $A \rightarrow b[_1\ B]$, then $B \in N$. There is a 1DNF grammar $G = <\Sigma', \Xi', N', R', S'>_1^1$ such that $S' \Rightarrow^* \alpha$ and $str(\alpha) = w$ if and only if

$S \Rightarrow^* \beta$ and $str(\beta) = w$, where $|w| \geqslant 0$.

PROOF. Construct G' from G by first setting $\Sigma' = \Sigma$, $\Xi' = \Phi$, and $S' = S$. Then, construct R' and N' from R and N according to the steps below. Assume that $A, B, C \in N, a, b \in \Sigma$, and $x \in \Xi$.

1) For every pair of rules $A \rightarrow a[_1 B]$ and $B \rightarrow b$ in R, add $A' \rightarrow a[_1 B']$ and $B' \rightarrow b$ to R', and add A' and B' to N'.

2) For every pair of rules $A \rightarrow a[_1 B]$ and $B \rightarrow b[_1 C]$ in R, add $A' \rightarrow a[_1 B']$ and $B' \rightarrow b[_1 C']$ to R', and add A', B' and C' to N'.

3) For every pair of rules $A \rightarrow a[_1 B]$ and $B \rightarrow x$ in R, add $A' \rightarrow a$ to R', and add A' to N'.

4) If R has a rule $S \rightarrow x$, add $S' \rightarrow \lambda$ to R' and add S' to N'.

We can show that G' satisfies the requirements of lemma 3-3 in two parts.

(A) If $A \Rightarrow^* \alpha$ and $str(\alpha) = w$ then $A' \Rightarrow^* \beta$ and $str(\beta) = w$.

Proof of (A) by induction on l, the length of w.

Base. $|w| \leqslant 1$.

Case 1. $A \Rightarrow^* x$ and $str(x) = \lambda$.

In this case, R has a rule $A \rightarrow x$ and R' has a rule $A' \rightarrow \lambda$ from step 4 of the construction of G'. It is clear that $A' \Rightarrow^* \lambda$ and $str(\lambda) = \lambda$.

Case 2. $A \Rightarrow^* a[_1 x]$, and $str(a[_1 x]) = a$.

In this case, R has rules $A \rightarrow a[_1 B]$ and $B \rightarrow x$, and R' has a rule $A' \rightarrow a$. So $A \Rightarrow^* a[_1 x]$ and $A' \Rightarrow^* a$ and $str(a[_1 x]) = str(a) = a$.

Case 3. $A \Rightarrow^* a$ and $str(a) = a$.

In this case, R has a rule $A \rightarrow a$ and R' has a rule $A' \rightarrow a$. So $A \Rightarrow^* a$ and

$A' \Rightarrow^* a$ and $str(a) = a$.

Inductive hypothesis: Assume (A) is true for $1 < l = n-1$.

Inductive step: Show (A) is true for $|w| = l = n$.

If $l > 1$ then $w = aw'$ where $|w'| = n-1$. We know that $A \Rightarrow^* \alpha$ and $str(\alpha) = aw'$. This means there must be a rule $A \rightarrow a[_1 B]$ in R where $B \Rightarrow^* \alpha'$, $str(\alpha') = w'$, and $|w'| \geqslant 1$. Since $|str(\alpha')| \geqslant 1$, there must be a rule $B \rightarrow b[_1 C]$ or $B \rightarrow b$ in R. But if R contains such rules for A and B, then R' contains $A' \rightarrow a[_1 B']$ and rules for B' constructed from B. By the inductive hypothesis, $B' \Rightarrow^* \beta'$ and $str(\beta') = w'$. So $str(a[_1 \beta']) = str(a[_1 \alpha']) = w$, and setting $\beta = a[_1 \beta']$, we have $str(\beta) = w$.

(B) If $A' \Rightarrow^* \beta$ and $str(\beta) = w$, then $A \Rightarrow^* \alpha$ and $str(\alpha) = w$.

The proof of (B) is similar to the proof of (A). QED.

THEOREM 3-4. If G is a grammar $<\Sigma, \Xi, N, R, S>_1^1$ then there is a 1DNF grammar $G' = <\Sigma', \Xi', N', R', S'>_1^1$ such that $Y_0(G') = Y_0(G)$.

PROOF. This theorem follows from the previous two lemmas. We transform G to G' by applying the construction of lemma 3-2 followed by the construction of lemma 3-3. Lemma 3-3 assures us that $w \in Y_0(G')$ if and only if $w \in Y_0(G)$. QED.

Equivalence of 0Y1F with the Regular Languages

Theorem 3-4 establishes that every 1-d regular forest grammar can be transformed to a 1DNF grammar. 1DNF grammars are directly related to regular grammars, and this allows us to assert that 0Y1F and the class of regular languages are equivalent.

THEOREM 3-5. The class of languages 0Y1F is equivalent to the class of regular languages.

PROOF. According to theorem 3-4, every language in 0Y1F is $Y_0(G)$ for some 1DNF grammar G. If we construct a grammar G' by removing the brackets from the right-hand sides of the rules of G, we have a regular grammar such that $L(G') = Y_0(G)$. Similarly, if we construct G' from a regular grammar G by adding brackets to the rules of G, we have a 1DNF grammar such that $Y_0(G') = L(G)$. QED.

COROLLARY 3-6. The class of languages 1Y1F corresponds to the class ALG_1^1.

PROOF. This corollary follows directly from theorem 3-5 and theorem 73 of Baldwin (1983), which establishes that every language in ALG_1^1 is a regular set if the brackets and selectors are removed. In 1-d forests, it doesn't matter whether extended selectors or standard selectors are used because the selectors are never applied in a frontier operation. QED.

CHAPTER 4.

TWO-DIMENSIONAL FOREST YIELD LANGUAGES

In 2-d forests, the difference between standard and extended selectors becomes significant. It is possible for the paths of extended selectors to overlap one another, and this gives the frontier operation more deleting power. This chapter establishes that a large subclass of 0Y2F is equivalent to the class of context-free languages.

2-d Forest Grammars with Standard Selectors

Only selectors in Ξ_1 are applied during a 1-d frontier operation on a 2-d forest, and the only selector in Ξ_1^s is $<1, \lambda>$. Thus, 2-d forests with standard selectors are strictly non-deleting, and it is easy to show that $ALG_2^1 \subseteq$ 1Y2F. The theorem below is presented for n-dimensional languages. The special case $n = 2$ is of immediate interest.

THEOREM 4-1. If $L \in ALG_n^1$ then $L \in$ 1YnF, $n \geqslant 1$.

PROOF. Strings in ALG_n^1 languages are the yields of n-d forests produced by regular forest grammars using only standard selectors. Since $\Xi^s \subset \Xi^e$, a n-d forest grammar with standard selectors is also a n-d forest grammar with extended selectors. So any language in ALG_n^1 is also in 1YnF. QED.

COROLLARY 4-2. If L is a context-free language, then $L \in$ 0Y2F.

PROOF. This follows immediately from the previous theorem, since Baldwin has shown that every context-free language corresponds to an ALG_2^1 language (Baldwin, 1983). QED.

THEOREM 4-3. If G is a 2-d forest grammar $<\Sigma, \Xi, N, R, S>_2^k$ for $1 \leqslant k \leqslant 2$ and $\Xi \subset \Xi^s$, then $Y_0(G)$ is a context-free language.

PROOF. If G has only standard selectors, then it can be converted directly to a context-free grammar G'. The conversion method is summarized as

1) $A \rightarrow a[_2 B][_1 C]$ becomes $A \rightarrow BC$,

2) $A \rightarrow a[_2 B]$ becomes $A \rightarrow B$,

3) $A \rightarrow a[_1 B]$ becomes $A \rightarrow aB$, and

4) $A \rightarrow x^0$ becomes $A \rightarrow \lambda$.

A simple induction on the length of a derivation will show that $L(G') = Y_0(G)$. The induction is not shown here. QED.

Two-Dimensional Normal Form

To facilitate the analysis of 2-d regular forest grammars which involve deletion, a normal form is defined below which will not affect the string yields of 2-d grammars. The normal form will prevent generation of some forests which are not frontierable and also some forests which contain useless subforests that would be eliminated by the frontier operation.

DEFINITION 4-4. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$. G is in *2–dimensional normal form*, 2DNF, if and only if every rule in R fits one of the following forms:

1) $A \rightarrow a[_2 B]$ for $A \in N_2$, $a \in \Sigma$, and $B \in N_1$,
2) $B \rightarrow a[_2 C][_1 D]$ for $B, C, D \in N_1$ and $a \in \Sigma$,
3) $B \rightarrow a[_1 C]$ for $B, C \in N_1$ and $a \in \Sigma$, or
4) $B \rightarrow x$ for $B \in N_1$ and $x \in \Xi_1$.

This normal form is analogous to Chomsky normal form for context-free grammars. Note that selectors in Ξ_2 are excluded and that forests without selectors in Ξ_1 cannot be generated. The constructions of lemma 4-6 through theorem 4-11 will show how to convert an arbitrary 2-d grammar into a 2DNF grammar. The first step in converting a grammar to normal form is to introduce new non-terminals and rules so that only single non-terminals appear inside brackets in the right-hand side of a rule, and terminals or selectors appear only outside brackets.

DEFINITION 4-5. A 2-d forest grammar $G = <\Sigma, \Xi, N, R, S>_2^k$ is a *short–rule* grammar if every rule in R fits one of the following forms for $A, B, C \in N$, $a \in \Sigma$, and $b \in \Sigma \bigcup \Xi$:

1) $A \rightarrow a[_2 B]$,
2) $A \rightarrow a[_2 B][_1 C]$,
3) $A \rightarrow b[_1 B]$, or
4) $A \rightarrow b$.

LEMMA 4-6. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$. There is a short-rule grammar $G' = <\Sigma', \Xi', N', R', S'>_2^k$ such that $Y_1(G) = Y_1(G')$.

PROOF. Construct G' from G by introducing new non-terminals C_i as required. Set $\Sigma' = \Sigma$, $\Xi' = \Xi$, $S' = S$, and construct R' and N' according to the algorithm below. Assume A , $B \in N'$ and $a \in \Sigma' \bigcup \Xi'$.

```
Put all the rules of $R$ in $R'$.
Set $N' = N$.
Set $i = 0$.
Repeat
    If $R$ has a rule $A \rightarrow \alpha[_r\ \beta]$ for $\beta$ not in $N'$ , $\alpha \in H_2^2(\Sigma, \Xi)$ then
        Add 1 to $i$.
        Invent $C_i$ and put it in $N_1'$.
        Replace $A \rightarrow \alpha[_r\ \beta]$ in $R'$ with $A \rightarrow \alpha[_r\ C_i]$.
        Add $C_i \rightarrow \beta$ to $R'$.
    Else if $R'$ has a rule $A \rightarrow a[_2\ \alpha][_1\ B]$ where $\alpha$ not in $N_1'$ then
        Add 1 to $i$.
        Invent $C_i$ and put it in $N_1'$.
        Replace $A \rightarrow a[_2\ \alpha][_1\ B]$ in $R'$ with $A \rightarrow a[_2 C_i][_1\ B]$.
        Add $C_i \rightarrow \alpha$ to $R'$.
    Else if $R'$ has rules $A \rightarrow B[_1\ \alpha]$ and $B \rightarrow a[_2\ \beta]$ or $B \rightarrow a$ then
        Replace $A \rightarrow B[_1\ \alpha]$ with rules $A \rightarrow a[_2\ \beta][_1\ \alpha]$ or $A \rightarrow a[_1\ \alpha]$
            using right-hand side of every rule for $B$.
    Else if $R'$ has rules $A \rightarrow B$ and $B \rightarrow \beta$ for $\beta$ not in $N$ then
        If $A = B$ then remove the rule from $R'$.
        Else replace $A \rightarrow B$ in $R'$ with $A \rightarrow \beta$.
Until every rule is in short-rule form.
```

It should be clear that this construction does not affect the language generated by the grammar. We have only introduced more non-terminals and rules without altering the forests which are derived. So $L(G) = L(G')$ and $Y_1(G) = Y_1(G')$. QED.

Any grammar can be partitioned into subgrammars by letting non-terminals other than S be the start symbol. If G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$, then G_A will represent the subgrammar of G whose start symbol is A. It will also be useful to partition a grammar G into subgrammars based on the last characters of the elements of $Y_1(G)$. The last character of $\sigma \epsilon\ H_1^1(\Sigma, \Xi)$ is defined as the label on the only node of σ which does not have an arc emanating from it.

DEFINITION 4-7. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$. Then $G_B:\Delta$ or $B:\Delta$, where $B \epsilon N$ and $\Delta \subset \Xi$, represents a subgrammar of G such that $\sigma \epsilon Y_1(B:\Delta)$ if and only if $B \Rightarrow^* \beta$, $fr_1(\beta) = \sigma$, and the last character of σ is in Δ.

Algorithms for generating $G_B:\Delta$ from a grammar G are presented in the proofs of the next two lemmas. The first lemma deals with the special case where $\Delta = \Xi$.

LEMMA 4-8. Suppose G is a short-rule grammar $<\Sigma, \Xi, N, R, S>_2^k$. There is an effective procedure for construction of $G_B:\Xi$ where $B \epsilon N$.

PROOF. $G_B:\Xi$ can be constructed simply by removing non-terminals and rules from G as follows:

1) Remove all non-terminals from N and rules from R which cannot be reached from B. This gives the subgrammar G_B.

2) Remove all productions of the from $A \rightarrow a$ where $A \in N$ and $a \in \Sigma$.

3) Eliminate non-terminals and rules which don't derive forests in $H_2^1(\Sigma, \Xi)$.

The resulting grammar does not generate any forest whose frontier ends with a terminal, for that would require a production $A \rightarrow a$. Also, since rules of the form $A \rightarrow x$ where $x \in \Xi$ were not removed, all the forests derived from B which have external selectors are unaffected. QED.

The next lemma handles the general case of $G_B:\Delta$. The proof will call for the merging of subgrammars with statements like "add G_A to G_B." This should be understood as an operation on G_B which sets $\Sigma_B = \Sigma_B \bigcup \Sigma_A$, $N_B = N_B \bigcup N_A$, $R_B = R_B \bigcup R_A$, $\Xi_B = \Xi_B \bigcup \Xi_A$, and S_B remains unchanged.

LEMMA 4-9. Let G be a short-rule grammar $<\Sigma, \Xi, N, R, S>_2^k$. There is an effective procedure for generating $G_B:\Delta = <\Sigma', \Xi', N', R', S'>_2^k$ where $B \in N$ and $\Delta \subset \Sigma \bigcup \Xi$.

PROOF. To construct $G_B:\Delta$, first remove all non-terminals and rules from N and R that are not reachable from B. Then, build $G_B:\Delta$ following the algorithm below. Assume that $A, B, C \in N$, $a \in \Sigma$, and $b \in \Sigma \bigcup \Xi$.

```
Set $N' = \{\}$, $R' = \{\}$, $\Xi' = \Xi$, and $\Sigma' = \Sigma$.
For each rule in $R$ of the form $A \rightarrow b$
    Put $A$ in $N'$.
    If $b \in \Delta$ then put $A \rightarrow b$ in $R'$.
    Else don't add the rule to $R'$.
    Remove $A \rightarrow b$ from $R$.
Repeat
    Select from $R$ a rule $A \rightarrow \beta$ such that
      the non-terminals in $\beta$ are already in $N'$.
    If the rule is $A \rightarrow a [_r\, C\, ]$ for $1 \leqslant r \leqslant 2$ then
        Put $A \rightarrow a [_r\, C\, ]$ in $R'$.
        Put $A$ in $N'$.
    If the rule is $A \rightarrow a [_2\, C\, ][_1\, D\, ]$ then
        If $C_x$ is not already in $N'$ then
            Form $G_C : \Xi$ with start symbol $C_x$ and add it to $G_B : \Delta$.
        Put $A \rightarrow a [_2\, C_x\, ][_1\, D\, ]$ in $R'$.
        If $C_t$ is not already in $N'$ then
            Form $G_C : (\Sigma \bigcap \Delta)$ with start symbol $C_t$ and add it to $G_B : \Delta$.
        Put $A \rightarrow a [_2\, C_t\, ]$ in $R'$.
        Put $A$ in $N'$.
    Remove $A \rightarrow \beta$ from $R$.
Until $R$ is empty.
Set $S' = B$.
```

The construction can be completed by removing the useless or unreachable non-terminals and rules. The initial steps of the algorithm make useless any non-terminal which does not derive a forest whose frontier ends with an element of Δ. The handling of a rule $A \rightarrow a [_2\, C\,][_1\, D\,]$ is complicated by the possibility that C

may derive a subforest whcih does not have an external selector. In that case the subforest derived from D is truncated during the frontier operation. The addition of $A \rightarrow a[_2 C_t]$ to R' and $G_C:(\Sigma \bigcap \Delta)$ to $G_B:\Delta$ assures that the appropriate subforests of truncating 2-1 structures are included. QED.

Partitioning can be used to isolate subforests which yield strings that end in a terminal instead of a selector. When the frontier is taken, such subforests may cause other subforests to be truncated. The next lemma shows how to eliminate truncating subforests from a 2-d forest grammar.

LEMMA 4-10. Let G be a short-rule grammar $<\Sigma, \Xi, N, R, S>_2^k$. There is a grammar $G' = <\Sigma', \Xi', N', R', S'>_2^k$ such that every rule in R' is either 2DNF or $A \rightarrow B$ where $A, B \in N'$ and $Y_0(G) = Y_0(G')$.

PROOF. To prove this lemma we describe the construction of G' and then prove two propositions to get the desired result. G' is constructed from G in six steps. Assume $A, B, C \in N$ and $a \in \Sigma$.

1) Set $\Sigma' = \Sigma$, $\Xi' = \Xi \bigcup \{<1, \lambda>\}$, $R' = \{\}$, and $N' = \{\}$.

2) If R has a rule $A \rightarrow x[_1 B]$ or $A \rightarrow x$ where $x \in \Xi_2$, then don't add a corresponding rule to R'.

3) Find rules in R of the form $A \rightarrow a[_2 B]$. Add A' and B' to N'. If $A = B$ or $A \neq S$, then put $A' \rightarrow B'$ in R'. Otherwise put $A' \rightarrow a[_2 B']$ in R'.

4) Find rules in R of the form $A \rightarrow a[_2 B][_1 C]$. Form $G_B:\Xi$ with start symbol B' and add it to G'. Form $G_B:\Sigma$ with start symbol B'' and add it to G'. Add $A' \rightarrow a[_2 B'][_1 C']$ and $A' \rightarrow B''$ to R', and add A' and C' to N'.

5) Find rules in R of the form $A \rightarrow a\,[_1\, B\,]$ or $A \rightarrow x$ for $x \in \Xi_1$. Add corresponding rules $A' \rightarrow a\,[_1\, B'\,]$ or $A' \rightarrow x$ to R', and add A' and B' to N'.

6) Find rules in R of the form $A \rightarrow a$. Invent a new non-terminal T, put $A' \rightarrow a\,[_1\, T\,]$ and $T \rightarrow x$ in R' where $x = <1, \lambda>$ and put A' and T in N'.

After application of these steps, all the right-hand sides of rules in R' are single non-terminals or they fit 2DNF. If propositions (A) and (B) below can be shown to be true, then the lemma is proven.

(A) If there is A in N such that $A \Rightarrow^* \alpha$ and $fr_1(\alpha) = \sigma$, then there is a corresponding non-terminal A' in N' such that $A' \Rightarrow^* \beta$ and $fr_1(\beta) = \sigma$ if α has an external selector, or $fr_1(\beta) = \sigma \cdot x$; where $x = <1, \lambda>$, if α has no external selector.

Proof of (A) by induction on $|\alpha|$, the number of nodes in α.

Base. $|\alpha| = 1$.

For G to derive a structure of one node, one of the following cases must be true.

Case 1. R contains a rule $S \rightarrow x$ where $x \in \Xi_1$.

In this case, $\alpha = x$ cannot be in Ξ_2, or $fr_1(\alpha)$ would not be defined. By step 5 in the construction, $S' \rightarrow x$ is in R', so (A) is satisfied.

Case 2. R contains a rule $S \rightarrow a$ where $a \in \Sigma$.

In this case, construction step 6 adds to R' the rules $S \rightarrow a\,[_1\, T\,]$ and $T \rightarrow x$ where $x = <1, \lambda>$. α is just a, and β is $a\,[_1\, x\,]$. So $fr_1(\alpha) = a$ and $fr_1(\beta) = a\,[_1\, x\,]$. Since α has no external selector, (A) is satisfied.

Inductive hypothesis. Assume (A) is true for $1 \leqslant |\alpha| < n$.

Inductive step. Show (A) is true for $|\alpha| = n$.

Case 1. $\alpha = a\,[_2\,\gamma]$ and R has a rule $A \rightarrow a\,[_2\,B\,]$ where $B \Rightarrow^* \gamma$.

According to step 3 of the construction, $A' \rightarrow a\,[_2\,B'\,]$ *or* $A' \rightarrow B'$ was added to R'. We can apply the inductive hypothesis to establish that $B \Rightarrow^* \gamma$, $B' \Rightarrow^* \delta$, and $fr_1(\gamma) = fr_1(\delta)$ or $fr_1(\gamma) \cdot x = fr_1(\delta)$ where $x = <1, \lambda>$. Since $fr_1(a\,[_2\,\gamma]) = fr_1(\gamma)$, or $fr_1(a\,[_2\,\gamma] \cdot x) = fr_1(\gamma) \cdot x$ and $fr_1(a\,[_2\,\delta]) = fr_1(\delta)$, we have satisfied (A) above.

Case 2. $\alpha = a\,[_2\,\gamma][_1\,\delta]$ and R has $A \rightarrow a\,[_2\,B\,][_1\,C\,]$.

By construction step 4, R' has a rule $A' \rightarrow a\,[_2\,B'\,][_1\,C'\,]$ *or* $A' \rightarrow B''$. First let us assume that γ has an external selector. Since both γ and δ are smaller than α, we can apply the inductive hypothesis to get μ and ρ such that $B' \Rightarrow^* \mu$ and $C' \Rightarrow^* \rho$, $fr_1(\gamma) = fr_1(\mu)$ and $fr_1(\delta) = fr_1(\rho)$, or $fr_1(\delta) \cdot x = fr_1(\rho)$ where $x = <1, \lambda>$. It follows that $fr_1(a\,[_2\,\mu][_1\,\rho]) = fr_1(a\,[_2\,\gamma][_1\,\delta])$ or $fr_1(a\,[_2\,\gamma][_1\,\delta]) \cdot x$ when δ has no external selector, and so (A) is satisfied.

Now assume that γ has no external selector. Since $B \Rightarrow^* \gamma$ and γ is smaller that α we can apply the inductive hypothesis to get $B'' \Rightarrow^* \mu$ and $fr_1(\gamma) \cdot x = fr_1(\mu)$. It follows that $fr_1(a\,[_2\,\gamma][_1\,\delta]) \cdot x = fr_1(a\,[_2\,\gamma]) \cdot x = fr_1(a\,[_2\,\mu])$, thus satisfying (A).

Case 3. $\alpha = a\,[_1\,\gamma]$ and R has $A \rightarrow a\,[_1\,B\,]$.

In this case, step 5 adds to R' the rule $A' \rightarrow a\,[_1\,B'\,]$. Since γ is smaller than α, we apply the inductive hypothesis to get $B' \Rightarrow^* \delta$ and $fr_1(\gamma) = fr_1(\delta)$ or $fr_1(\gamma) \cdot x = fr_1(\delta)$ where $x = <1, \lambda>$. It follows that $fr_1(a\,[_1\,\delta]) = fr_1(a\,[_1\,\gamma])$ or $fr_1(a\,[_1\,\gamma]) \cdot x$, thus satisfying (A).

(B) If there is A' in N' such that $A' \Rightarrow^* \beta$ and $fr_1(\beta) = \sigma \cdot x$ where $x = <1, \lambda>$, then there is a corresponding non-terminal A in N such that $A \Rightarrow^* \alpha$ and $fr_1(\alpha) = \sigma \cdot x$ if α has an external selector or $fr_1(\alpha) = \sigma$ if α has no external selector.

Proof of (B). The proof of (B) is similar to the proof of (A). QED.

Finally, it can be shown that every 2-d forest grammar has a yield-equivalent 2DNF grammar.

THEOREM 4-11. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$. There is a 2DNF grammar $G' = <\Sigma', \Xi', N', R', S'>_2^k$ such that $Y_0(G) = Y_0(G')$.

PROOF. Form a grammar G'' by applying the constructions of lemmas 4-6 and 4-10 to G. The lemmas guarantee that $Y_0(G'') = Y_0(G)$. G'' can be converted to 2DNF grammar G' by eliminating the productions of the form $A \rightarrow B$ where both A and B are non-terminals. This can be done by replacing $A \rightarrow B$ with $A \rightarrow \beta$ for each rule $B \rightarrow \beta$, and the yield of the grammar is not affected. Unreachable non-terminals and rules can also be removed. Since $Y_0(G) = Y_0(G'')$ and $Y_0(G'') = Y_0(G')$, we have $Y_0(G) = Y_0(G')$. QED.

The Deleting Power of Extended Selectors

Using extended selectors, we can write 2-d forest grammars in which the selector paths overlap one another. If this happens in a cycle of non-terminals, then the grammar contains a deletion cycle. Consider the following example grammar, which will be named ABC. Let $N_2 = \Phi$, $N_1 = \{S, A\}$, $\Xi = \{x, y\}$ where $x = <1, 1>$ and $y = <1, \lambda>$, $\Sigma = \{\#, a, b, c\}$, and R consists of 4 rules:

1) $S \rightarrow \#[_2\ \#[_2\ x\ ][_1\ S\ ]][_1\ c\ [_1\ y\ ]]$,

2) $S \rightarrow A$,

3) $A \rightarrow \#[_2\ a\ [_1\ y\ ]][_1\ \#[_2\ A\ ][_1\ b\ [_1\ y\ ]]]$, and

4) $A \rightarrow a\ [_1\ b\ [_1\ y\ ]]$.

This grammar yields the string language $\{a^{n-m}\ b^n\ c^m\} \bigcup \{b^{n-m}\ c^{n+m}\}$ for $n > 0$ and $0 < m \leqslant n$. Rules 3 and 4 constitute a context-free subgrammar which yields $\{a^n\ b^n\}$, $n > 0$. Each application of rule 1 attaches a selector to the front of the derived structure and the terminal c to the end. During the frontier operation, a c is attached to the end of the resulting string every time a terminal is removed from the front. Figure 3 contains a forest derived from grammar ABC and its 1-d frontier.

In a forest produced by grammar ABC, a subtree may be repeatedly subjected to deletion during the frontier operation. This happens because the forest contains overlapping selectors. In order to define overlapping selectors formally, we will first introduce external selectors and a forest truncation function. Truncation, external selectors, and overlapping selectors are defined for n-dimensional structures so that the definitions will be useful in later chapters.

DEFINITION 4-12. Let β be a forest in $H_n^k(\Sigma, \Xi)$. x is an *external selector* of β if and only if $x \in \Xi_{n-1}$ and x is in $fr_{n-1}(\beta)$.

```
#---2-------------------------------------------1
    |                                           |
    #---2---1                                   c---1
        |   |                                       |
        x   #---2-------------------------------1   y
                |                               |
                #---2---1                       c---1
                    |   |                           |
                    x   #---2-------------------1   y
                            |                   |
                            #---2---1           c---1
                                |   |               |
                                x   #---2-------1   y
                                        |       |
                                        a---1   #---2-------1
                                            |       |       |
                                            y       a---1   b---1
                                                        |       |
                                                        b---1   y
                                                            |
                                                            y
```

forest derived from grammar ABC

```
b---1
    |
    c---1
        |
        c---1
            |
            c---1
                |
                y
```

its 1-d frontier

FIGURE 3. Example 2-d forest containing overlapping selectors

If x is an external selector of an n-dimensional forest β, then the selector x will not be applied when $fr_1(\beta)$ is taken. If any subforest of β has no external selector, then neither does β. A 2-d forest β has at most one external selector, and it will occur at the end of the string resulting from $fr_1(\beta)$. If a 2-d forest has an external selector, it can be found by starting at the root and traversing the arcs, choosing 1-arcs instead of 2-arcs if there is a choice. For the rest of chapter 4, we will adopt the notational convention that x^p, $p > 0$, is an abbreviation for the selector $<1, 1^p>$, and x^0 represents the selector $<1, \lambda>$.

DEFINITION 4-13. If G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ such that $A \in N$, then *exsel* (A) is the set of all external selectors of forests derived from A.

Since *exsel* $(A) \subseteq \Xi$ for any grammar, it is clear that *exsel* (A) is finite. A procedure for constructing *exsel* (A) is given in the proof of the next theorem.

THEOREM 4-14. If G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ such that $A \in N$, then there is an effective procedure for constructing *exsel* (A).

PROOF. The set *exsel* (A) can be constructed by deriving a finite subset of $L(A)$ called *minset* (A), taking the frontiers of the derived forests, and examining the selectors at the ends of the resulting strings. Put β in *minset* (A) if $A \Rightarrow^* \beta$ and no rule number appears more than once on any path from the root to the leaves of the derivation tree for β. To construct a derivation tree, first number the rules of the grammar. Whenever a rule is applied in the derivation of a forest, label the node which was expanded with the number of the applied rule. Since there are a finite number of rules and a maximum number of non-terminals introduced in applying a rule, the set *minset* (A) is finite.

To prove the theorem, we need to show that if β is derived by repeating a rule, then there is a smaller forest β' which has the same external selector. Suppose $A \Rightarrow^* \beta$ and β contains β' such that $A \Rightarrow^* \beta'$, and the same rule is applied first in both derivations.

Case 1. If β' has no external selector, then neither does β. So β and β' have the same external selector.

Case 2. If β' has an external selector, then it will be replaced during the frontier operation by a string whose external selector is that of β. So we can replace β' with any subforest that has an external selector without affecting the external selector of β. We can choose a subforest to replace β' which is derived without a previously used rule.

The process described above can be applied until all repeated rules are eliminated and the resulting forest has the same external selector as the original. Since the forest has no repeated rules, it will be in *minset* (A). QED.

DEFINITION 4-15. The truncation function *trunc* (β): $H_n^k(\Sigma, \Xi) \rightarrow H_n^n(\Sigma, \Xi)$ is defined as

$trunc(a) = a$ for $a \in \Sigma \bigcup \Xi$,

$trunc(a[_m\ \alpha]) = a[_m\ trunc(\alpha)]$
for $a \in \Sigma$, $\alpha \in H_n^{n-1}(\Sigma, \Xi)$ and $m \leqslant n$,

$trunc(a[_m\ \alpha][_r\ \beta]) = a[_m\ trunc(\alpha)][_r\ trunc(\beta)]$
for $a \in \Sigma$, $\alpha \in H_n^{m-1}(\Sigma, \Xi)$, $\beta \in H_n^{r-1}(\Sigma, \Xi)$, $m \leqslant n$, $r \leqslant n-1$, and

$trunc(a[_n\ \alpha][_{n-1}\ \beta]) = a[_n\ trunc(\alpha)]$
for $a \in \Sigma$, $\alpha \in H_n^{n-1}(\Sigma, \Xi)$, and $\beta \in H_n^{n-2}(\Sigma, \Xi)$.

For a 2-d forest, the truncation function removes subforests joined to a node by a 1-arc only when the node also has a 2-arc. Now truncation and external selectors can be used to define overlapping selectors.

DEFINITION 4-16. A forest $\beta \in H_n^k(\Sigma, \Xi)$ contains an *overlapping selector* if either

1) some subforest of β contains an overlapping selector, or

2) $\beta = a[_n\ \alpha][_{n-1}\ \gamma]$ and $sel(<n-1,\ \sigma>,\ fr_{n-1}(trunc(\gamma))) = x$ where $x \in \Xi_{n-1}$ and $<n-1,\ \sigma\pi>$ is an external selector of α for $\pi \neq \lambda$, $a \in \Sigma$, $\alpha \in H_n^{n-1}(\Sigma, \Xi)$, and $\gamma \in H_n^{n-2}(\Sigma, \Xi)$.

Overlapping selectors are impossible in n-d forests without extended paths. The selection operation always retrieves an n-d tree, and every path of an n-d tree starts with n. Since an n-d standard selector has a path in $\{1, \ldots, n-1\}^*$, the path of one n-d selector cannot penetrate into a tree selected by another n-d selector. The increased deleting power of extended selectors will be analyzed step-by-step, beginning with selectors that do not overlap.

Grammars with Non-Overlapping Selectors

As shown in the proof of theorem 4-3, a 2-d forest grammar in which every selector is x^0 can be converted directly to a context-free grammar. A similar scheme can be used for 2DNF grammars with extended selectors, but no overlapping selectors. The conversion will require the use of the *debts* of non-terminals and partitioning of forest grammars by external selectors. If a non-terminal derives a forest α and has debt i, then i characters will be deleted from the yield of α when the frontier operation is performed on a larger forest which contains α.

DEFINITION 4-17. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ with non-terminal $A \in N$. *debts* (A) is the set of integers such that $i \in$ *debts* (A) if and only if $A \Rightarrow^* \alpha$, $S \Rightarrow^* \beta$, α is a substructure of β, $sfr(\beta) = uvw$ where v is a suffix of the yield of α, and $sfr(\alpha) = yv$, where $|y| = i$.

LEMMA 4-18. Given a grammar $G = <\Sigma, \Xi, N, R, S>_2^k$ with no deletion cycles, there is an effective procedure for constructing *debts* (A) for every non-terminal $A \in N$.

PROOF. The construction of *debts* (A) requires prior calculation of the debts of all the non-terminals which can precede A in a derivation. The algorithm below constructs all the *debts* sets of a grammar by making multiple passes through the rule set R.

```
Set $debts(S) = \{0\}$.
Set $debts(A) = \{\}$ for all other non-terminals.
Repeat
    For each rule $A \rightarrow \beta$ in $R$
        If $\beta = a[_1 B]$ then
            If $0 \in debts(A)$ put 0 in $debts(B)$.
            If $r > 0$ and $r \in debts(A)$ put $r-1$ in $debts(B)$.
            If $\beta = a[_2 B]$ then
                Add every element of $debts(A)$ to $debts(B)$.
            If $\beta = a[_2 B][_1 C]$ then
                Add every element of $debts(A)$ to $debts(B)$.
                For each $m \in exsel(B)$ and $r \in debts(A)$
        If $B:m$ yields a string of length $l \geqslant r$ then
            Put $m$ in $debts(C)$.
        For each $l$ such that $1 \leqslant l < m$
            If $B:m$ yields a string of length $l$ then
                Put $m+r-l$ in $debts(C)$.
    Until no change is made in any debts set.
```

It can be established that the construction algorithm produces the right debts by proving the two propositions below.

(A) If $A \in N$, $A \Rightarrow^* \alpha$, $S \Rightarrow^* \beta$, α is a substructure of β, $sfr(\alpha) = yv$, $sfr(\beta) = uvw$, and $|y| = i$, then $i \in debts(A)$.

(B) If $i \in debts(A)$, then $A \Rightarrow^* \alpha$, $S \Rightarrow^* \beta$, α is a substructure of β, $sfr(\alpha) = yv$, $sfr(\beta) = uvw$, and $|y| = i$.

These can be proven by induction on the difference in size between α and β, $|\beta| - |\alpha|$. The inductions are not shown here. QED.

In the case of a rule $A \rightarrow a[_2 B][_1 C]$ when calculating *debts*(A), if the grammar has no overlapping selectors, then no word which B yields is smaller than any debt of A. If the grammar contains overlapping selectors, then the lengths of strings yielded from B must be determined. A length-predicting scheme will be presented later. First we demonstrate that 2-d grammars which have no overlapping selectors can be converted to context-free grammars.

THEOREM 4-19. Given a 2DNF grammar $G = <\Sigma, \Xi, N, R, S>_2^k$ with no overlapping selectors, there is a context-free grammar G' such that $Y_0(G) = L(G')$.

PROOF. To construct G', follow the algorithm below. Assume that $A, B, C \in N$, $a \in \Sigma$, and $x^r \in \Xi_1$.

```
Set $\Sigma' = \Sigma$, $R' = \{\}$, and $N' = \{S'\}$.
For each $x^r \in exsel(S)$
   Put ${}_0S_r$ in $N'$.
   Put $S' \to {}_0S_r$ in $R'$.
Repeat
   Take a non-terminal ${}_jA_r$ from $N'$ for which there are
      no rules in $R'$.
   For each rule $A \to \beta$ in $R$
L1:    If $\beta = x^r$ and $j = 0$, put ${}_0A_r \to \lambda$ in $R'$.
L2:    If $\beta = a[_1 B]$ then
          If $j = 0$ add ${}_0A_r \to a\ {}_0B_r$ to $R'$ and ${}_0B_r$ to $N'$.
          Else add ${}_jA_r \to {}_{j-1}B_r$ to $R'$ and ${}_{j-1}B_r$ to $N'$.
L3:    If $\beta = a[_2 B]$ add ${}_jA_r \to {}_jB_r$ to $R'$ and ${}_jB_r$ to $N'$.
L4:    If $\beta = a[_2 B][_1 C]$ then
          Add ${}_jA_r \to {}_jB_m\ {}_mC_r$ to $R'$ and ${}_jB_m$, ${}_mC_r$ to $N'$.
             for each $x^m \in exsel(B)$.
Until all non-terminals in $N'$ have rules in $R'$.
```

The theorem is true if propositions (A) and (B) below can be proven.

(A) If $A \in N$, $A \Rightarrow^* \beta$, $sfr(\beta) = vw$, $|vw| = j$, $j \in debts(A)$, and $x^r \in exsel(A)$ then ${}_jA_r \in N'$ and ${}_jA_r \Rightarrow^* w$.

Proof of (A) by induction on $|\beta|$.

Base. $|\beta| = 1$.

In this case, R has a rule $A \rightarrow x^r$. Since G has no overlapping selectors, the debt of a is 0. Also, the external selector of x^r is x^r, $fr_1(x^r) = x^r$, and $str(x^r) = \lambda$. By line L1 of the construction algorithm, R' contains ${}_0A_r \rightarrow \lambda$. So (A) is satisfied.

Inductive hypothesis. Assume (A) is true for $1 < |\beta| < n$.

Inductive step. Show (A) is true for $|\beta| = n$.

Case 1. R has a rule $A \rightarrow a[_1 B]$ and $\beta = a[_1 \gamma]$ where $B \Rightarrow^* \gamma$.

We know A has external selector x^r and debt j. First, suppose $j = 0$. Then $sfr(\beta) = aw'$. We can apply the inductive hypothesis to γ with debt 0 and external selector x^r to get ${}_0B_r$ in R' such that ${}_0B_r \Rightarrow^* w'$. By line L2, we also have ${}_0A_r \rightarrow a\,{}_0B_m$ in R'. So ${}_0A_r \Rightarrow^* aw'$, and (A) is satisfied. Now suppose $j > 0$. Then $sfr(\beta) = sfr(a[_1 \gamma]) = av'w = vw$, where γ yields $v'w$, and B has external selector r and debt $j-1$. The inductive hypothesis gives us ${}_{j-1}B_r \Rightarrow^* w$ for ${}_{j-1}B_r$ in N' and line L2 puts ${}_jA_r \rightarrow {}_{j-1}B_r$ in R'. So ${}_jA_r \Rightarrow^* w$, and (A) is satisfied.

Case 2. R has $A \rightarrow a[_2 B]$ and $\beta = a[_2 \gamma]$ where $B \Rightarrow^* \gamma$.

B has the same debt j as A, and γ has the same external selector x^r and frontier as β. So $sfr(\beta) = sfr(\gamma) = vw$. Since γ is smaller that β, we apply the inductive hypothesis to get ${}_jB_r$ in N' such that ${}_jB_r \Rightarrow^* w$. Line L3 gives us the rule ${}_jA_r \rightarrow {}_jB_r$ in R', so ${}_jA_r \Rightarrow^* w$, satisfying (A).

Case 3. R has $A \rightarrow a\,[_2 B\,][_1 C\,]$, $\beta = a\,[_2\gamma][_1\delta]$, $B \Rightarrow^* \gamma$, and $C \Rightarrow^* \delta$.

Since A has debt j and external selector x^r, B has debt j and δ has external selector x^r. B has some other external selector x^m and the debt of C is therefore m, since selectors cannot overlap. We know that $sfr(\beta) = vw$, $|v| = j$. This means that $sfr(\gamma) = vw_1$ and $sfr(\delta) = uw_2$, where $|u| = m$ and $w = w_1w_2$. All of v is derived from B since selectors cannot overlap. γ and δ are smaller than β, so we apply the inductive hypothesis to get ${}_jB_m$ and ${}_mC_r$ in N' such that ${}_jB_m \Rightarrow^* w_1$ and ${}_mC_r \Rightarrow^* w_2$. Line L4 also gives ${}_jA_r \rightarrow {}_jB_m\ {}_mC_r$ in R', so ${}_jA_r \Rightarrow^* w$ and (A) is satisfied.

(B) If ${}_jA_r \in N'$ and ${}_jA_r \Rightarrow^* w$, then there is A in N such that $A \Rightarrow^* \beta$, $j \in debts(A)$, $x^r \in exsel(A)$, and $sfr(\beta) = vw$ where $|v| = j$.

Proof by induction on d, the number of derivation steps required for w.

Base. $d = 1$.

N' has a rule ${}_0A_r \rightarrow \lambda$. Then N has $A \rightarrow x^r$ by line L1, A has debt 0 and external selector x^r, and $sfr(x^r) = \lambda$. Thus, (B) is satisfied.

Inductive hypothesis. Assume (B) is true for $1 < d < n$.

Inductive step. Show (B) is true for $d = n$.

Case 1. R' has ${}_0A_r \rightarrow a\ {}_0B_r$ and ${}_0A_r \Rightarrow^* aw'$ where ${}_0B_r \Rightarrow^* w'$.

We apply the inductive hypothesis for the derivation of w' to get $B \Rightarrow^* \gamma$,

$sfr(\gamma) = w'$, B has debt 0 and external selector x^r. By line L2, R contains $A \rightarrow a\,[_1 B\,]$, and so $A \Rightarrow^* a\,[_1 \gamma]$. But $sfr(a\,[_1 \gamma]) = str(a\,[_1 fr_1(\gamma)]) = a \cdot sfr(\gamma) = aw'$. A has debt 0 and external selector x^r, so (B) is satisfied.

Case 2. R' has ${}_jA_r \rightarrow {}_{j-1}B_r$ and ${}_jA_r \Rightarrow {}_{j-1}B_r \Rightarrow^* w$.

We apply the inductive hypothesis to get B in N such that $B \Rightarrow^* \gamma$, $sfr(\gamma) = v'w$, $|v'| = j-1 = debt(B)$, and $x^r \in exsel(B)$. By line L2, we have $A \rightarrow a\,[_1 B\,]$ in R where A has external x^r and debt j. So $A \Rightarrow^* a\,[_1 \gamma]$, and $sfr(a\,[_1 \gamma]) = a \cdot sfr(\gamma) = av'w = vw$ where $|v| = j$. Thus, (B) is satisfied.

Case 3. R' has a rule ${}_jA_r \rightarrow {}_jB_r$ and ${}_jA_r \Rightarrow {}_jB_r \Rightarrow^* w$.

We apply the inductive hypothesis to get B in N such that $B \Rightarrow^* \gamma$, $sfr(\gamma) = vw$, $|v| = j = debt(B)$, and $x^r \in exsel(B)$. By line L3, we have $A \rightarrow a\,[_2 B\,]$ in R where A has external selector x^r and debt j. So $A \Rightarrow^* a\,[_2 \gamma]$ and $sfr(a\,[_2 \gamma]) = sfr(\gamma) = vw$. Thus, (B) is satisfied.

Case 4. R' has a rule ${}_jA_r \rightarrow {}_jB_m\,{}_mC_r$, $w = w_1w_2$, ${}_jB_m \Rightarrow^* w_1$ and ${}_mC_r \Rightarrow^* w_2$.

We apply the inductive hypothesis to get B and C in N such that $B \Rightarrow^* \gamma$, $C \Rightarrow^* \delta$, $sfr(\gamma) = v_1w_1$, $|v_1| = j$, $x^m \in exsel(B)$, $sfr(\delta) = v_2w_2$, $|v_2| = m$, and $x^r \in exsel(C)$. From line L4, we have $A \rightarrow a\,[_2 B\,][_1 C\,]$ in R where A has debt j and external selector x^r. So $sfr(a\,[_2 \gamma][_1 \delta]) = str(subs_1(fr_1(\gamma), fr_1(\delta))) = v_1w_1w_2 = v_1w$ where $|v_1| = j$. Thus, (B) is satisfied. QED.

Grammars Without Deletion Cycles

2-d forest grammars are harder to analyze when they contain overlapping selectors, particularly when there's a cycle of overlapping selectors called a deletion cycle. To define a deletion cycle, it will be necessary to distinguish the substructures of a forest which precede a given node from those which follow it. The deletion cycle definition also contains some terminology which can be informally defined as follows: a 1-node in a forest is a node which has only a 1-arc enamating from it, a 2-1-node has both a 1-arc and a 2-arc, and a 2-node has only a 2-arc.

DEFINITION 4-20. Suppose α and β are subforests of $\gamma \epsilon H_n^k(\Sigma, \Xi)$ such that $sel(<n, p>, \gamma) = \alpha$, and $sel(<n, q>, \gamma) = \beta$. Let $p = \sigma i \pi$ and $q = \sigma j \rho$ where $i \neq j$. α precedes β if $i > j$, and α follows β if $i < j$.

DEFINITION 4-21. Suppose a grammar $G = <\Sigma, \Xi, N, R, S>_2^k$ has a non-terminal M such that $M \Rightarrow^* \gamma$, β is a subforest of γ, and $M \Rightarrow^* \beta$. If the sum of the lengths of the selector paths exceeds the number of 1-nodes over all the subtrees that precede β in γ, then G has a *deletion cycle*.

So a deletion cycle deletes more than it adds when the frontier is taken. If we try to find the debts of non-terminals in a grammar with a deletion cycle, the algorithm of lemma 4-18 will never halt. If there is no deletion cycle, the algorithm will halt and a maximum debt for any non-terminal can be identified. If a grammar has overlapping selectors but no deletion cycles, it can be converted to context-free by a procedure similar to that of theorem 4-19. In order to calculate the debts of the non-terminals, however, it is necessary to predict the lengths of words yielded from certain non-terminals.

LEMMA 4-22. Let G be a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ which is partitioned according to external selectors, has no deletion cycles, and has non-terminal A. There is an effective procedure for determining whether $Y_0(L(A))$ contains a string whose length is l, for arbitrary $l \geqslant 0$.

PROOF. The algorithm is presented as a recursive function *wordsize* which has three parameters: a non-terminal A, and integer length l, and a set *callset* of (non-terminal, integer) pairs which is used to prevent repetition of useless function calls. The function answers *yes* if A yields a word of length l, and *no* otherwise.

```
wordsize $(A, l, callset)$:
    If $(A, l) \in callset$ then set answer to no.
    Else
        Set answer to no.
        Repeat
            Select a rule $A \mapsto \beta$ in $R$.
            If $\beta = a[_1 B]$ then
                If $l > 0$ then
                    Set answer to wordsize$(B, l-1, callset \bigcup \{(A, l)\})$.
            If $\beta = a[_2 B]$ then
                Set answer to wordsize$(B, l, callset \bigcup \{(A, l)\})$.
            If $\beta = x^r$ then
                If $l = 0$ then
                    Set answer to yes.
            If $\beta = a[_2 B][_1 C]$ where $exsel(B) = \{m\}$ then
        Set $i$ to 0.
        Repeat
            If $wordsize(B, i, callset \bigcup \{(A, l)\})$ then
                Set $answer$ to $wordsize(C, l+m-1, callset \bigcup \{(A, l)\})$.
            Add 1 to $i$.
        Until $i = l$ or $answer = yes$.
    Until all rules for $A$ have been examined.
Return the value of $answer$.
```

The *wordsize* function answers *yes* when l is 0 and it finds a production $A \rightarrow x^r$. Otherwise, it calls itself with the appropriate non-terminals and new value of l. When it finds a rule $A \rightarrow a[_2 B][_1 C]$, it makes calls using all possible sublengths of l with B and C. In some instances, the value of l in the new function call is larger than that which was passed in. If the grammar were to contain a deletion cycle, the function would call itself indefinitely. But since the grammar does not contain a deletion cycle, the number of symbols added (the sum of the values of i) exceeds the number of symbols deleted (the sum of the values of m) for each pass through a cycle. When the function calls itself at the beginning of each new pass through a cycle, the value of l is smaller than before.

The function always adds the current values of A and l to the *callset* when it makes new calls. If the grammar has a cycle which neither increases nor decreases the size of the yielded string, the function will not traverse the cycle more than once. Having traversed such a cycle once, it is useless to continue. Thus, execution of the function eventually halts with an answer of *yes* or *no*.

QED.

Now it is possible to demonstrate that any 2-d forest grammar which does not have a deletion cycle can be converted to context-free. The conversion process is very similar to that used for grammars without overlapping selectors.

THEOREM 4-23. Let G be a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$. If G does not contain a deletion cycle, then there is a context-free grammar G' such that $L(G') = Y_0(G)$.

PROOF. To construct G', follow the algorithm below. Assume that $A, B, C \in N$, $a \in \Sigma$, and $x^r \in \Xi_1$.

```
    Set $\Sigma' = \Sigma$, $R' = \{\}$, and $N' = \{S'\}$.
    For each $x^r \in exsel(S)$
        Put ${}_0S_r$ in $N'$.
        Put $S' \rightarrow {}_0S_r$ in $R'$.
    Repeat
        Take a non-terminal ${}_jA_r$ from $N'$ for which there are
            no rules in $R'$.
        For each rule $A \rightarrow \beta$ in $R$
    L1:     If $\beta = x^r$ and $j = 0$, put ${}_0A_r \rightarrow \lambda$ in $R'$.
    L2:     If $\beta = a[_1 B]$ then
                If $j = 0$ add ${}_0A_r \rightarrow a\ {}_0B_r$ to $R'$ and ${}_0B_r$ to $N'$.
                Else add ${}_jA_r \rightarrow {}_{j-1}B_r$ to $R'$ and ${}_{j-1}B_r$ to $N'$.
```

L3: If $\beta = a[_2 B]$ add ${}_j A_r \rightarrow {}_j B_r$ to R' and ${}_j B_r$ to N'.

L4: If $\beta = a[_2 B][_1 C]$ then

Add ${}_j A_r \rightarrow {}_j B_m\ {}_m C_r$ to R' and ${}_j B_m, {}_m C_r$ to N'.

for each $m \in exsel(B)$.

Add ${}_j A_r \rightarrow {}_{m+j-l} C_r$ to R' and ${}_{m+j-l} C_r$ to N'

for each $m \in exsel(B)$ and $l \in paths(B:m)$ such that $l < j$

and $B:m$ yields a string of length l.

Until all non-terminals in N' have rules in R'.

The theorem is established by proving the propositions (A) and (B) of theorem 4-19.

Proof of (A). This proof is the same as in theorem 4-19 with the following addition:

Case 3. R has $A \rightarrow a[_2 B][_1 C]$, $\beta = a[_2 \gamma][_1 \delta]$, $B \Rightarrow^* \gamma$ and $C \Rightarrow^* \delta$.

We know that A has debt j and external selector x^r, and $sfr(\beta) = vw$ where $|v| = j$. Suppose that $sfr(\gamma)$ has length l less than j and assume that B has external selector x^m. Then C has debt $m+j-l$ and external selector x^r. We apply the inductive hypothesis to establish that ${}_{m+j-l} C_r \Rightarrow^* w$ for ${}_{m+j-l} C_r$ in R'. Line L4 also gives us ${}_j A_r \rightarrow {}_{m+j-l} C_r$ in R'. So ${}_j A_r \Rightarrow^* w$ and (A) is satisfied.

Proof of (B). This proof is the same as in theorem 4-19 with the following case added:

Case 5. R' has ${}_j A_r \rightarrow {}_{m+j-l} C_r$ and ${}_j A_r \Rightarrow^* w$.

${}_{m+j-l}C_r$ derives w in one less step than ${}_jA_r$, so we apply the inductive hypothesis to get $C \Rightarrow^* \delta$ and $sfr(\delta) = vw$ for C in N with debt $m+j-l$ and external selector x^r, and $|v| = m+j-l$. By line L4, we also know that R has a rule $A \rightarrow a[_2 B][_1 C]$ and $B \Rightarrow^* \gamma$ such that $|sfr(\gamma)| = l$ and γ has external selector m. So $sfr(a[_2\gamma][_1\delta]) = v_1v_2w$ where $|v_1| = l$ and $|v_2| = |v|-m = j-l$. But then $|v_1v_2| = j$, satisfying (B). QED.

Theorems 4-19 and 4-23 show that any forest 2-d grammar that does not have a deletion cycle can be converted to a context-free grammar. It follows that any 2-d forest grammar that does not have a deletion cycle can be rewritten using only standard selectors. Just convert the grammar to context-free, and then convert it back to a 2DNF grammar whose only selector is x^0.

Length Predictors for 2-d Forest Grammars

Before deletion cycles are analyzed, it will be demonstrated that a 2-d forest grammar can be converted to a system which predicts the lengths of the strings yielded by the grammar. Given a grammar and a non-terminal A, the set *paths*(A) contains the lengths of all the strings which A yields.

DEFINITION 4-24. If G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ and $A \in N$, then *paths*(A) is the set of integers such that $i \in$ *paths*(A) if and only if $A \Rightarrow^* \beta$ and $|sfr(\beta)| = i$.

The rewriting rules for a non-terminal A implicitly contain rules for the construction of *paths*(A). The set of expressions for constructing *paths*(A) will be denoted $\hat{A}$. The rules will contain base values and will specify addition and subtraction operations on values generated from other sets of expressions.

LEMMA 4-25. Suppose G is a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ and $A \in N$.

There is an effective procedure for constructing $\hat{A}$, a set of expressions which specify *paths* (A).

PROOF. To construct $\hat{A}$, first form $G' = <\Sigma', \Xi', N', R', S'>_2^k$ such that the derived forests are partitioned according to their external selectors. If N has non-terminal A, then N' has $A_{r_1}, A_{r_2}, \ldots, A_{r_m}$ for each r_i in *exsel* (A), $1 \leqslant i \leqslant m$, and A_{r_i} represents $A:\{x^{r_i}\}$. The set *paths* (A) will be the union for $i = 1$ to m of *paths* (A_{r_i}). To form $\hat{A}_j$ where $j = r_i$, find all the rules for non-terminal A_j in G' and follow these steps:

1) If N' has $A_j \rightarrow x^j$, put 0 in $\hat{A}_j$.

2) If N' has $A_j \rightarrow a[_1 B_j]$, put the expression $\hat{B}_j + 1$ in $\hat{A}_j$.

3) If N' has $A_j \rightarrow a[_2 B_j]$, put the expression $\hat{B}_j$ in $\hat{A}_j$.

4) If N' has $A_j \rightarrow a[_2 B_r][_1 C_j]$, put the expression $\hat{B}_r + \hat{C}_j \dot{-} r$ in $\hat{A}_j$.

Complete the construction by forming the rule set $\hat{B}_m$ for each B_m such that $\hat{B}_m$ is used in an expression of $\hat{A}_j$. An expression $\hat{B}_m + 1$ in $\hat{A}_j$ means that $b+1$ belongs to *paths* (A_j) for every b in *paths* (B_m). An expression $\hat{C}_r + \hat{B}_m \dot{-} r$ means that $c+b\dot{-}r$ belongs to *paths* (A_j) for every b in *paths* (B_m) and c in *paths* (C_r). The operator $\dot{-}$ is a subtraction operation which is defined for $b\dot{-}r$ only if $b \geqslant r$. Negative string lengths are not meaningful. We will adopt the convention that the $\dot{-}$ operator applies only to the term that immediately precedes it. Thus, $8\dot{-}6$ and

$(4+4)\dot{-}6$ are meaningful, but $4+4\dot{-}6$ is not defined.

It can be shown by induction on the number of steps required to derive β that $\hat{A}$ will put $|sfr(\beta)|$ in *paths*(A) if and only if $A \Rightarrow^* \beta$. The restricted subtraction operator assures that no elements will be added for forests which contain path errors. QED.

The length-predicting scheme described in the previous lemma builds sets of integers with repeated addition and subtraction on a finite number of base elements. Some operations on the length-predicting expressions can be performed which will eventually eliminate subtraction from all the expressions. A substitution operation will allow some initial simplification of the system. This simple substitution is defined below and justified in the lemma following the definition.

DEFINITION 4-26. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ for which $\hat{A}$ and $\hat{B}$ generate *paths*(A) and *paths*(B) respectively where $A, B \in N$. If $\hat{A}$ has an expression e_i which refers to $\hat{B}$ and $\hat{B}$ consists of expressions $e_1', e_2', \ldots, e_m'$, then the substitution of $\hat{B}$ into e_i is accomplished by replacing e_i with m expressions $e_{i_1}, e_{i_2}, \ldots, e_{i_m}$ in which (e_r') is substituted for $\hat{B}$, $1 \leqslant r \leqslant m$.

LEMMA 4-27. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ for which $\hat{A}$ and $\hat{B}$ generate *paths*(A) and *paths*(B) respectively where $A, B \in N$. If $\hat{A}$ has an expression e_i and $\hat{A}'$ is the same as $\hat{A}$ with $\hat{B}$ substituted into e_i, then $\hat{A}'$ also generates *paths*(A).

PROOF. The textual substitution of subexpressions has no effect on the set generated. The subset of expressions $\{e_{i_1}, e_{i_2}, \ldots, e_{i_m}\}$ in $\hat{A}'$ specifies the same elements for *paths*(A) as the expression e_i from $\hat{A}$ in combination with $\hat{B}$, and no expressions other that e_i are modified. The meaning of a restricted subtraction operator following $\hat{B}$ in e_i is preserved by putting parentheses around the subexpressions which replace $\hat{B}$. QED.

If $\hat{A}$ has expressions which contain a term $\hat{B}$ and *paths*(B) is a finite set, then $\hat{B}$ can be substituted into the expressions of $\hat{A}$. This increased the number of expressions in $\hat{A}$, but it allows some arithmetic operations to be performed and it may eliminate the need for $\hat{B}$. Any subexpressions $b+c$ or $b \dot{-} c$ can be evaluated. If $b<c$ in a subexpression $b \dot{-} c$, then the expression should be removed from the system.

If a grammar contains cycles of non-terminals, then its length-predicting system will contain corresponding cycles. Expressions can refer to one another, and those that refer to themselves, either directly or indirectly, are called self-referential.

DEFINITION 4-28. Suppose G is a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$, $A \in N$, and $\hat{A}$ specifies the set *paths*(A). An expression in $\hat{A}$ refers to $\hat{B}$ (or non-terminal B) if and only if

1) it contains an occurrence of $\hat{B}$, or
2) it contains an occurrence of $\hat{C}$ and an expression in $\hat{C}$ refers to B.

DEFINITION 4-29. Suppose G is a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$, $A \in N$, and $\hat{A}$ specifies the set *paths*(A). An integer b is a base element of $\hat{A}$ if it can be generated without applying any expression more than once and without applying any self-referential expression which contains an arithmetic operator.

Since the number of expressions in a length-predicting scheme for a grammar is finite, the set of base elements of a non-terminal is finite, and it can easily be enumerated. Substitution of base elements can be used to reduce the complexity of expressions which have two terms that are self-referential. This process is called linearization.

LEMMA 4-30. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ for which $\hat{A}$ generates *paths* (A), $A \in N$. There exists $\hat{A}'$ such that $\hat{A}'$ generates *paths* (A) and no expression in $\hat{A}'$ has two terms which refer to A.

PROOF. The construction of $\hat{A}'$ is described for two situations, depending on whether $\hat{A}$ refers to itself directly or indirectly.

1) If $\hat{A}$ has an expression $e_0 : \hat{A} + \hat{A} \dot{-} p$ where p is an integer, put all the expressions of $\hat{A}$ except e_0 in $\hat{A}'$. Suppose $b_1, b_2, \ldots, b_r$ are the base elements of $\hat{A}$. Put the expressions $b_1 + \hat{A}' \dot{-} p$, $b_2 + \hat{A}' \dot{-} p$, . . . , and $b_r + \hat{A}' \dot{-} p$ in $\hat{A}'$ instead of e_0.

2) Suppose $\hat{A}$ has an expression $e_0 : \hat{B} + \hat{C} \dot{-} p$ where p is an integer, an expression in $\hat{B}$ refers to $\hat{A}$, an expression of $\hat{C}$ refers to $\hat{A}$, and $b_1, b_2, \ldots, b_r$ are the base elements of $\hat{A}$. To form $\hat{A}'$, put all the expressions of $\hat{A}$ in $\hat{A}'$ except e_0. Replace references to $\hat{A}$ with $\hat{A}'$ in these expressions. Put expressions $\hat{B}' + \hat{C} \dot{-} p$ and $\hat{B} + \hat{C}' \dot{-} p$ in $\hat{A}'$ instead of e_0. $\hat{B}'$ is formed from $\hat{B}$ as follows: if $\hat{B}$ has an expression f which refers to $\hat{A}$, replace it with expressions $f_1, f_2, \ldots, f_r$, where f_i is the expression f with $\hat{A}$ replaced by b_i, $1 \leqslant i \leqslant r$. $\hat{C}'$ is formed from $\hat{C}$ in a similar manner.

It is clear in each situation that if $\hat{A}'$ generates l, then $\hat{A}$ generates l, since generations in $\hat{A}'$ can be exactly duplicated in $\hat{A}$. It is less obvious that l generated by $\hat{A}$ is also generated by $\hat{A}'$. This can be proven by induction on the number of steps required to generate l. The proof is shown below for situation 1 in the construction rules. The proof for situation 2 is similar. In the proof, the symbol b will represent a base element of $\hat{A}$, and the symbol l will represent an integer which has been generated by a series of steps from one or more base elements.

(A) If $\hat{A}$ generates l, then $\hat{A}'$ generates l.

Proof of (A) where $\hat{A}'$ is constructed as in situation 1 by induction on the number of steps in the generation of l.

Base. $\hat{A}$ generates l in one step.

In this case, $l = b$ is a base element. $\hat{A}'$ has the same base elements as $\hat{A}$, so $\hat{A}'$ also generates l.

Inductive hypothesis. Assume (A) is true when $\hat{A}$ generates l in less than n steps.

Inductive step. Show (A) is true when $\hat{A}$ generates l in n steps.

If l is generated by an expression other than e_0, then $l = l_1 \pm i$ or $l = l_1 + l_2 \dot{-} p$, where i, p are integers and l_1, l_2 are generated by $\hat{A}$. In these cases, we use the inductive hypothesis to extablish that l_1 and l_2 are generated by $\hat{A}'$, and then observe that $\hat{A}'$ has the same expressions as $\hat{A}$ except for e_0. Since e_0 is not used in the last step, we know that $\hat{A}'$ generates l.

If l is generated by e_0, then $l = l_1 + l_2 \dot{-} p$. We can show that l is in $\hat{A}'$ by examining the ways in which l_1 can be generated.

Case 1. $l_1 = b$, a base element of $\hat{A}$.

In this case, the inductive hypothesis can be applied to extablish that l_2 is in $\hat{A}'$. There is also a rule $b + \hat{A}' \dot{-} p$, so $l = b + l_2 \dot{-} p$ is generated in $\hat{A}'$.

Case 2. $l_1 = l_3 \pm i$ for integer i and l_3 in $paths(A)$.

In this case, $\hat{A}$ has a rule $\hat{A} \pm i$. The element l can also be generated by applying the expressions in a slightly different order without causing a path error: $l = l_1 + l_2 \dot{-} p = l_3 \pm i + l_2 \dot{-} p = l_3 + l_2 \dot{-} p \pm i$. Since l_2 and l_3 are both in $paths(A)$, $\hat{A}$ generates $l_3 + l_2 \dot{-} p$, and the inductive hypothesis can be applied to show that $\hat{A}'$ generates $l_3 + l_2 \dot{-} p$. $\hat{A}'$ has a rule $\hat{A}' \pm i$, so $\hat{A}'$ generates l.

Case 3. $l_1 = l_3 + l_4 \dot{-} p$ for integer p and l_3, l_4 in $paths(A)$.

In this case, l_1 is generated by a previous application of e_0. The inductive hypothesis can be applied to establish that l_1 is generated in $\hat{A}'$ as $l_1 = b + l_4' \dot{-} p$. Since every generation in $\hat{A}'$ can be imitated step for step in $\hat{A}$, l_1 can be generated in $\hat{A}$ as $b + l_4' \dot{-} p$ also. Now we have $l = l_1 + l_2 \dot{-} p = (b + l_4' \dot{-} p) + l_2 \dot{-} p$. Observe that $l_4' + l_2 \dot{-} p$ is larger than l_4', so l can also be generated as $b + (l_4' + l_2 \dot{-} p) \dot{-} p$ in $\hat{A}$. The inductive hypothesis can be applied to establish that $l_2' = l_4' + l_2 \dot{-} p$ can also be generated in $\hat{A}'$, and since $\hat{A}'$ has an expression $b + \hat{A}' \dot{-} p$, l is generated in $\hat{A}'$. QED.

Subtraction which is not part of a deletion cycle is easy to eliminate from a length-predicting scheme. To subtract a constant value d from every element of *paths*(A), just subtract d from the base elements of $\hat{A}$. If some of the base elements are smaller than d, then it is necessary to expand the base before subtracting.

LEMMA 4-31. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_2^k$ with non-terminal A and $\hat{A}$ specifies the set *paths*(A). If no expression of $\hat{A}$ uses subtraction and every expression has been linearized, then for any positive number d, there is $\hat{A}'$ which specifies *paths*(A') where *paths*$(A') = \{n \mid l \dot{-} d = n$ and $l \in$ *paths*$(A)\}$.

PROOF. To construct $\hat{A}'$, first place all the expressions of $\hat{A}$ in $\hat{A}'$. For each base element of $\hat{A}'$, apply every possible combination of increasing steps to get a number m such that $d \leqslant m < i+d$ where i is the smallest increment of any step applied in the generation of m. Add each such m to $\hat{A}'$ as a base element. These values are easily enumerated, since every expression is increasing. Remove all base elements from $\hat{A}'$ which are less than d. Subtract d from every base element which is greater than or equal to d.

Now $\hat{A}'$ generates n if and only if $\hat{A}$ generates $n+d$. If $\hat{A}'$ generates n from base element b, then $\hat{A}$ has base element $b+d$ and generates $n+d$ by the same sequence of operations. If $\hat{A}$ generates $n+d$ from base element $b \geqslant d$, then $\hat{A}'$ generates n from base element $b-d$ by the same sequence of operations. If b is less than d, then $\hat{A}'$ generates n in fewer steps from a larger base element $b+i-d$, where i is the smallest sum of initial increments such that $b+i-d \geqslant 0$. QED.

Removal of subtraction in deletion cycles in considerably more complicated, but it is possible because the number of distinct increments and decrements in a length-predicting system is finite. Subtraction is eliminated by enumerating base elements below a certain value and then changing the subtractions to additions. This method is described in lemma 4-33. An auxiliary lemma is presented first which will be needed for the proof of lemma 4-33.

LEMMA 4-32. Suppose $i_1 + i_2 + \cdots + i_p$ is a sum of positive integers where p is larger than some number d. There is a partial sum $i_l + i_{l+1} + \cdots + i_m = n \times d$ such that $l > 1$ or $m < p$, and $n \geqslant 1$.

PROOF. First observe a property involving remainders of integer division: if $a \ mod \ d = b \ mod \ d$ then $(a-b) \ mod \ d = 0$ for positive integers a, b, and d such that $a > b$. In other words, if some number is added to a and the addition does not change the remainder of dividing by d, then the amount added is a multiple of d.

Now consider the sum $i_1 + i_2 + \cdots + i_p$. We can calculate the remainder of dividing each partial sum by d: $(\sum_{j=1}^{t} i_j) \ mod \ d = r_t$ for every value $1 \leqslant t \leqslant p$. Since p is larger than d, there must be a repeated remainder in $r_1, r_2, \ldots, r_d$. This means there are h and m such that $1 \leqslant h < m \leqslant d$ and $(\sum_{j=1}^{h} i_j) \ mod \ d = (\sum_{j=1}^{m} i_j) \ mod \ d$. Applying the property observed in the previous paragraph, $(\sum_{j=h+1}^{m} i_j) \ mod \ d = 0$. So we have $i_{h+1} + i_{h+2} + \cdots + i_m = n \times d$ for some

$n \geqslant 1$. QED.

LEMMA 4-33. Suppose $\hat{A}$ specifies *paths* (A) for non-terminal A of a 2DNF grammar $G = <\Sigma, \Xi, N, R, S>_2^k$. If every expression of $\hat{A}$ has been linearized, one self-referential expression contains a subexpression $\hat{A} \dot{-} p$, and no base element of $\hat{A}$ is smaller than p, then there is a system $\hat{A}'$ which specifies *paths* (A) and has no subtraction operators.

PROOF. If *paths* (A) is a finite set, then its elements can be enumerated in $\hat{A}'$ as base elements, and $\hat{A}'$ will contain neither additions nor subtractions.

If *paths* (A) is an infinite set, then some expressions of $\hat{A}$ specify constant increments, $i_1, i_2, \ldots, i_r$, which can be repeatedly added to get larger elements of *paths* (A). The largest of these increments will be represented as $i_{\max}$. Let $e = \hat{A} \dot{-} p + i$ be the self-referential expression in $\hat{A}$ from which the subtraction operator will be removed. Construct $\hat{A}'$ as follows:

1) Put every expression of $\hat{A}$ in $\hat{A}'$ except e.
2) Add more base elements to $\hat{A}'$ by taking the base elements of $\hat{A}$ and applying all possible sequences of increasing steps which give a total increment less than or equal to $d \times i_{\max}$. This adds a finite number of base elements to $\hat{A}'$, since there is a finite number of possible increments.
3) Add more base elements to $\hat{A}'$ by applying the subtracting expression e in $\hat{A}$ to the existing base elements of $\hat{A}'$. This should be repeated until no new base elements are added.
4) Put expression e' in $\hat{A}'$ such that e' is $\hat{A} + d$ where $d = |i - p|$.

If $i > p$ in expression e, then e is an increasing step even though it contains subtraction. The net increase is $i - p = d$, and the expression $e' : \hat{A} + d$ appropriately replaces e. Application of e and e' give different results only if the element of *paths* (A) to which they are applied is less than p. In that case e cannot be applied, but e' can be applied. This never happens, however, because no base element of $\hat{A}$ is less than p, and no rule is decreasing. So every value to which e is applied must be at least as big as p.

If $i < p$ in expression e, then e is a decreasing step. An inductive proof will be required to show that $e' : \hat{A} + (p - i)$ is an appropriate replacement for e. $\hat{A}'$ can be shown to generate *paths* (A) by proving propositions (A) and (B) below.

(A) If $\hat{A}$ generates l, then $\hat{A}'$ generates l.

Proof of (A) by induction on the number of steps in the generation of l.

Base. $\hat{A}$ generates 1 in one step.

In this case, l is a base element in $\hat{A}$. By construction step 1, l is also a base element of $\hat{A}'$.

Inductive hypothesis. Assume (A) true when $\hat{A}$ generates l in less than n steps.

Inductive step. Show (A) is true when $\hat{A}$ generates l in n steps.

Case 1. l is generated without increasing steps.

In this case, l is added to $\hat{A}'$ as a base element by construction rule 3.

Case 2. l is generated without subtraction steps.

In this case, $\hat{A}'$ has the same increasing steps as $\hat{A}$ by construction rule 1. The same generation sequence can be used to generate l in $\hat{A}'$.

Case 3. l is generated with a mixed sequence of addition and subtraction steps.

If the total increment does not exceed $d \times i_{\max}$, then l is a base element of $\hat{A}'$ by construction rules 2 and 3.

If there are x addition steps and y subtraction steps in the generation of l where $x > d$ and $y \leqslant i_{\max}$, then it can be shown that there is another generation of l in $\hat{A}$ which takes fewer steps. If there are more than d increment steps, lemma 4-32 tells us that there is a sequence of x' increment steps such that $x' \leqslant d$ and the sum of the x' increments is a multiple of d, say $y' \times d$ The sum of the x' increments must be less than or equal to $d \times i_{\max}$, and therefore $y' \leqslant i_{\max} \times d$. Thus, there is a generation of l which takes $x - x'$ increment steps and $y - y'$ decrement steps. The inductive hypothesis can be applied to establish that $\hat{A}'$ also generates l.

Now suppose there are x additions and y subtractions where $x > d$ and $y < i_{\max}$. Let i_j be the smallest of the x increments in the generation of l. Consider l', a number with the same generation sequence as l except for one less addition of i_j: $l' = l - i_j$. $\hat{A}$ also generates l' if it can be confirmed that no path error will occur. In the generation of l, the amount added exceeds $d \times i_{\max}$, and the amount subtracted is less than $d \times i_{\max}$. If the difference between the amount added and the amount subtracted is called the *gain*, then

$$\begin{aligned} gain &\geqslant (d+1) \times i_{\max} - (i_{\max} - 1) \times d \\ &\geqslant d \times i_{\max} + i_{\max} - d \times i_{\max} + d \\ &\geqslant i_{\max} + d. \end{aligned}$$

In the generation of l', there is still a positive *gain*, since $i_{\max} + d - i_j > 0$. So

l' is generated by $\hat{A}$ in fewer steps than l, and the inductive hypothesis can be applied to establish that l' is generated by $\hat{A}'$. By construction rule 1, $\hat{A}'$ has the same increasing expressions as $\hat{A}$, so $l = l' + i_j$ is generated by $\hat{A}'$.

(B) If $\hat{A}'$ generates l, then $\hat{A}$ generates l.

Proof of (B) by induction on the number of steps in the generation of l in $\hat{A}'$.

Base. $\hat{A}'$ generates l in one step.

In this case, l is a base element of $\hat{A}'$. By construction rules 1, 2, and 3, either l is a base element of $\hat{A}$, or it is generated in $\hat{A}$ with increment less than $d \times i_{\max}$ and a number of subtractions not exceeding $i_{\max}$.

Inductive hypothesis. Assume (B) is true when $\hat{A}'$ generates l in fewer than n steps.

Inductive step. Show that (B) is true when $\hat{A}'$ generates l in n steps.

If l is generated using a step which is not an application of e', then consider l' generated in $\hat{A}'$ with one less step: $l' = l - i_j$. The inductive hypothesis can be applied to establish that l' is also generated by $\hat{A}$. But since $\hat{A}'$ and $\hat{A}$ share all addition expressions except e', l is also generated by $\hat{A}$.

If l is generated by applications of e' only, then $l' = l - d$ is generated in $\hat{A}'$ and the inductive hypothesis to establish that l' is also generated by $\hat{A}$. But if l' is generated by $\hat{A}$, then d applications of an expression with some increment i_j and $i_j - 1$ applications of expression e can be made to get another element of $\hat{A}$:

$$l' + d \times i_j - (i_j - 1) \times d$$

$= l - d + d \times i_j - i_j \times d + d$

$= l.$

Thus, if l' is generated by $\hat{A}$, then so is l. QED.

The previous five lemmas supply enough machinery to remove all subtraction from an arbitrary length-predicting system, as shown in the theorem below. The proof of the theorem specifies the order in which the constructions of the supporting lemmas should be applied.

THEOREM 4-34. Suppose G is a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ and $\hat{G}$ is its corresponding length-predicting system. There is an equivalent system $\hat{G}'$ which does not use subtraction.

PROOF. To construct $\hat{G}'$, first apply finite substitution, arithmetic simplification, and linearization to each $\hat{A}$ where $A \in N$, giving $\hat{A}'$. Remove subtraction according to the method below, starting with $\hat{A}' = \hat{S}'$, the expression set corresponding to the start symbol.

1) For each expression in $\hat{A}'$ containing $\hat{B} \dot{-} p$ where no expression of $\hat{B}'$ refers to $\hat{A}$, remove the subtraction from $\hat{B}'$ by the current method and then subtract p from $\hat{B}'$ using the method of lemma 4-31.

2) Create two sets of expressions: $\hat{A}_0$ and $\hat{A}_+$. Put all the non-subtracting expressions of $\hat{A}'$ in $\hat{A}_+$ and leave $\hat{A}_0$ empty.

3) Remove the subtracting expression from $\hat{A}'$ which subtracts the smallest amount, p, and put it in $\hat{A}_+$.

4) Generate all the elements of $\hat{A}_+$ without using the subtracting expression

which are smaller than p or which exceed p by less than one increment. Remove the base elements less than p from $\hat{A}_+$ and put them in $\hat{A}_0$.

5) Transform $\hat{A}_+$ using the method of lemma 4-33 to remove the subtraction.

6) Repeat steps 3, 4, and 5 until no subtracting expressions remain.

7) Set $\hat{A}' = \{\hat{A}_0, \hat{A}_+\}$.

The steps in the above method use the constructions of the previous lemmas which guarantee that the path sets generated by the transformed systems are the same as those generated by the original systems. Small base elements are isolated from sets of expressions in step 4 to prevent transformed expressions from being applied to values for which the original expressions are undefined. The self-referential subtracting expressions which correspond to deletion cycles in the original grammar are transformed one at a time, beginning with the expression which deletes the least. Steps in a grammar cycle can be arbitrarily intermixed to give words of the same length. For any productive forest derived from a non-terminal which has deletion cycles, there is a corresponding forest which has the same derivation steps where repeated applications of the same rule are grouped together and the smallest deletions are first. The two forests may not yield the same string, but they yield strings of the same length. This justifies the consecutive transformation of deletion cycles. Simultaneous transformation would give the same result. QED.

COROLLARY 4-35. If A is a non-terminal of a 2DNF grammar $G = \langle \Sigma, \Xi, N, R, S \rangle_2^k$, then *paths* (A) is a regular set.

PROOF. Form the length-predicting system for the subgrammar whose start symbol is A and remove subtraction from the system. The unary representation of a positive integer n is just a string of n ones, and 0 is represented as the empty string. If all the integers in the length-predicting system are converted to unary representation and the addition operators are changed to concatenation, the system becomes a linear context-free grammar over a one-symbol alphabet which generates *paths*(A). Such a grammar can easily be converted to a regular grammar, and so *paths*(A) must be a regular set. QED.

The Remainder Operation on Context-Free Languages

The 2-d forest yield languages which involve deletion can be related to context-free languages with the help of the *remainders* of context-free languages. The remainder of language L_1 with respect to language L_2 is the set of strings that remain after prefixes from L_2 are deleted from words in L_1.

DEFINITION 4-36. Let L_1 and L_2 be string languages. The *remainder* of L_1 with respect to L_2 is defined as follows:

$$L_2 \setminus L_1 = \{w \mid \text{for some } v \text{ in } L_2, vw \text{ is in } L_1\}.$$

The remainder operation is defined for sets of strings in the preceding definition. It will be convenient to use the operator on single strings as well as sets of strings. Thus, $ab \setminus abcde = cde$. The remainder of a language is a close relative of the quotient of a language, as defined by Ginsburg and Greibach (1969). The quotient of a language L_1 with respect to a language L_2 is the set of prefixes that remain after suffixes from L_2 are deleted from words in L_1.

DEFINITION 4-37. Let L_1 and L_2 be string languages. The *quotient* of L_1 with respect to L_2 is defined as follows:

$$L_1 \,/\, L_2 = \{w \mid \text{for some } v \text{ in } L_2,\ wv \text{ is in } L_1\}.$$

It has been established that the quotient of a context-free language with respect to a regular language is context-free. It is also well known that both context-free and regular languages are closed under reversal. These facts can be used to show first that remainder can be expressed in terms of reversal and quotient, and then that the remainder of a context-free language with respect to a regular language is context-free.

THEOREM 4-38. Suppose L_1 and L_2 are string languages. Then

$$\mathit{reverse}(L_2 \backslash L_1) = \mathit{reverse}(L_1) \,/\, \mathit{reverse}(L_2).$$

PROOF. This theorem follows directly from the definitions of remainder, quotient, and reversal:

$$\begin{aligned} &\mathit{reverse}(L_2 \backslash L_1) \\ &= \mathit{reverse}(\{w \mid \text{for some } v \text{ in } L_2,\ vw \text{ is in } L_1\}) \\ &= \{w \mid \text{for some } v \text{ in } \mathit{reverse}(L_2),\ wv \text{ is in } \mathit{reverse}(L_1)\} \\ &= \mathit{reverse}(L_1) \,/\, \mathit{reverse}(L_2). \end{aligned}$$

QED.

THEOREM 4-39. If L_1 is a context-free language and L_2 is a regular language, then $L_2 \backslash L_1$ is a context-free language.

PROOF. This theorem is proven by using the previous theorem and closure for reversal and quotient.

L_1 is context-free and L_2 is regular

$\Rightarrow$ *reverse*(L_1) is context-free and *reverse*(L_2) is regular

$\Rightarrow$ *reverse*(L_1) / *reverse*(L_2) is context-free

$\Rightarrow$ *reverse*(*reverse*(L_1) / *reverse*(L_2)) is context-free

$\Rightarrow$ *reverse*(*reverse*$(L_2 \backslash L_1)$) is context-free

$\Rightarrow$ *reverse*$(L_2 \backslash L_1)$ is context-free

$\Rightarrow$ $L_2 \backslash L_1$ is context-free.

QED.

The strings that result from a deleting frontier operation on elements of a set of 2-d forests can be described as a remainder language. If $\alpha \in H_1^1(\Sigma, \Xi)$, then $str(sel(x^r, \alpha))$ is just $str(\alpha)$ with the first r characters removed. This can also be expressed as $\Sigma^r \setminus str(\alpha)$.

LEMMA 4-40. Suppose $\alpha = \#[_2 x^r][_1 \beta]$ is in $H_2^1(\Sigma, \Xi)$, where $\# \in \Sigma$, $x^r \in \Xi_1$, and $\beta \in H_2^1(\Sigma, \Xi)$, and $fr_1(\alpha)$ is defined. Then $str(fr_1(\alpha)) = \Sigma^r \setminus sfr(\beta)$.

PROOF. This follows from the definitions of remainder, the frontier function, and the *str* function:

$$\begin{aligned}
& sfr(\alpha) \\
&= str(subs_1(fr_1(x^r), fr_1(\beta))) \\
&= str(subs_1(x^r, fr_1(\beta))) \\
&= str(sel(x^r, fr_1(\beta))) \\
&= sfr(\beta) \text{ with a prefix of } r \text{ characters deleted.}
\end{aligned}$$

QED.

COROLLARY 4-41. Suppose $\alpha = \#[_2\,\beta][_1\,\gamma]$ is in $H_2^1(\Sigma, \Xi)$ where $\# \in \Sigma$, β and $\gamma \in H_2^1(\Sigma, \Xi)$, β has external selector $x^r \in \Xi_1$, and $fr_1(\alpha)$ is defined. Then $sfr(\alpha) = sfr(\beta)\cdot\Sigma^r \setminus sfr(fr_1(\gamma))$.

PROOF. This corollary follows from the previous lemma and the observation that $fr_1(\#[_2\,\beta][_1\,\gamma]) = fr_1(\#[_2\,\beta'][_1\,\#[_2\,x^r][_1\,\gamma]])$ where x^r is the external selector of β and β' is the same as β with external selector x^r replaced by x^0. QED.

In theorems 4-19 and 4-23, context-free grammars were built with non-terminals like ${}_jA_r$. The subgrammar constructed for ${}_jA_r$ is actually a grammar for $\Sigma^j \setminus (L(A):r)$. Remainder languages are helpful in relating deletion cycles to the theory of context-free languages. Analyses in computer science literature of deleting operations on context-free and context-sensitive languages suggest that there must always be a constant bound on the number of consecutive deletion operations. Theorem 4-39, however, demonstrates that regular deletion cycles are tolerated in context-free languages.

Normal Form for Deletion Cycle Analysis

Several terms are defined below to aid the analysis of deletion cycles in 2-d forest grammars. One non-terminal in a cycle is designated the *root*, or reference point for analysis. In many cycles, the number of non-terminals can be reduced to only one by expansion of the grammar rules. Such cycles have only one essential non-terminal. Cycles which cannot be expressed with only one non-terminal have several essential non-terminals.

DEFINITION 4-42. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$ such that A, B, and $C \in N$. B is a *descendent* of A if and only if

1) $A = B$,
2) there is a rule $A \rightarrow \beta$ and β contains B, or
3) there is a rule $A \rightarrow \beta$, β contains C, and B is a descendent of C.

DEFINITION 4-43. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$. G contains a *cycle* if and only if there is a subset of N such that each non-terminal in the subset is a descendent of every other non-terminal in the subset.

DEFINITION 4-44. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains a cycle $\{A_1, A_2, \ldots, A_r\} \subseteq N$. A_i is a root of the cycle if it appears in the right-hand side of a rule for a non-terminal which is not in the cycle.

DEFINITION 4-45. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains a cycle $C = \{A_1, A_2, \ldots, A_r\} \subseteq N$. A non-terminal A_i in the cycle is *essential* if it is a root of C, or if it is a root of another cycle which is a subset of C.

DEFINITION 4-46. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains a cycle $\{A_1, A_2, \ldots, A_r\}$ with root A_1. A cycle step is an expansion of A_1 to derive the smallest structure α such that α contains an essential non-terminal A_i and no non-terminal which precedes A_i in α is in the cycle.

The analysis of deletion cycles will be made easier by the use of a normal form for grammar cycles. Different shaped forests often have the same frontier. If $\alpha_1 = \#[_2 \#[_2 \beta][_1 \gamma]][_1 \delta]$ and $\alpha_2 = \#[_2 \beta][_1 \#[_2 \gamma][_1 \delta]]$, then $fr_1(\alpha_1) = fr_1(\alpha_2)$ provided the frontier of α_1 is defined. A grammar is in cyclic normal form if the forests derived from the cycles have the shape of α_1 rather than α_2.

DEFINITION 4-47. Let G be a grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains one or more deletion cycles. G is in *cyclic normal form* if and only if the path in every cycle step from the root of the cycle to the leftmost essential non-terminal is 2^i or $2^i\,1$, $i \geqslant 0$.

A grammar can be in cyclic normal form and 2-d normal form at the same time. In that case it will be called a cyclic 2DNF grammar. The conversion of a grammar to cyclic normal form takes advantage of the fact mentioned above that different shaped forests can yield the same strings. This yield-equivalence is formalized in the lemma below.

LEMMA 4-48. Suppose α_1, α_2, β, γ, and δ are forests in $H_2^1(\Sigma, \Xi)$ such that $\alpha_1 = \#[_2\, \#[_2\, \beta][_1\, \gamma]][_1\, \delta]$, $\alpha_2 = \#[_2\, \beta][_1\, \#[_2\, \gamma][_1\, \delta]]$, β has external selector x^p, and γ has external selector x^r. If $|fr_1(\gamma)| \geqslant p$, then $fr_1(\alpha_1) = fr_1(\alpha_2)$. If $|fr_1(\gamma)| = m < p$, then $fr_1(\#[_2\, \beta'][_1\, \delta]) = fr_1(\alpha_2)$ where β' is β with external selector x^p replaced by x^{p+r-m}.

PROOF. This lemma follows from the definition of the frontier function. First of all, assume that $|fr_1(\gamma)| \geqslant p$. Then the order of the substitutions during the frontier operation can be changed without affecting the resulting string:

$$\begin{aligned} fr_1(\alpha_1) &= subs_1(fr_1(\#[_2\, \beta][_1\, \gamma]),\ fr_1(\delta)) \\ &= subs_1(subs_1(fr_1(\beta),\ fr_1(\gamma)),\ fr_1(\delta)) \\ &= subs_1(fr_1(\beta),\ subs_1(fr_1(\gamma),\ fr_1(\delta))) \\ &= fr_1(\alpha_2) \end{aligned}$$

This is not true when $m = |fr_1(\gamma)| < p$, for then $fr_1(\#[_2\, \beta][_1\, \gamma])$ is undefined. The frontier operation on α_1 fails while it succeeds on α_2.

$fr_1(\alpha_2) = subs_1(fr_1(\beta), fr_1(\#[_2\,\gamma][_1\,\delta]))$

$= subs_1(fr_1(\beta), subs_1(fr_1(\gamma), fr_1(\delta)))$

The substitution of $fr_1(\delta)$ into $fr_1(\gamma)$ removes r symbols from the front of $fr_1(\delta)$. The substitution of this subforest into $fr_1(\beta)$ removes all m symbols of $fr_1(\gamma)$ and $p-m$ more symbols from $fr_1(\delta)$. So there is a forest with an equivalent frontier that does not contain γ at all: $fr_1(\alpha_2) = fr_1(\#[_2\,\beta'][_1\,\delta])$ where β' is constructed from β by replacing external selector x^p with x^{p+r-m}. QED.

THEOREM 4-49. Let G be a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ with one or more deletion cycles. There is a grammar $G' = <\Sigma', \Xi', N', R', S'>_2^k$ such that G' is in cyclic normal form and $Y_0(G) = Y_0(G')$.

PROOF. To begin the construction, set $G' = G$. Then, partition the non-terminals of G' according to external selectors and repeat the steps below until no further changes can be made. Assume A, B, C, D, and E are in N', and $\#, a \in \Sigma'$.

1) If R' has rules $A \rightarrow \#[_2\,B\,][_1\,C\,]$ and $C \rightarrow a\,[_1\,D\,]$ where A, C, and D are part of a deletion cycle and x^p is the external selector of B, then replace the rule for A with $A \rightarrow \#[_2\,B\,][_1\,C'\,]$ and add the rule $C' \rightarrow \#[_2\,a\,[_1\,x^0]][_1\,D\,]$, for new non-terminal C'.

2) If R' has rules $A \rightarrow \#[_2\,B\,][_1\,C\,]$ and $C \rightarrow \#[_2\,D\,][_1\,E\,]$ where A, C, and E are part of a deletion cycle, C is not essential, B and D are outside the cycle, the external selector of B is x^p, and the external selector of D is x^m, then replace the rule for A with $A \rightarrow \#[_2\,T\,][_1\,E\,]$, $A \rightarrow \#[_2\,B_1][_1\,E\,]$, $A \rightarrow \#[_2\,B_2][_1\,E\,]$, . . . , $A \rightarrow \#[_2\,B_{p-1}][_1\,E\,]$. Then, add rules for B_i only when $0 \leqslant i < p$ and

$i \in paths(D)$. B_i has the same rules as B, except that external selector x^p is replaced by x^{p+m-i}. Finally, add a rule $T \rightarrow \#[_2\, B\,][_1\, D\,]$ for newly invented non-terminal T.

3) Perform the same modification as in step 2 when A, C, and D are part of the cycle, C is not essential, and B and E are outside the cycle.

To prove the theorem, it needs to be shown for the construction steps above that the algorithm halts, the resulting grammar G' is in cyclic normal form, and the yield of G' is the same as the yield of G. Step 1 just changes the form of certain cycle rules so that step 2 can be applied. Since no rules of the form $C \rightarrow a\,[_1\, D\,]$ are added by any step, the number of applications of step 1 is finite. Step 1 does not affect the yield of the grammar because $fr_1(\#[_2\, a\,[_1\, x^0]][_1\, D\,]) = fr_1(a\,[_1\, D\,])$.

Steps 2 and 3 both have the effect of eliminating a non-essential non-terminal with a 1-arc pointing to it. When all such non-terminals are gone, the algorithm halts, and the resulting grammar will be in cyclic normal form. Since no 1-arcs point to non-essential cycle non-terminals, the path from the root to an essential non-terminal in a cycle step can have 1 only at the end. It is evident from lemma 4-48 that the modifications made in steps 2 and 3 will not affect the yields of derived forests. Thus, G' is in cyclic normal form, and $Y_0(G') = Y_0(G)$. QED.

Once a grammar has been put into cyclic normal form, the right-hand sides of rules for essential cycle non-terminals can be expanded to become full cycle steps. It then becomes evident that there are three kinds of cycle steps, as illustrated in figure 4 for cycles with one essential non-terminal B. A cycle which has only type

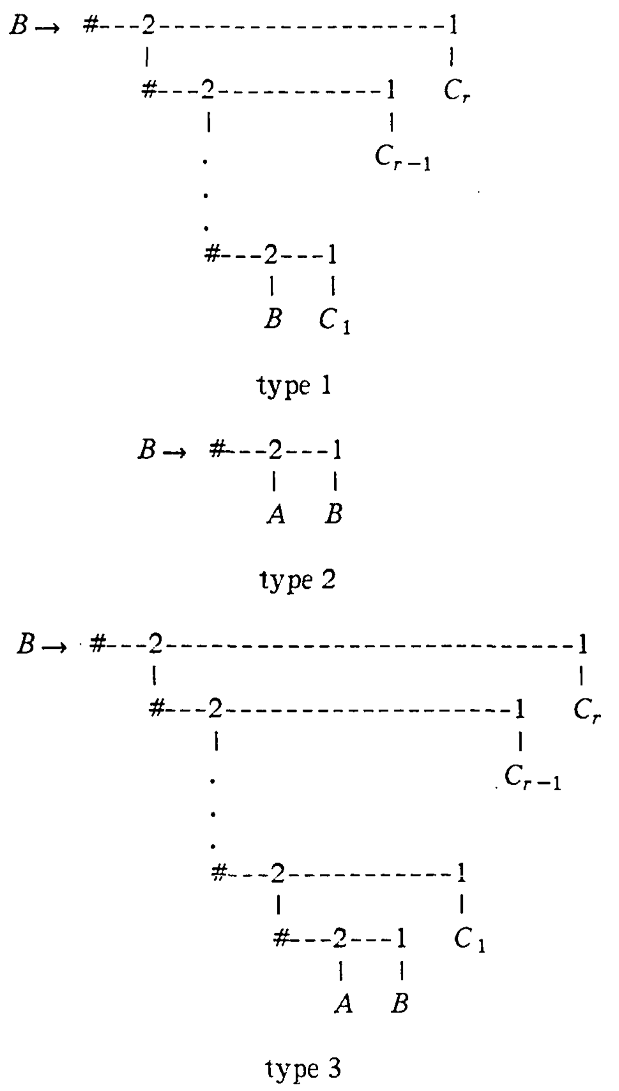

FIGURE 4. Normal form cycle steps

1 steps is non-deleting. If the external selectors of the non-terminals C_i overlap one another, a path error will occur. Cycles with type 2 and type 3 steps can be deleting cycles, since the external selectors from repeated occurrences of A can

overlap one another without causing path errors. Type 3 steps are more complicated. They allow a suffix to be attached to the end of a string every time a prefix is deleted from the front during the frontier operation. The example grammar ABC from the beginning of this chapter contains a deletion cycle with type 3 steps. Deletion cycles with type 2 steps can be shown to have corresponding context-free grammars by application of the remainder operation. Cycles with type 3 steps are more difficult to simulate with a context-free grammar.

2-d Forest Grammars with Regular Deletion Cycles

Cycles of type 2 in figure 4 will be called regular deletion cycles, and 2-d forest grammars which have only regular deletion cycles can be shown to yield context-free languages. A regular deletion cycle can be used to form a regular set called a *deletion map*, which predicts the number of symbols that will be deleted in the cycle. The deletion map can be applied with the remainder operation on subgrammars which are not part of the cycle to generated the yield.

DEFINITION 4-50. Let G be a cyclic 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$. It G contains a deletion cycle such that the left-most non-terminal in every cycle step is not part of the cycle, then G has regular deletion cycle.

The deletion map for a cyclic 2DNF grammar $G = <\Sigma, \Xi, N, R, S>_2^k$ with a regular deletion cycle can be constructed according to the steps below.

1) Partition the grammar G according to external selectors. If there is a rule $B \rightarrow \#[_2 C\,][_1\, D\,]$ where B is in a deletion cycle but C and D are not, replace the rule with $B \rightarrow T$ and $T \rightarrow \#[_2 C\,][_1\, D\,]$ for newly invented non-terminal T.

2) Make a regular grammar from the rules in R for each i, m, A, and B where i and m are paths on external selectors, A is a non-terminal in the deletion cycle, and B is any non-terminal in N. The grammar ${}_iAB_m$ = $<N', \Sigma', R', S'>$ is formed as follows:

   a) Set $\Sigma' = \{0, 1\}$, $N' = N$, and $S' = A$.

   b) Put a rule $S \rightarrow 0^i \cdot A$ in R'.

   c) If R has a rule $C \rightarrow \#[_2 D][_1 E]$, then put $C \rightarrow \hat{D} \cdot 0^i \cdot E$ in R' and add to R' rules for regular grammar $\hat{D}$ which generates the lengths of all the strings yielded by D.

   d) If R has a rule $C \rightarrow \#[_2 D][_1 E]$, either $E = B$ or R also has a rule $E \rightarrow B$, and the path on the external selector of D is m, then put $C \rightarrow \hat{D}$ in R', and add the rules for $\hat{D}$ to R'.

   e) Rewrite the rules of R', adding new non-terminals as required, so that each rule has the form $E \rightarrow n \cdot F$ or $E \rightarrow n$, for $n = 0$ or 1, and non-terminals E and F.

3) Convert each ${}_iAB_m$ to a push-down automaton ${}_i\overline{AB}_m$ = $<Q, \Sigma'', \Gamma, \delta, \Gamma_0, Q_0, Q_F>$ as follows:

   a) Set $Q = N' \bigcup \{F\}$, $\Sigma'' = \{0\}$, $\Gamma = \{\epsilon, 0\}$, $\Gamma_0 = \epsilon$, $Q_0 = S'$, and $Q_F = \{F\}$.

   b) If R' has a rule $C \rightarrow 0 \cdot D$, put $(C, *, \lambda) = (D, *0)$ in δ, where $*$ is any stack symbol and λ means no input is consumed.

   c) If R' has a rule $C \rightarrow 1 \cdot D$, put $(C, \gamma 0, \lambda) = (D, \gamma)$ in δ, where $\gamma \epsilon \Gamma^+$.

   d) If R' has a rule $C \rightarrow 0$, put $(C, *, \lambda):(F, *0)$ in δ.

   e) If R' has a rule $C \rightarrow 1$, put $(C, \gamma 0, \lambda):(F, \gamma)$ in δ.

f) Add $(F, 0, 0){:}(F, \lambda)$ to δ. The automaton stops, accepting a string of zeros only when the stack is empty and all the input has been consumed.

The regular set constructed in step 2 contains strings of ones and zeros. The zeros are unary representations of the paths on selectors in the cycle. The ones represent characters embedded within the cycle that will be deleted. The strings of interest in the set are those in which the zeros outnumber the ones, not just overall, but in every possible prefix. The grammar is converted to a pda in step 3 to accept only the strings of interest in the regular set. The pda uses λ-moves as it simulates the regular grammar. Zeros from the regular grammar are pushed on the stack, and the stack is popped when the regular grammar produces a 1. If any initial sequence of moves is chosen in which the ones outnumber the zeros, then the pda gets stuck. If the zeros always outnumber the ones, however, the simulation of the regular grammar terminates and the pda moves to a final state which reads zeros from input and accepts the string if it matches what is on the stack. Since Σ'' has only one element, the accepted set is regular.

DEFINITION 4-51. The *deletion map* for a regular deletion cycle which contains non-terminals A and B in a 2DNF grammar G is a regular set ${}_n\overline{AB}_m$, formed by the construction steps above.

LEMMA 4-52. Let G be a cyclic 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ which has a regular deletion cycle containing non-terminals A and B. Suppose ${}_n\overline{AB}_m$ is a regular deletion map for the cycle. The string 0^d is recognized by ${}_n\overline{AB}_m$ if and only if there are α and β such that $A \Rightarrow^* \alpha$, $B \Rightarrow^* \beta$, β is a subforest of α, m is the path on the external selector of the subforest which precedes β, and $\Sigma^n \setminus sfr(\alpha) = \Sigma^{d+m} \setminus sfr(\beta)$.

PROOF. This lemma is established by proving propositions (A) and (B) below.

(A) If there exist α and β such that $A \Rightarrow^* \alpha$, $B \Rightarrow^* \beta, \ldots$, and $\Sigma^n \setminus sfr(\alpha) = \Sigma^{d+m} \setminus sfr(\beta)$, then 0^d is recognized by ${}_n\overline{AB}_m$.

Proof of (A) by induction on the number of cycle steps in the derivation of α.

Base. $A \Rightarrow^* \alpha$ in one cycle step.

In this case, there are productions in the forest grammar $A \rightarrow \#[_2 D][_1 B]$ and $B \rightarrow E$ where A and B are in the cycle, D and E are not in the cycle, and the external selector of D is x^m. So $\alpha = \#[_2 \delta][_1 \beta]$, and $\Sigma^n \setminus sfr(\alpha) = \Sigma^{n-l+m} \setminus sfr(\beta)$ where l is the length of the yield of δ. We know that δ contributes nothing to the frontier, so it must be the case that $n > l$. By step 2 of the construction, ${}_nAB_m$ is given rules $S \rightarrow 0^n \cdot A$ and $A \rightarrow \hat{D}$. By step 3, ${}_n\overline{AB}_m$ has corresponding rules which 1) push n zeros on the stack, 2) pop l zeros off the stack where $\hat{D}$ derives a string of l ones, and 3) move to a final state which accepts an input string of $n-l$ zeros. Thus, (A) is satisfied where $d = n-l+m$.

Inductive hypothesis. Assume (A) is true for $A \Rightarrow^* \alpha$ in $r-1$ cycle steps.

Inductive step. Show (A) is true for $A \Rightarrow^* \alpha$ in r cycle steps.

There are productions $A \rightarrow \#[_2 D_1][_1 C_1]$, $C_1 \rightarrow \#[_2 D_2][_1 C_2], \ldots$, $C_q \rightarrow \#[_2 D_q][_1 B]$, and $B \rightarrow E$ where $A, C_1, C_2, \ldots, C_q, B$ are in the cycle, $D_1, \ldots, D_q$ and E are not in the cycle, and the external selector of D_q is x^m. Let the external selectors of D_i, $1 \leqslant i \leqslant q-1$, be x^{p_i}. We have $\alpha = \#[_2 \delta][_1 \gamma]$ where $D_1 \Rightarrow^* \delta$ and $C_1 \Rightarrow^* \gamma$, $sfr(\alpha) = str(subs_1(fr_1(\delta), fr_1(\gamma)))$, and β is a substructure of γ. Since the yield of α is a suffix of the yield of β, the yield of δ is completely deleted. If $|sfr(\delta)| = l_1$, then $l_1 < n$. So $\Sigma^n \setminus sfr(\alpha) = \Sigma^n \setminus (sfr(\delta)) \cdot \Sigma^{p_1} \setminus sfr(\gamma)) = \Sigma^{d+m} \setminus sfr(\beta)$. An expansion of d would show that $d + m = n - l_1 + p_1 - l_2 + \ldots + p_{q-1} - l_q + m$, and so $\Sigma^{p_1} \setminus sfr(\gamma) = \Sigma^{d'+m} \setminus sfr(\beta)$ where $d' = d - n + l_1$. The inductive hypothesis can be applied to establish that ${}_{p_1}\overline{C_1 B_m}$ contains d'. By construction steps 2 and 3, ${}_n AB_m$ has a rule $A \to \hat{D}_1 \cdot 0^{p_1} \cdot C_1$, and ${}_n\overline{AB_m}$ pushes n zeros on the stack, pops l_1 zeros, pushes p_1 zeros and goes to a state C_1, where l_1 is a length generated by $\hat{D}_1$. We also know that ${}_{p_1}\overline{C_1 B_m}$ pushes p_1 zeros and goes to state C_1. So if ${}_{p_1}\overline{C_1 B_m}$ accepts d', then ${}_n\overline{AB_m}$ accepts $n - l_1 + d'$. But $n - l_1 + d' = n - l_1 + d - n + l_1 = d$, so (A) is satisfied.

(B) If 0^d is recognized by ${}_n\overline{AB_m}$, then there are α and β such that $A \Rightarrow^* \alpha$, $B \Rightarrow^* \beta, \ldots,$ and $\Sigma^n \setminus sfr(\alpha) = \Sigma^{d+m} \setminus sfr(\beta)$.

Proof of (B) by induction on the number of cycle steps in the derivation of α. The proof of (B) is similar to the proof of (A). QED.

THEOREM 4-53. Suppose G is a cyclic 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains only regular deletion cycles. There is a context-free grammar $G' = <N', \Sigma', R', S'>$ such that $L(G') = Y_0(G)$.

PROOF. To construct G' from G, first partition G according to external selectors. The non-terminal set will include ${}_i A_m$ for every A in N, every external selector path m, and every integer $0 \leqslant i \leqslant p_{\max}$ where $p_{\max}$ is the longest path on any selector. Set $\Sigma' = \Sigma$ and $S' = {}_0 S_m$ for each external selector path m. Begin making the rules with the non-terminals in the deletion cycle closest to the start symbol, following the steps below.

1) Find each non-terminal E which is not in the cycle, but is in the right-hand side of a rule for non-terminal A which is in the cycle. Form rules for ${}_iE_m$ and add them to R', where m is the path on the external selector for any subforest which can precede E in a derivation. If the subgrammar G_E has no deletion cycles, then form its rules using the method of theorem 4-23. If G_E does contain a deletion cycle, then use the method currently being described.

2) Form ${}_i\overline{AB}_m$ for each pair of non-terminals, A and B, in the cycle and each pair of paths, i and m, on external selectors.

3) If there is a rule $A \rightarrow E$ such that A is in the deletion cycle but E is not, put a rule ${}_iA_m \rightarrow {}_iE_m$ in R' for each possible i and for m such that x^m is the external selector of A.

4) If there is a rule $A \rightarrow \#[_2 E][_1 B]$ where A and B are in the cycle but E is not, put ${}_iA_m \rightarrow {}_iE_r \cdot {}_rB_m$ in R' for each possible i, r such that x^r is the external selector of E, and m such that x^m is the external selector of A.

5) If R has rules $A \rightarrow \#[_2 E][_1 B]$ and $C \rightarrow F$ where A, B, and C are in the deletion cycle but E and F are not, put rules ${}_iA_m \rightarrow \; < {}_i\overline{AC}_r \setminus {}_rF_m >$ in R' for each possible i, r such that x^r is the external selector preceding C, and m such that x^m is the external selector of F. Then, add a subgrammar to R' for each non-terminal $< {}_i\overline{AC}_r \setminus {}_rF_m >$ such that $< {}_i\overline{AC}_r \setminus {}_rF_m >$ generates the context-free language $h(L({}_i\overline{AC}_r)) \setminus L({}_rF_m)$, where h is the following homomorphism on strings of zeros in $L({}_i\overline{AC}_r)$: $h(0) = a$ where a is any

element of Σ.

6) If R has rules $A \rightarrow \#[_2 E][_1 B]$ and $C \rightarrow \#[_2 F][_1 D]$ where A, B, C, and D are in the deletion cycle but E and F are not, put rules ${}_i A_m \rightarrow \; <_i \overline{AC}_r \setminus {}_r F_l> \cdot {}_l D_m$ in R' for each possible i, r such that x^r is an external selector which precedes C, l such that x^l is the external selector of F, and m such that x^m is the external selector of D. Then, add a subgrammar to R' for each non-terminal $<_i \overline{AC}_r \setminus {}_r F_l>$ such that $<_i \overline{AC}_r \setminus {}_r F_l>$ generates the context-free language $h(L({}_i \overline{AC}_r)) \setminus L({}_r F_l)$, where h is the following homomorphism on strings of zeros in $L({}_i \overline{AC}_r)$: $h(0) = a$ where a is any element of Σ.

Regular deletion maps and remainder operations are used in steps 5 and 6. The remainder operation is applied only to non-terminals outside the cycle, allowing context-free subgrammars to be constructed before the operation is applied. Then, the remainder of a context-free language with respect to a regular language is formed to give another context-free language (theorem 4-39). The grammar G' can be shown to satisfy theorem 4-53 in the proof of propositions (A) and (B) below. (A) If $A \in N$, $A \Rightarrow^* \alpha$, $\Sigma^i \setminus sfr(\alpha) = w$, and α has external selector x^m, then N' contains ${}_i A_m$ such that ${}_i A_m \Rightarrow^* w$.

Proof of (A) by induction on the number of steps in the derivation of α.

Base. $A \Rightarrow^* \alpha$ in one step.

It must be the case that $\alpha = x^m$, $i = 0$, and $w = \lambda$. R has a rule $A \rightarrow x^m$ and R' has ${}_0 A_m \rightarrow \lambda$, as in the proof of theorem 4-23.

Inductive hypothesis. Assume (A) is true when $A \Rightarrow^* \alpha$ in less than n steps.

Inductive step. Show that (A) is true when $A \Rightarrow^* \alpha$ in n steps.

If the first step in the derivation of α is $A \rightarrow a[_1 B]$ or $A \rightarrow \#[_2 B]$, then the proof is the same as in theorem 4-19.

If the first step in the derivation of α is $A \rightarrow E$, then $E \Rightarrow^* \alpha$ in one less step. If $\Sigma^i \setminus sfr(\alpha) = w$ and α has external selector m, the inductive hypothesis can be applied to establish that ${}_iB_m \Rightarrow^* w$. By construction step 3, R' has ${}_iA_m \rightarrow {}_iE_m$, and so ${}_iA_m \Rightarrow^* w$.

If the first step in the derivation of α is $A \rightarrow \#[_2 E][_1 B]$ and neither A nor B are in a deletion cycle, then ${}_iA_m \Rightarrow^* w$ as in the proofs of theorems 4-19 and 4-23.

If the first step is $A \rightarrow \#[_2 E][_1 B]$ where A and B are in a deletion cycle, first consider the case in which the yield of E is not completely deleted. Then $E \Rightarrow^* \gamma$, $B \Rightarrow^* \beta$, $\alpha = \#[_2 \gamma][_1 \beta]$, $w = w_1 w_2$, $\Sigma^i \setminus sfr(\gamma) = w_1$, and $\Sigma^r \setminus sfr(\beta) = w_2$ where the external selector of γ is x^r. The inductive hypothesis can be applied to establish that R' has ${}_iE_r \Rightarrow^* w_1$ and ${}_rB_m \Rightarrow^* w_2$. Step 4 of the construction gives ${}_iA_m \rightarrow \ {}_iE_r \cdot {}_rB_m.$ Thus, ${}_iA_m \Rightarrow^* \ w_1 w_2 = w$, and (A) is satisfied.

Now suppose the first derivation step is $A \rightarrow \#[_2 E][_1 B]$ with A and B in the deletion cycle, and the yield of E is completely deleted during the frontier operation. Let $\alpha = \#[_2 \gamma_1][_1 \#[_2 \gamma_2][_1 \cdots \#[_2 \gamma_q][_1 \delta] \cdots]]$ where γ_q is either the first subforest that is not completely deleted in the cycle, or it is the last subforest in the cycle. Assume also that $C \Rightarrow^* \delta$, C is in the deletion cycle, the external selector of γ_q is x^r, and the external selectors for γ_i, $1 \leqslant i < q$, are x^{p_i}. If l_i represents the length of the string yielded by γ_i, then $w =$

$\Sigma^i \setminus sfr(\alpha) = \Sigma^{d+r} \setminus sfr(\delta)$ where $d = i - l_1 + p_1 - l_2 + ... + p_{q-1} - l_q$. This means that $\Sigma^r \setminus sfr(\delta) = w'w$ where $|w'| = d$. The inductive hypothesis can be applied to establish that ${}_rC_m \Rightarrow^* w'w$.

If γ_q is the last leading subforest of the cycle, then the first step in the derivation of δ is $C \rightarrow F$, where F is outside the cycle. According to construction step 5, R' has a rule ${}_iA_m \rightarrow \ < {}_i\overline{AC}_r \setminus {}_rF_m >$ and rules for $h(L({}_i\overline{AC}_r)) \setminus L({}_rF_m)$, and lemma 4-52 assures us that 0^d is accepted by ${}_i\overline{AC}_r$. Since ${}_rF_m \Rightarrow^* w'w$ and $|w'| = d$, it follows that ${}_iA_m \Rightarrow^*$ $\Sigma^d \setminus w'w = w$.

Finally, suppose γ_q is not the last subforest in the deletion cycle, but it does contribute to the yield. In this case, the first step of the derivation of δ is $C \rightarrow \#[_2 F][_1 D]$ where C and D are both in the cycle. R' has a corresponding rule ${}_rC_m \rightarrow \ {}_rF_l \cdot {}_lD_m$ and the inductive hypothesis establishes that ${}_rC_m \Rightarrow^* w'w$ where $|w'| = d$. Since F contributes to the yield, it must be true that ${}_rF_l \Rightarrow^* w'w_1$, ${}_lD_m \Rightarrow^* w_2$, and $w_1w_2 = w$. According to construction step 6, R' has a rule ${}_iA_m \rightarrow \ < {}_i\overline{AC}_r \setminus {}_rF_l > \cdot {}_lD_m$ as well as rules for $h(L({}_i\overline{AC}_r)) \setminus \ L({}_rF_l)$. We also know from lemma 4-52 that ${}_i\overline{AC}_r$ accepts 0^d. It follows that ${}_iA_m \Rightarrow^* \ d \setminus w'w_1 \cdot w_2 = w_1w_2 = w$, satisfying (A).

(B) If N' contains ${}_iA_m$ such that ${}_iA_m \Rightarrow^* w$, then N contains A such that $A \Rightarrow^* \alpha$, $\Sigma^i \setminus sfr(\alpha) = w$, and α has external selector x^m.

Proposition (B) can be proven in a manner similar to the proof of (A) by induction on the number of steps in the derivation of w. QED.

Grammars with Consistent Deletion Cycles

The theorem above shows that a forest grammar whose deletion cycles have only type 2 steps can be converted to a context-free grammar. A similar result can be obtained for forest grammars with type 3 deletion cycles, provided they are consistent. The conversion process begins with the formation of an inversion map for the cycle. This is a regular set which specifies the number of symbols that will be deleted by the cycle and indicates what substrings will be attached to the end of the resulting string as the deletions occur. An inversion operation is then applied which takes an inversion map and a context-free grammar, and produces a new context-free grammar. The new grammar produces only the strings which are yielded by the deletion cycle.

DEFINITION 4-54. Let G be a cyclic 2DNF grammar with a deletion cycle. A non-terminal D is a *leading non−terminal* in a cycle step if the path from the root of the cycle step to D is in 2^+, and D is not in the cycle. A non-terminal is a *trailing non−terminal* if it follows a non-terminal which is in the cycle.

DEFINITION 4-55. Let G be a cyclic 2DNF grammar with a deletion cycle. The deletion cycle is *consistent* if the external selector of the leading non-terminal in each cycle step is always an overlapping selector.

Given a grammar $G = <\Sigma, \Xi, N, R, S>_2^k$ in cyclic 2DNF, the inversion map can be constructed by the steps below.

1) Partition the non-terminals of G which are not in the deletion cycle according

to external selectors. If there is a rule $B \rightarrow \#[_2 C\ ][_1 E]$ where B is in the cycle but C and E are not, then replace the rule with $B \rightarrow T$ and $T \rightarrow \#[_2 C\ ][_1 E]$ for newly invented non-terminal T.

2) Form a regular grammar *lmap* (G, A, D) for each A and E in the deletion cycle of G. The terminal set of *lmap* is a subset of $\{0, t_{A_1}, t_{A_2}, \ldots, t_{A_r}, l_{A_1}, \ldots, l_{A_r}\}$ where $\{A_1, A_2, \ldots, A_r\} = N$. The rules of *lmap* (G, A, D) are constructed according to the steps below. Assume A, B, C, D, E, and J are non-terminals A_i for some i.

   a) If R has $B \rightarrow \#[_2 E\ ][_1 C\ ]$ where B and E are in the deletion cycle but C is not, add $B \rightarrow E \cdot t_C$ to the rules of *lmap* (G, A, D).

   b) If R has a rule $B \rightarrow \#[_2 J\ ][_1 E]$ where B and E are in the cycle but C is not, add $B \rightarrow E \cdot 0^p \cdot l_J$ to the rules of *lmap* where x^p is the external selector of C.

   c) If R has $D \rightarrow \beta$ for any right-hand side β, add $D \rightarrow \lambda$ to the rules of *lmap* (G, A, D).

   d) Make A the start symbol of *lmap* (G, A, D).

3) Form a regular grammar *tmap* (G, A, D) for each A and D in the deletion cycle of G such that there is a rule $B \rightarrow \#[_2 D\ ][_1 E]$ where B is in the cycle but E is not. Use the same rules to construct *tmap* as in step 2, but replace 2c with the following:

   c) If R has $B \rightarrow \#[_2 D\ ][_1 E]$ where B and D are in the cycle but E is not, add $B \rightarrow 0^p \cdot l_D$ to *tmap* where x^p is the external selector of D.

4) Form a general sequential machine Z which operates on strings produced by

lmap or *tmap*. The input alphabet is the same as that of *lmap*, and the output alphabet is the input alphabet with $\{s_{A_1}, s_{A_2}, \ldots, s_{A_r}\}$ added, where A_i is a non-terminal of G. There is a start state z_0, a final state z_f, and a state z_i for each A_i. The rules of the mapping are defined as follows:

a) Add $(z_0, \lambda) = (z_i, s_{A_i})$ for every $1 \leqslant i \leqslant r$.

b) Add $(z_i, 0) = (z_i, 0)$, $(z_i, l_{A_i}) = (z_i, l_{A_j})$, and $(z_i, t_{A_j}) = (z_i, t_{A_j})$ for every i and j between 1 and r.

c) Add $(z_i, l_{A_i}) = (z_f, \lambda)$ for every $1 \leqslant i \leqslant r$.

d) Add $(z_f, 0) = (z_f, 0)$ and $(z_f, t_{A_i}) = (z_f, t_{A_i})$ for every A_i.

e) Add $(z_0, \lambda) = (z_f, \lambda)$.

5) Form the length-predicting grammars $\hat{A}_i$ for each non-terminal A_i which is in the deletion cycle of G or is a leading non-terminal in the deletion cycle. Convert each $\hat{A}_i$ to general sequential machine M_i as follows:

a) Convert $\hat{A}_i$ to a finite automaton $\hat{A}'_i$.

b) Give M_i the same state set, alphabet and start state as $\hat{A}'_i$.

c) If $\hat{A}'_i$ has a rule $(q, 1) = p$ for states q and p, put a corresponding rule $(q, 0) = (p, \lambda)$ in the mapping of M_i.

d) Add a rule $(q, t_C) = (q, t_C)$ for each state q of M_i and each terminal of the form t_C in the terminal set of $lmap(G, A, D)$.

6) Combine the machines M_i to form a gsm M_0 which will operate on strings of $lmap(G, A, D)$ or $tmap(G, A, D)$. The input alphabet is the alphabet of *lmap*, and the output alphabet is a subset of the input alphabet without l_{A_i}

for A_i which are leading non-terminals in the deletion cycle. The start state and final state is a new state q_0. The mapping rules are as follows:

a) Add a rule $(q_0, l_{A_i}) = (m_{0_i}, \lambda)$ for each l_{A_i} and m_{0_i} such that A_i is a leading non-terminal of the deletion cycle and m_{0_i} is the start state of M_i.

b) Add a rule $(q_0, l_{A_i}) = (q_0, l_{A_i})$ for each A_i which is in the deletion cycle.

c) Add a rule $(q_0, t_{A_i}) = (q_0, t_{A_i})$ and $(q_0, s_{A_i}) = (q_0, s_{A_i})$ for each A_i.

d) Add a rule $(q_0, 0) = (q_0, 0)$.

e) Add a rule $(m_{f_i}, \lambda) = (q_0, \lambda)$ for each m_{f_i} which is a final state of M_i.

7) Construct another gsm P from the machines M_i such that non-terminal A_i is in the deletion cycle of G. P will operate on the output of M_0. The input alphabet is the same as the output alphabet of M_0. The output alphabet is the input alphabet with any terminals of the form l_{A_i}. The start state and final state is p_0. The mapping rules are as follows:

a) Add a rule $(p_0, l_{A_i}) = (m_{0_i}, \lambda)$ for l_{A_i} and m_{0_i} such that A_i is in the deletion cycle of G.

b) Add a rule $(m_{f_i}, \lambda) = (p_0, \lambda)$ for each m_{f_i} which is a final state of M_i.

c) Add rules $(p_0, 0) = (p_0, 0)$, $(p_0, t_{A_i}) = (p_0, t_{A_i})$, and $(p_0, s_{A_i}) = (p_0, s_{A_i})$ for each A_i.

DEFINITION 4-56. Let G be a cyclic 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ with a consistent deletion cycle which contains non-terminals A and D. Then ${}_A\overline{X}_D$ is an *inversion map* $P(M_0(Z(lmap(G, A, D))))$ and ${}_A\overline{Y}_D$ is an *inversion map* $P(M_0(Z(tmap(G, A, D))))$ where $lmap$, $tmap$, P, Z, and M_0 are defined by the construction steps above.

A grammar *lmap* (G, A, D) or *tmap* (G, A, D) generates a regular set which contains strings of zeros and terminals of the form s_J, l_J, and t_C, corresponding to non-terminals J and C of G. Each string in the set corresponds to some forest derived from G. There is a zero in a string for each symbol that will be deleted by the deletion cycle during the frontier operation on the forest to which the string corresponds. Each string starts with a terminal s_J representing the initial undeleted subforest derived from J. There is also a terminal l_J for each subforest derived from non-terminal J which will be completely deleted by the deletion cycle. For each subforest derived from trailer non-terminal C which will be attached to the end of the yield during the frontier operation, the *lmap* (or *tmap*) string has a terminal t_C. The grammar *tmap* (G, A, D) differs from *lmap* (G, A, D) in that its strings correspond to forests in which subforest derived from trailing non-terminals will be deleted during the frontier operation. In forests which have corresponding *lmap* strings, no trailers are deleted by the deletion cycle.

The general sequential machines Z, M_0, and P remove the l_J terminals from strings of *lmap* and *tmap*. Z has the effect of moving the last l_J in the string to the front and renaming it s_J. M_0 simulates embedded deletion. Each time M_0 encounters a terminal l_J in its input string where J is a leading non-terminal in the deletion cycle, control is transferred to a submachine which reads a number of zeros corresponding to the length of some word yielded by J. As the zeros are read, no output is produced. This simulates the deletion of subforests derived from leading non-terminals in the cycle. Given the restriction that the deletion cycles are consistent, this simulation can be accomplished by a gsm. Without the consistency requirement, a stack transducer would be required.

The machine P is similar to M_0, except that it operates only on terminals l_B where B is in the deletion cycle. There are no such terminals in *lmap* strings, and there is at most on such terminal at the beginning of *tmap* strings. P, therefore, has no effect on *lmap* strings. Operating on *tmap* strings, P simulates the deletion of an arbitrarily large subforest containing both leaders and trailers. A gsm successfully achieves this simulation since it happens only once in the string, and since all other terminals l_J have been removed by M_0 before P is applied.

Note that *lmap* and *tmap* are regular grammars, and regular sets are closed under gsm mappings, so ${}_A\overline{X}_D$ and ${}_A\overline{Y}_D$ are also regular sets. An inversion operation will be defined later to complete the deletion and move the terminals T_C to their proper locations. The next lemma establishes the relationship between the reduced *lmap* strings and the strings yielded from forest grammar G.

DEFINITION 4-57. Let h_1 be a homomorphism on a set of strings such that $h_1(1) = 1$, $h_1(0) = 0$, and $h_1(a) = \lambda$ for $a \neq 1$ and $a \neq 0$.

DEFINITION 4-58. Let h_2 be a homomorphism on a set of strings such that $h_2(1) = \lambda$, $h_2(0) = \lambda$, and $h_2(a) = a$ for $a \neq 1$ and $a \neq 0$.

LEMMA 4-59. Let G be a cyclic 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains a consistent deletion cycle. Suppose non-terminals A and E are in the cycle, non-terminals I, B, $C_1, C_2, \ldots, C_r$ are outside the cycle, and R has a rule $D \rightarrow T$. Then ${}_A\overline{X}_D$ contains w such that $h_1(w) = 0^d$ and $h_2(w) =$

$s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ if and only if $A \Rightarrow^* \beta$, $T \Rightarrow^* \gamma_0$, $I \Rightarrow^* \eta$, $C_j \Rightarrow^* \gamma_j$ for $1 \leqslant j \leqslant r$, and $sfr(\alpha) = sfr(\eta) \cdot \Sigma^d \setminus sfr(\gamma_0) \cdot \Sigma^{p_0} \setminus sfr(\gamma_1) \cdot \ldots \cdot \Sigma^{p_{r-1}} \setminus sfr(\gamma_r)$ where $|sfr(\eta)| = i$ and x^{p_j} is the external selector of γ_j for $0 \leqslant j \leqslant r$.

PROOF. The lemma is established by proving propositions (A) and (B) below.

(A) If $A \Rightarrow^* \alpha, T \Rightarrow^* \gamma_0, \ldots,$ then ${}_A\overline{X}_D$ has w such that $h_1(w) = 0^d$ and $h_2(w) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Proof of (A) by induction on the number of derivation steps for α using non-terminals in the cycle.

Base. $A \Rightarrow^* \alpha$ using one step for A.

It must be the case that $A = D$, and α is derived with $A \Rightarrow T \Rightarrow^* \gamma_0$. $lmap(G, A, A)$ has a rule $A \to \lambda$. Z, M_0, and P produce λ on input λ. So ${}_A\overline{X}_A$ has $w = \lambda$, $h_1(w) = \lambda$ and $h_2(w) = \lambda$. $sfr(\alpha) = sfr(\gamma_0) = \Sigma^0 \setminus sfr(\gamma_0)$, which satisfies (A) when $d = 0$ and s_I and t_{C_i} are not present.

Inductive hypothesis. Assume (A) is true when fewer than n steps involving cycle non-terminals are used to derive α.

Inductive step. Show (A) is true when α is derived using n steps involving cycle non-terminals.

Case 1. The derivation of α begins with $A \to \#[_2 E][_1 C_r]$ where E is in the deletion cycle, but C_r is not. We are given that $A \Rightarrow^* \alpha$, $I \Rightarrow^* \eta$, $T \Rightarrow^* \gamma_0$, $C_j \Rightarrow^* \gamma_j$, and $sfr(\alpha) = sfr(\eta) \cdot \Sigma^d \setminus sfr(\gamma_0) \cdot \Sigma^{p_0} \setminus sfr(\gamma_1) \cdot \ldots \cdot \Sigma^{p_{r-1}} \setminus sfr(\gamma_r)$. Since $\alpha = \#[_2 \beta][_1 \gamma_r]$ where $E \Rightarrow^* \beta$ and $C_r \Rightarrow^* \gamma_r$, it follows that $sfr(\beta) = sfr(\eta) \cdot \Sigma^{d} \backslash sfr(\gamma_0) \cdot \Sigma^{p_0} \backslash sfr(\gamma_1) \cdot \ldots \cdot \Sigma^{p_{r-2}} \backslash$ $sfr(\gamma_{r-1})$. The inductive hypothesis can be applied to establish that ${}_E\overline{X}_D$ contains w such that $h_1(w) = 0^d$ and $h_2(w) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_{r-1}}$. The string w is a gsm mapping of a word from $lmap(G, E, D)$. From construction step 2a, $lmap(G, A, D)$ has a rule $A \rightarrow E \cdot t_{C_r}$, and so $w \cdot t_{C_r}$ must be in ${}_A\overline{X}_D$. But $h_1(w \cdot t_{C_r}) = h_1(w) = d$ and $h_2(w \cdot t_{C_r}) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$, so (A) is satisfied.

Case 2. The derivation of α begins with $A \rightarrow \#[_2 I][_1 E]$, where E is in the cycle but I is not. In this case, $\alpha = \#[_2 \eta][_1 \beta]$ where $I \Rightarrow^* \eta$ and $E \Rightarrow^* \beta$. We are given that $sfr(\alpha) = sfr(\eta) \cdot \Sigma^{d} \backslash sfr(\gamma_0) \cdot \Sigma^{p_0} \backslash sfr(\gamma_1) \cdot \ldots$ $\cdot \Sigma^{p_{r-1}} \backslash sfr(\gamma_r)$, $T \Rightarrow^* \gamma_0$, $C_j \Rightarrow^* \gamma_j$, and $|sfr(\eta)| = i$. The nonterminals T and C_j are descendents of E in the derivation of β. The forest β may contain a leading subforest η' which is entirely deleted during the frontier operation. In that case, $sfr(\beta) = sfr(\eta') \cdot \Sigma^{d'} \backslash sfr(\gamma_0) \cdot \Sigma^{p_0} \backslash sfr(\gamma_1) \cdot \ldots$ $\cdot \Sigma^{p_{r-1}} \backslash sfr(\gamma_r)$, where $|sfr(\eta')| = i'$ and $d' = d + i' - m$ for x^m the external selector of η. If β does not contain a leading η', then $i' = 0$ and $sfr(\eta')$ will not be present in $sfr(\beta)$. The inductive hypothesis can be applied to establish that ${}_E\overline{X}_D$ produces w' such that $h_1(w') = 0^{d'}$ and $h_2(w') =$ $s_J \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$. w' is a gsm mapping on a string v' in $lmap(G, E, D)$ derived $E \Rightarrow^* B \cdot 0^p \cdot l_J \cdot t_{C_j} \cdot \ldots \cdot t_{C_r} \Rightarrow^* v'$ for $j \leqslant r$ or $E \Rightarrow B \cdot 0^p \cdot l_J \Rightarrow^* v'$. By construction step 2b, $lmap(G, A, D)$ has a rule $A \rightarrow E \cdot 0^m \cdot l_I$, and $A \Rightarrow^* B \cdot l_J \cdot t_{C_j} \cdot \ldots \cdot t_{C_r} \cdot 0^m \cdot l_I$ or $B \cdot 0^p \cdot l_J \cdot 0^m \cdot l_I$. When machine Z is applied to this string to get w, l_I, not l_J, will be moved to the front to become s_I. When M_0 scans the string, l_J will trigger the removal of i' zeros from the string. Also, the string has m more zeros than v does. So $h_1(w) = h_1(w') \cdot 0^{m-i'} = 0^{d'+m-i'} = 0^d$, $h_2(w) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$, and (A) is satisfied.

(B) If ${}_A\overline{X}_D$ produces w such that $h_1(w) = 0^d$ and $h_2(w) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$, then $A \Rightarrow^* \alpha, \ldots,$ and $sfr(\alpha) = sfr(\eta) \cdot \Sigma^d \setminus sfr(\gamma_0) \cdot \Sigma^{p_0} \setminus sfr(\gamma_1) \cdot \ldots \cdot \Sigma^{p_{r-1}} \setminus sfr(\gamma_r)$.

Proof of (B) by induction on the number of steps in the derivation of v in the grammar *lmap* (G, A, D) such that ${}_A\overline{X}_D$ contains w.

The proof of (B) is similar to the proof of (A). QED.

LEMMA 4-60. Let G be a cyclic 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ which contains a consistent deletion cycle. Suppose non-terminals A, D, and E are in the cycle, non-terminals $B, C_1, C_2, \ldots, C_r$ are outside the cycle, $I \in N$, and R has a rule $E \rightarrow \#[_2 D][_1 T]$. Then ${}_A\overline{Y}_D$ contains w such that $h_1(w) = 0^d$ and $h_2(w) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ if and only if $A \Rightarrow^* \beta, T \Rightarrow^* \gamma_0, I \Rightarrow^* \eta$, $C_j \Rightarrow^* \gamma_j$ for $1 \leqslant j \leqslant r$, and $sfr(\alpha) = sfr(\eta) \cdot \Sigma^d \setminus sfr(\gamma_0) \cdot \Sigma^{p_0} \setminus sfr(\gamma_1) \cdot \ldots \cdot \Sigma^{p_{r-1}} \setminus sfr(\gamma_r)$ where $|sfr(\eta)| = i$ and x^{p_j} is the external selector of γ_j for $0 \leqslant j \leqslant r$.

PROOF. This is the same as the previous lemma except that *tmap* is used instead of *lmap*. The two propositions involved can be proven by induction on the number of steps using cycle non-terminals which precede the application of the rule $E \rightarrow \#[_2 D][_1 T]$ in the derivation of α. The proof is very similar to that of the previous lemma, and it is not shown here. QED.

A context-free grammar G_T and an inversion map $_A\overline{X}_D$ or $_A\overline{Y}_D$ representing an inversion map can be combined to form a new context-free grammar K. The rules of G_T and $_A\overline{X}_D$ are examined simultaneously to create rules for K. Terminals in rules of G_T which match zeros in rules of $_A\overline{X}_D$ are eliminated. Also, occurrences of terminals t_C in rules of $_A\overline{X}_D$ are moved to the ends of rules in K. The grammar K is called the inversion of G_t and $_A\overline{X}_D$. As K is built, three kinds of non-terminals are used: the single non-terminals of G_T called singlets, symbols taken from $N_G \times N_X$ called doublets, and symbols from $N_G \times N_X \times N_X$ called triplets. The singlets will generate substrings of words in $L(G_T)$ which have not been subjected to deletion. The doublets will generate partially deleted words of $L(G_T)$. The triplets generate strings of terminals t_C which represent the complete deletion of a word from $L(G_T)$. The inversion process is described in the steps below. The steps are carried out in the context of a 2-d forest grammar G from which G_T and $_A\overline{X}_D$ are constructed.

1) Modify the context-free grammar $G_T = <N_G, \Sigma_G, R_G, \Sigma_G>$ so that every rule has the form $W \rightarrow a$, $W \rightarrow a \cdot U$, or $W \rightarrow a \cdot U \cdot V$ for $W, U, V \in N_G$ and $a \in \Sigma_G$.

2) Write a regular grammar $<N_X, \Sigma_X, R_X, S_X>$ for $_A\overline{X}_D$.

3) Form a set of rules $xout\,(_A\overline{X}_D, G_T)$, whose non-terminals are triplets as specified below. Assume A, B, D, E, and $J \in N_X$, T, U, V, and $W \in N_G$, # and % represent any non-terminal in N_X, t_C and $s_I \in \Sigma_X$, $a \in \Sigma_G$, and β is any

right hand side in a rule of R_G.

If R_X has	and R_G has	put in $xout\,(_A\overline{X}_D, G_T)$
a) $B \rightarrow t_C \cdot E$	$W \rightarrow \beta$	$WB\# \rightarrow t_C \cdot WE\#$
b) $B \rightarrow 0 \cdot E$	$W \rightarrow a$	$WBE \rightarrow \lambda$
c) $B \rightarrow 0 \cdot E$	$W \rightarrow a \cdot U$	$WB\# \rightarrow UE\#$
d) $B \rightarrow 0 \cdot E$	$W \rightarrow a \cdot U \cdot V$	$WB\# \rightarrow UE\% \cdot W\%\#$

4) Form a set of rules $tail\,(_A\overline{X}_D)$ according to the steps below. Assume A, B, D, E are non-terminals in N_X and $t_C \in \Sigma_X$.

a) If R_X has $B \rightarrow t_C \cdot E$, add $0BD \rightarrow t_C \cdot 0ED$.

b) If R_X has $B \rightarrow t_C \cdot D$, add $0BD \rightarrow t_C$.

5) Form a context-free grammar $K = \langle N_K, \Sigma_K, R_K, S_K \rangle = invert\,(_A\overline{X}_D, G_T)$. Σ_K will contain Σ_G and the terminals t_C from Σ_X. N_K will contain singlets, doublets, and triplets from N_G and N_X. Put $xout\,(_A\overline{X}_D, G_T)$ and $tail\,(_A\overline{X}_D)$ in K as rules for the triplets. Rules for singlets are the same as in G_T, and rules for the doublets are added according to the chart below. Assume that $T, U, V, W \in N_G$, $a \in \Sigma_G$, $0, t_C \in \Sigma_X$, and $A, B, D, E \in N_X$. The symbol $\#$ is a wildcard representing any non-terminal in N_X, and ρ represents any right-hand side in R_G. Also assume that a *code* table associates with each doublet a distinct integer greater than zero. The doublet MJ_i is a wildcard doublet which represents every doublet whose *code* is i such that $i = code(WB)$.

If R_G has	and R_X has	add to $invert\,(_A\overline{X}_D, G_T)$
a) $W \rightarrow \rho$	$B \rightarrow t_C$	$WB_0 \rightarrow W \cdot MJ_i \cdot t_C$

b) $W \to \rho$	$B \to t_C \cdot E$	$WB_j \to MJ_i \cdot t_C$
c) $W \to a$	$B \to 0$	$WB_0 \to MJ_i$
d) $W \to a \cdot U$	$B \to 0$	$WB_0 \to U \cdot MJ_i$
e) $W \to a \cdot U \cdot V$	$B \to 0$	$WB_0 \to U \cdot V \cdot MJ_i$
f) $W \to a$	$B \to 0 \cdot E$	$WB_0 \to MJ_i \cdot 0ED$
g) $W \to a \cdot U$	$B \to 0 \cdot E$	$WB_j \to MJ_i$ for $j = code(UE)$
h) $W \to a \cdot U \cdot V$	$B \to 0 \cdot E$	$WB_j \to V \cdot MJ_i$ for $j = code(UE)$
i) $W \to a \cdot U \cdot V$	$B \to 0 \cdot E$	$WB_j \to MJ_i \cdot UE\#$ for $j = code(V\#)$

j) If there is a rule $WB_j \to \sigma_1 \cdot MJ_i \cdot \sigma_2$ where $W = T$ and R_X has $A \to s_I \cdot B$, then add a rule $WB_j \to \sigma_1 \cdot \sigma_2$.

k) Add $S_K \to t_I \cdot WE_0$ for every doublet with subscript 0. Also, add $S_K \to t_I \cdot TBD$ where R_X has $A \to s_I \cdot B$.

DEFINITION 4-61. Let G be a 2DNF grammar which has a consistent deletion cycle with non-terminals A and D. Suppose also that ${}_A\overline{X}_D$ (or ${}_A\overline{Y}_D$) is an inversion map formed from G, and G has a non-terminal T whose yield can be generated by a context-free grammar G_T. Then $xout({}_A\overline{X}_D, G_T)$ is a regular grammar formed by construction step 3 above. $tail({}_A\overline{X}_D)$ is a regular grammar formed by step 4 above, and $invert({}_A\overline{X}_D, G_T)$ is a context-free grammar formed by all the steps above.

LEMMA 4-62. Suppose ${}_A\overline{X}_D$ (or ${}_A\overline{X}_D$) is a regular inversion map and G_T is a context-free grammar. $xout({}_A\overline{X}_D, G_T) = X$ has a non-terminal WBE such that $WBE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$ if and only if G_T has non-terminal W such that $W \Rightarrow^* \beta$, ${}_A\overline{X}_D$ has B such that $B \Rightarrow^* z \cdot 0 \cdot E$ for z in $\Sigma_X{}^*$, $h_1(z \cdot 0) = 0^d$, $h_2(z) = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and $|\beta| = d$.

PROOF. The lemma is established by proving propositions (A) and (B) below.

(A) If G_T has W such that . . . , then X has WBE such that $WBE \rightarrow t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Proof of (A) by induction on $|z|$.

Base. $|z| = 0$.

In this case, ${}_A\overline{X}_D$ has a rule $B \rightarrow 0 \cdot E$, $d = 1$, $h_2(z) = \lambda$. Since $d = 1$, $\beta = \alpha$ and G_T has $W \rightarrow a$. By construction step 3b, X has $WBE \rightarrow \lambda$.

Inductive hypothesis. Assume (A) is true for $0 \leqslant |z| \leqslant n$.

Inductive step. Show (A) is true for $|z| = n$.

Case 1. $z = t_{C_1} \cdot z'$ where $J \Rightarrow^* z'$.

${}_A\overline{X}_D$ has $B \rightarrow t_{C_1} \cdot J$, $B \Rightarrow^* t_{C_1} \cdot z' \cdot 0 \cdot E$, and G_T has $W \Rightarrow^* \beta$. Also, $h_1(z' \cdot 0) = h_1(z \cdot 0)$ and $h_2(z') = t_{C_1} \cdot \ldots \cdot t_{C_r}$. The inductive hypothesis can be applied to establish that *xout* $({}_A\overline{X}_D, G_T)$ contains WJE such that $WJE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$. Step 3a adds $WBE \rightarrow t_{C_1} \cdot WJE$ to X. So $WBE \Rightarrow t_{C_1} \cdot WJE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 2. $z = 0 \cdot z'$ where $J \Rightarrow^* z'$ and $\beta = a \cdot \beta'$.

xout $({}_A\overline{X}_D, G_T)$ has $B \rightarrow 0 \cdot J$ and $B \Rightarrow^* 0 \cdot z' \cdot 0 \cdot E$. $h_1(z' \cdot 0) = d - 1$, and $h_2(z') = h_2(z) = t_{C_1} \cdot \ldots \cdot t_{C_r}$. Also, there is a rule in G_T $W \rightarrow a \cdot U$ where $U \Rightarrow^* \beta'$. The inductive hypothesis can be applied to establish that *xout* $({}_A\overline{X}_D, G_T)$ contains UJE such that $UJE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$. Step 3c adds $WBE \rightarrow UJE$ to X, so $WBE \Rightarrow UJE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 3. $z = 0 \cdot z_1 \cdot z_2$ where $J \Rightarrow^* z_1$, $F \Rightarrow^* z_2$, and $\beta = a \cdot \beta_1 \cdot \beta_2$.

${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot J$, $J \Rightarrow^* z_1 \cdot F$, and $F \Rightarrow^* z_2 \cdot 0 \cdot E$. $h_1(z_1) = d = |\beta_1|$,

$h_1(z_2) = d = |\beta_2|$, and $d = 1 + d_1 + d_2$. Also, $h_2(z_1) = t_{C_1} \cdot \ldots \cdot t_{C_j}$,

$h_2(z_2) = t_{C_{j-1}} \cdot \ldots \cdot t_{C_r}$, and G_T has $W \rightarrow a \cdot U \cdot V$ where $U \Rightarrow^* \beta_1$ and

$V \Rightarrow^* \beta_2$. The inductive hypothesis can be applied to establish that

$xout\,({}_A\overline{X}_D, G_T)$ has UJF such that $UJF \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_j}$, and VFE such

that $VFE \Rightarrow^* t_{C_{j+1}} \cdot \ldots \cdot t_{C_r}$. Step 3d adds the rule $WBE \rightarrow UJF \cdot VFE$. So

$WBE \Rightarrow UJF \cdot VFE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_j} \cdot t_{C_{j+1}} \cdot \ldots \cdot t_{C_r} = t_{C_1} \cdot \ldots \cdot t_{C_r}$.

(B) If X has a non-terminal WBE such that $WBE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r} = \tau$, then G_T has non-terminal W such that $W \Rightarrow^* \beta$, ${}_A\overline{X}_D$ has B such that $B \Rightarrow^* z \cdot 0 \cdot E$ for z in $\Sigma_X{}^*$, $h_1(z \cdot 0) = 0^d$, $h_2(z) = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and $|\beta| = d$.

Proof of (B) by induction on the number of steps in the derivation of τ.

Base. WBE derives τ in one step.

τ must be λ, and X has $WBE \Rightarrow^* \lambda$. This means that ${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot E$ and G_T has $W \rightarrow a$. $h_1(0) = d$ and $h_2(0) = \lambda$, so (B) is satisfied.

Inductive hypothesis. Assume (B) is true when $WBE \Rightarrow^* \tau$ in less than n steps.

Inductive step. Show (B) is true when $WBE \Rightarrow^* \tau$ in n steps.

Case 1. $WBE \Rightarrow t_{C_1} \cdot WJE \Rightarrow^* \tau$.

Construction step 3a was applied in this case, so ${}_A\overline{X}_D$ has $B \rightarrow t_{C_1} \cdot J$ and G_T has W such that $W \Rightarrow^* \beta$. The inductive hypothesis can be applied with $WJE \Rightarrow^* t_{C_2} \cdot \ldots \cdot t_{C_r}$ to establish that ${}_A\overline{X}_D$ has $J \Rightarrow^* z' \cdot 0 \cdot E$, $h_1(z' \cdot 0) = d$, $h_2(z') = t_{C_2} \cdot \ldots \cdot t_{C_r}$, and $|\beta| = d$. So $B \Rightarrow^*$ $t_{C_1} \cdot z' \cdot 0 \cdot E = z \cdot 0 \cdot E$, $h_1(z \cdot 0) = d$, and $h_2(z) = t_{C_1} \cdot h_2(z') = t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 2. $WBE \Rightarrow UJE \Rightarrow^* \tau$.

Construction step 3c was applied to get $WBE \rightarrow UJE$. So ${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot J$ and G_T has $W \rightarrow a \cdot U$. The inductive hypothesis can be applied to establish that ${}_A\overline{X}_D$ has $J \Rightarrow^* z' \cdot 0 \cdot E$, $h_1(z' \cdot 0) = d-1$, $h_2(z') = \tau$, and $U \Rightarrow^* \beta'$ where $|\beta'| = d-1$. So $W \Rightarrow^* a \cdot \beta' = \beta$, $|\beta| = d$, $B \Rightarrow^* 0 \cdot z' \cdot 0 \cdot E = z \cdot 0 \cdot E$, $h_1(z \cdot 0) = d$, and $h_2(z) = \tau$.

Case 3. $WBE \Rightarrow UJF \cdot VFE \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_j} \cdot t_{C_{j+1}} \cdot \ldots \cdot t_{C_r} = \tau$.

Construction step 3d was applied to get $WBE \rightarrow UJF \cdot VFE$. So ${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot J$ and G_T has $W \rightarrow a \cdot U \cdot V$. The inductive hypothesis can be applied twice to establish that ${}_A\overline{X}_D$ has $J \Rightarrow^* z_1 \cdot 0 \cdot F$ and $F \Rightarrow^* z_2 \cdot 0 \cdot E$, $h_1(z_1 \cdot 0) = d_1$, $h_1(z_2 \cdot 0) = d_2$, $h_2(z_1) = t_{C_1} \cdot \ldots \cdot t_{C_j}$, $h_2(z_2) = t_{C_{j+1}} \cdot \ldots \cdot t_{C_r}$, $U \Rightarrow^* \beta_1$, $V \Rightarrow^* \beta_2$, $|\beta_1| = d_1$, and $|\beta_2| = d_2$. So $W \Rightarrow a \cdot U \cdot V \Rightarrow^* a \cdot \beta_1 \cdot \beta_2 = \beta$, $|\beta| = 1 + d_1 + d_2 = d$, $B \Rightarrow 0 \cdot J \Rightarrow^* 0 \cdot z_1 \cdot 0 \cdot z_2 \cdot 0 \cdot E = z \cdot 0 \cdot E$, $h_1(z \cdot 0) = 1 + d_1 + d_2$, and $h_2(z) = t_{C_1} \cdot \ldots \cdot t_{C_r} = \tau$. QED.

LEMMA 4-63. Let ${}_A\overline{X}_D$ (or ${}_A\overline{Y}_D$) be a regular inversion map. The set of rules *tail* $({}_A\overline{X}_D)$ has $0BD$ such that $0BD \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$ if and only if ${}_A\overline{X}_D$ has B such that $B \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_r}$ where B is a descendent of A.

PROOF. The theorem is proven by a simple induction on r, which corresponds to the number of steps in the derivation of $t_{C_1} \cdot \ldots \cdot t_{C_r}$. The induction is not shown here. QED.

LEMMA 4-64. Let G be a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ which has a consistent deletion cycle containing non-terminals A and D. Suppose that ${}_A\overline{X}_D$

is an inversion map for the deletion cycle and T is a non-terminal outside the deletion cycle whose yield is generated by context-free grammar G_T. ${}_A\overline{X}_D$ has non-terminal B such that $B \Rightarrow^* w$, $h_1(w) = 0^d$, $h_2(w) = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has a non-terminal W such that $W \Rightarrow^* \beta$, $|\beta| \geqslant d$, if and only if *invert* $({}_A\overline{X}_D, G_T)$ has a non-terminal YF_0 such that $YF_0 \Rightarrow^*$ $\Sigma^d \setminus \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ for every doublet MJ_i where $i = code(WB)$ and $YF_0 \Rightarrow^*$ $\Sigma^d \setminus \beta \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ when $W = T$ and ${}_A\overline{X}_D$ has a rule $A \rightarrow s_I \cdot B$.

PROOF. The lemma is established by proving the propositions (A) and (B) below. The proofs are written for the general case in which $W \neq T$ or A does not derive $s_I \cdot B$. The proofs of the special cases involving start symbols are the same as below except that MJ_i should be removed from the sentential forms. Construction step 5j adds terminating rules without MJ_i for start symbols T and B.

(A) If ${}_A\overline{X}_D$ has B such that $B \Rightarrow^* w$, $h_1(w) = 0^d$, $h_2(w) = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has W such that $W \Rightarrow^* \beta$, $|\beta| \geqslant d$, then *invert* $({}_A\overline{X}_D, G_T) = K$ has YF_0 such that $YF_0 \Rightarrow^*$ $\Sigma^d \setminus \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ for every MJ_i where $i = code(WB)$.

Proof of (A) by induction on $|w|$.

Base. $|w| = 1$.

Case 1. $w = t_C$.

In this case, $d = 0$ and $h_2(w) = t_C$, $W \Rightarrow^* \beta$, $|\beta| > 0$, and ${}_A\overline{X}_D$ has a rule $B \rightarrow t_C$. Step 5a of the construction adds to K a rule $WB_0 \rightarrow W \cdot MJ_i \cdot t_C$ where $i = code(WB)$. So $WB_0 \Rightarrow W \cdot MJ_i \cdot t_C \Rightarrow^* \beta \cdot MJ_i \cdot t_C =$ $\Sigma^0 \setminus \beta \cdot MJ_i \cdot t_C$.

Case 2. $w = 0$.

We have $d = 1$, $h_2(w) = \lambda$, $W \Rightarrow^* \beta$, $|\beta| > 1$, and a rule $B \rightarrow 0$ in ${}_A\overline{X}_D$. If $\beta = a$ then G_T has $W \rightarrow a$, and step 5c puts $WB_0 \rightarrow MJ_i$ in K, where $i = code(WB)$. So $WB_0 \Rightarrow \lambda \cdot MJ_i = \Sigma^1 \setminus a \cdot MJ_i = \Sigma^1 \setminus \beta \cdot MJ_i$.

If $\beta = a \cdot \beta'$ where $U \Rightarrow^* \beta'$, then G_T has $W \rightarrow a \cdot U$, and step 5d adds rule $WB_0 \rightarrow U \cdot MJ_i$, $i = code(WB)$. So $WB_0 \Rightarrow U \cdot MJ_i \Rightarrow^* \beta' \cdot MJ_i = \Sigma^1 \setminus \beta \cdot MJ_i$.

If $\beta = a \cdot \beta_1 \cdot \beta_2$ where $U \Rightarrow^* \beta_1$, and $V \Rightarrow^* \beta_2$, then G_T has $W \rightarrow a \cdot U \cdot V$, and step 5e adds rule $WB_0 \rightarrow U \cdot V \cdot MJ_i$, $i = code(WB)$. So $WB_0 \Rightarrow U \cdot V \cdot MJ_i \Rightarrow^* \beta_1 \cdot \beta_2 \cdot MJ_i = \Sigma^1 \setminus \beta \cdot MJ_i$.

Inductive hypothesis. Assume (A) is true for $|w| < n$.

Inductive step. Show (A) is true for $|w| = n$.

Case 1. $w = t_{C_1} \cdot w'$.

In this case, $h_1(w') = h_1(w) = 0^d$, $h_2(w') = t_{C_1} \cdot \ldots \cdot t_{C_r}$, ${}_A\overline{X}_D$ has $B \rightarrow t_{C_1} \cdot E$ where $E \Rightarrow^* w'$, and G_T has $W \Rightarrow^* \beta$. The inductive hypothesis establishes that *invert* $({}_A\overline{X}_D, G_T) = K$ has YF_0 such that $YF_0 \Rightarrow^* \Sigma^d \setminus \beta \cdot MJ_i \cdot t_{C_2} \cdot \ldots \cdot t_{C_r}$ where $j = code(WE)$. Step 5b adds a rule $WB_j \rightarrow MJ_i \cdot t_{C_1}$ where $i = code(WB)$. So $YF_0 \Rightarrow^* \Sigma^d \setminus \beta \cdot WB_j \cdot t_{C_2} \cdot \ldots \cdot t_{C_r} \Rightarrow \Sigma^d \setminus \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 2. $w = 0 \cdot w'$ and $\beta = a$.

In this case, ${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot E$ and $E \Rightarrow^* w'$. $h_1(w) = 1$, $h_2(w') = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has $W \rightarrow a$. Since $|\beta| \geqslant d$, d must be 1, and w' has no zeros.

So $h_2(w') = w' = t_{C_1} \cdot \ldots \cdot t_{C_r}$. Step 5c adds a rule $WB_0 \rightarrow MJ_i \cdot 0ED$ where $i = code(WB)$. Step 4 of the construction added $tail({}_A\overline{X}_D)$ to K, and lemma 4-63 establishes that $0ED$ in $tail({}_A\overline{X}_D)$ derives $t_{C_1} \cdot \ldots \cdot t_{C_r}$. So $WB_0 \Rightarrow MJ_i \cdot 0ED \Rightarrow^* MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} = \Sigma^1 \setminus \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 3. $w = 0 \cdot w'$ and $\beta = a \cdot \beta'$.

${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot E$ and $E \Rightarrow^* w'$, $h_1(w) = 0^{d-1}$, $h_2(w') = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has $W \rightarrow a \cdot U$ where $U \Rightarrow^* \beta'$, and $|\beta'| \geqslant d-1$. The inductive hypothesis establishes that $invert({}_A\overline{X}_D, G_T) = K$ has YF_0 such that $YF_0 \Rightarrow^* \Sigma^{d-1} \setminus \beta \cdot WB_j \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ where $j = code(UE)$. Step 5g adds a rule $WB_j \rightarrow MJ_i$ where $i = code(WB)$. So $YF_0 \Rightarrow^* \Sigma^{d-1} \setminus \beta' \cdot WB_j \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} \Rightarrow \Sigma^{d-1} \setminus \beta' \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} = \Sigma^d \setminus \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 4. $w = 0 \cdot w_1 \cdot w_2$, $\beta = a \cdot \beta_1 \cdot \beta_2$, and $|\beta| > d > 1 + |\beta_1|$.

${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot E$, $E \Rightarrow^* w_1 \cdot w_2$, H is a descendent of E such that $H \Rightarrow^* w_2$, $h_1(w_1) = 0^{d_1}$, $h_2(w_1) = t_{C_1} \cdot \ldots \cdot t_{C_h}$, $h_1(w_2) = 0^{d_2}$, $h_2(w_2) = t_{C_{h+1}} \cdot \ldots \cdot t_{C_r}$, $d_1 = |\beta_1|$, $d_2 \leqslant |\beta_2|$, $d = 1 + d_1 + d_2$, and G_T has $W \rightarrow a \cdot U \cdot V$ where $U \Rightarrow^* \beta_1$ and $V \Rightarrow^* \beta_2$. In this case, β_1 is completely deleted, and β_2 is at least partially deleted. The inductive hypothesis establishes that $invert({}_A\overline{X}_D, G_T) = K$ has YF_0 such that $YF_0 \Rightarrow^* \Sigma^{d_2} \setminus \beta_2 \cdot WB_j \cdot t_{C_{h+1}} \cdot \ldots \cdot t_{C_r}$ where $j = code(VH)$. Step 5i adds a rule $WB_j \rightarrow MJ_i \cdot UEH$ where $i = code(WB)$. Step 3 of the construction adds the rules $xout({}_A\overline{X}_D, G_T)$ to K. The value of d_1 must be at least 1, so it is true that $E \Rightarrow^* w_1 \cdot H = z \cdot 0 \cdot H$. $h_1(z \cdot 0) = 0^{d_1}$, and $h_2(z) = t_{C_1} \cdot \ldots \cdot t_{C_h}$. We can apply lemma 4-62 to establish that $UEH \Rightarrow^* t_{C_1} \cdot \ldots \cdot t_{C_h}$. So $YF_0 \Rightarrow^*$

$\Sigma^{d_2} \backslash \beta_2 \cdot WB_j \cdot t_{C_{h-1}} \cdot \ldots \cdot t_{C_r} \Rightarrow \Sigma^{d_2} \backslash \beta_2 \cdot MJ_i \cdot UEH \cdot t_{C_{h-1}} \cdot \ldots \cdot t_{C_r} \Rightarrow^*$

$\Sigma^{d_2} \backslash \beta_2 \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} = \Sigma^{1+d_1+d_2} \backslash a \cdot \beta_1 \cdot \beta_2 \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} =$

$\Sigma^d \backslash \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

Case 5. $w = 0 \cdot w'$ and $\beta = a \cdot \beta_1 \cdot \beta_2$, and $1 + |\beta_1| \geqslant d > 1$.

${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot E$ and $E \Rightarrow^* w'$, $h_1(w) = 0^{d-1}$, $h_2(w') = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has $W \rightarrow a \cdot U \cdot V$ where $U \Rightarrow^* \beta_1$ and $V \Rightarrow^* \beta_2$. In this case, part or all of β_1 is deleted, but none of β_2 is deleted. The hypothesis establishes that *invert* $({}_A\overline{X}_D, G_T) = K$ has YF_0 such that $YF_0 \Rightarrow^* \Sigma^{d-1} \backslash \beta_1 \cdot WB_j \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ where $j = code(UE)$. Step 5h adds a rule $WB_j \rightarrow V \cdot MJ_i$ where $i = code(WB)$. So $YF_0 \Rightarrow^* \Sigma^{d-1} \backslash \beta_1 \cdot V \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} \Rightarrow^*$

$\Sigma^{d-1} \backslash \beta_1 \cdot \beta_2 \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r} = \Sigma^d \backslash \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

(B) If *invert* $({}_A\overline{X}_D, G_T) = K$ has YF_0 such that $YF_0 \Rightarrow^* \zeta$ where $\zeta =$ $\Sigma^d \backslash \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ for every doublet MJ where $i = code(WB)$, then ${}_A\overline{X}_D$ has B such that $B \Rightarrow^* w$, $h_1(w) = 0^d$, $h_2(w) = t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has W such that $W \Rightarrow^* \beta$, $|\beta| \geqslant d$.

Proof of (B) by induction on the number of doublet rule applications in the derivation of ζ.

Base. $YF_0 \Rightarrow \zeta$ in one step.

Case 1. ζ is derived with $WB_0 \Rightarrow W \cdot MJ_i \cdot t_C \Rightarrow^* \beta \cdot MJ_i \cdot t_C$.

K contains such a rule (step 5a) only if ${}_A\overline{X}_D$ has $B \rightarrow t_C$ and G_T has a rule for W. So $B \Rightarrow^* t_C$, $d = 0$, $h_2(t_C) = t_C$, and $W \Rightarrow^* \beta$, satisfying (B).

There are similar cases for steps 5c through 5f.

Inductive hypothesis. Assume (B) is true when $YF_0 \Rightarrow^* \zeta$ in less than n steps.

Inductive step. Show (B) is true when $YF_0 \Rightarrow^* \zeta$ in n steps.

There are four cases which correspond to cases 1, 3, 4, and 5 in the inductive step of the proof of (A). Only case 3 is shown here.

Case 3. $YF_0 \Rightarrow^* \Sigma^d \setminus \beta \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$, and $WB_j \rightarrow V \cdot MJ_i$ is the last doublet rule applied, where $i = code(WB)$ and $j = code(UE)$. This rule is added by step 5h, so ${}_A\overline{X}_D$ has $B \rightarrow 0 \cdot E$ and G_T has $W \rightarrow a \cdot U \cdot V$. If one less step is taken in the derivation from YF_0, we have $YF_0 \Rightarrow^* \Sigma^d \setminus \beta_1 \cdot WB_j \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$ where $\beta = \beta_1 \cdot \beta_2$ such that $V \Rightarrow^* \beta_2$. The inductive hypothesis can be applied with this shorter derivation to get $U \Rightarrow^* \beta_1$, $E \Rightarrow^* w'$, $h_1(w') = 0^d$, and $h_2(w') = t_{C_1} \cdot \ldots \cdot t_{C_r}$. It is certainly true that $\Sigma^d \setminus \beta = \Sigma^{d+1} \setminus a \cdot \beta$. So we can write $YF_0 \Rightarrow^* \Sigma^{d+1} \setminus a \cdot \beta_1 \cdot \beta_2 \cdot MJ_i \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$. Since $U \Rightarrow^* \beta_1$, $V \Rightarrow^* \beta_2$, and there is a rule $W \rightarrow a \cdot U \cdot V$, we have $W \Rightarrow^* a \cdot \beta_1 \cdot \beta_2 = a \cdot \beta$. Also, $B \Rightarrow 0 \cdot E \Rightarrow^* 0 \cdot w' = w$, $h_1(w) = 0^{1+d}$, and $h_2(w) = t_{C_1} \cdot \ldots \cdot t_{C_r}$. Thus, (B) is satisfied. QED.

LEMMA 4-65. Let G be a 2DNF grammar with a consistent deletion cycle for which ${}_A\overline{X}_D$ is defined, and G_T is a context-free grammar which generates the yield of $T \in N$. ${}_A\overline{X}_D$ has non-terminal A such that $A \Rightarrow^* w$, $h_1(w) = 0^d$, $h_2(w) = s_I \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$, and G_T has non-terminal T such that $T \Rightarrow^* \beta$, $|\beta| \geqslant d$, if and only if *invert* $({}_A\overline{X}_D, G_T)$ has start symbol S_K such that $S_K \Rightarrow^* t_I \cdot \Sigma^d \setminus \beta \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$.

PROOF. Step 5k of the construction of *invert* (${}_A\overline{X}_D$, G_T) adds a rule $S_K \rightarrow t_T \cdot YF_0$ for each doublet with a zero subscript. Lemma 4-64 takes care of the rest. QED.

THEOREM 4-66. Let G be a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ such that all of its deletion cycles are consistent. There is a grammar $G' = <\Sigma', \Xi', N', R', S'>_2^k$ such that G' has no deletion cycles and $Y_0(G) = Y_0(G')$.

PROOF. To construct G', set $\Sigma' = \Sigma$, $\Xi' = \Xi$, and $S' = S$. For every subgrammar G_C which has no deletion cycles, put the rules and non-terminals of G_C in G'. For every G_A where A is the root of a deletion cycle, follow these steps:

1) Find every rule in G_A of the form a) $D \rightarrow T$ or b) $B \rightarrow \#[_2 D][_1 T]$ where B and D are in the deletion cycle, but T is not.

2) For each such rule, form context-free grammar G_T, regular grammar a) ${}_A\overline{X}_D$ or b) ${}_A\overline{Y}_D$, and context-free grammar a) K_i = *invert* (${}_A\overline{X}_D$, G_T) or b) K_i = *invert* (${}_A\overline{X}_D$, G_T), where i ranges from 1 to the number of rules of the form a) or b).

3) Convert each K_i to a 2DNF grammar K'_i.

4) In the rules of every K'_i, replace t_C with non-terminal C. Replace occurrences of non-terminal T in the right-hand sides of rules with $\#[_2 T][_1 x^p]$ where x^p is the external selector of T.

5) For each i, add K'_i to G' along with a rule $A \rightarrow S_{K'_i}$ where $S_{K'_i}$ is the start symbol of K'_i.

The theorem is established in the proofs of propositions (A) and (B) below.

(A) If G has non-terminal A such that $A \Rightarrow^* \alpha$ and $sfr(\alpha) = w$, then G' has A' such that $A' \Rightarrow^* \alpha'$ and $sfr(\alpha') = w$.

This can be proven by induction on the length of the derivation of α. The formal induction is not shown here, but its substance is summarized. If G_A has no deletion cycles, then it is clear that α can be derived in G' using the same rules as in G. But suppose $A \Rightarrow^* \alpha$, $sfr(\alpha) = w$, and A is the root of a deletion cycle. Then $sfr(\alpha) = sfr(\eta) \cdot \Sigma^d \backslash sfr(\gamma_0) \cdot \Sigma^{p_0} \backslash sfr(\gamma_1) \cdot \ldots \cdot \Sigma^{p_{r-1}} \backslash sfr(\gamma_r)$, where the first leading non-terminal in the cycle is I and $I \Rightarrow^* \eta$, d is the accumulated deficit of the overlapping selectors, $T \Rightarrow^* \gamma_0$, trailer non-terminals $C_j \Rightarrow^* \gamma_j$ for $1 \leqslant j \leqslant r$, and x^{p_j} is the external selector of γ_j. Lemmas 4-59, 4-60, and 4-65 establish that *invert* $({}_A\overline{X}_D, G_T) = K_i$ derives a string $s_I \cdot \Sigma^d \backslash sfr(\gamma_0) \cdot t_{C_1} \cdot \ldots \cdot t_{C_r}$. Steps 3 and 4 of the construction turn K_i into K'_i which derives $\alpha' = \#[_2 I][_1 \#[_2 \#[_2 T][_1 x^p]][_1 \#[_2 C_1][_1 \#[_2 C_2][_1 \ldots \#[_2 C_r] \ldots]]]]$ where $T \Rightarrow^* \Sigma^d \backslash sfr(\gamma_0)$. So $sfr(\alpha) = sfr(\alpha')$.

(B) If G' has a non-terminal A' such that $A' \Rightarrow^* \alpha'$ and $sfr(\alpha') = w$, then G has A such that $A \Rightarrow^* \alpha$ and $sfr(\alpha) = w$.

The proof of (B) is similar to the proof of (A). QED.

COROLLARY 4-67. Let G be a 2DNF grammar $<\Sigma, \Xi, N, R, S>_2^k$ with consistent deletion cycles. There is a context-free grammar G' such that $Y_0(G) = L(G')$.

PROOF. This follows from theorem 4-66 and theorem 4-23. QED.

The grammar ABC presented at the beginning of this chapter is an example of a 2-d grammar with a consistent deletion cycle. As a result of the previous corollary, the yield of ABC is a context-free language.

0Y2F and the Subclass $0Y2F_c$

Recall the language generated by grammar ABC. A derivation and subsequent frontier operation generates a string $a^i b^i$ for $i \geqslant 0$, and then deletes an arbitrary number of characters from the front of the string while adding c's at the end. Suppose a rule $S \rightarrow \#[_2 \#[_2 e [_1 f [_1 x]]][_1 S]][_1 d [_1 y]]$ were added to ABC to create ABC'. Then the deleting steps could be intermixed with steps that add ef to the front and d to the end of the string. The deletion cycle would no longer be consistent, and the methods above could not be used to convert ABC' to a context-free grammar. The modified deletion cycle contains mixed increasing and decreasing steps, and a general sequential machine is not be adequate to create the inversion map. This does not necessarily mean that the yield of ABC' is not context-free. If the pumping lemma for context-free languages is applied to arbitrary words of $Y_0(ABC')$, we find that every word can be successfully pumped. It seems likely that a method will be discovered for transforming inconsistent deletion cycles to context-free subgrammars. The methods employed above are inadequate for this purpose, and it remains an open question whether 0Y2F $\equiv$ context-free. We can, however, define a large subclass of 0Y2F which is context-free.

DEFINITION 4-68. A language L belongs to the class $0Y2F_c$ if and only if $L = Y_0(G)$ and G is a 2-d forest grammar such that G has no deletion cycles or every deletion cycle in G is consistent or regular.

THEOREM 4-69. $L \in 0Y2F_c$ if and only if L is context-free.

PROOF. Theorem 4-53 and corollary 4-67 establish that $L \in 0Y2F_c \Rightarrow L$ is context-free. Corollary 4-2 establishes that a context-free grammar can be converted to a yield-equivalent 2-d forest grammar. The 2-d grammar will contain only standard selectors, and thus no deletion cycles at all. So L is context-free implies $L \in 0Y2F_c$. QED.

We conclude the analysis of 2-d forest grammars by showing that 0Y2F is a proper subset of 0Y3F. First it is shown that a language known to be in 0Y3F is not in 0Y2F.

LEMMA 4-70. The language $L = \{a^{2^n}\}, n > 0$, is not in 0Y2F.

PROOF. Suppose $L \in$ 0Y2F. Then $L = Y_0(G)$ for some 2DNF grammar $G = \langle \Sigma, \Xi, N, R, S \rangle_2^k$. If $w \in L$, then $w = sfr(\beta)$ and $S \Rightarrow^* \beta$. Since L is an infinite language, there must be a increasing cycle in G. Let β_i be a forest generated from i applications of an increasing cycle step, and let $w_i = sfr(\beta_i)$. L contans a string w_i for every value of $i \geqslant 0$. Regarding the difference in size of β_i and β_{i+1}, we have $|\beta_{i+1}| - |\beta_i| = c$, where c is a constant which can be determined by examining the rules of the grammar. Observe that if β is a 2-d forest, then $|fr_1(\beta)| \leqslant |\beta|$. The 1-d frontier operation can only decrease the number of nodes in a forest. Since β_i is a subforest of β_{i+1}, it follows that $|w_{i+1}| \leqslant$

$|w_i| + c$. There are also β_{i+2} and w_{i+2} such that $|w_{i+2}| \leqslant |w_{i+1}| + c$. But now we have a contradiction. If the difference in size between any three consecutive words in L is a constant, $L \neq \{a^{2^n}\}$. QED.

COROLLARY 4-71. 0Y2F is a proper subset of 0Y3F.

PROOF. If L is in 0Y2F, then it is the yield of some 2-d grammar G. This grammar can be converted to a 3-d grammar G' with the same string yield by adding $S' \rightarrow \#[_3 \#[_2 S]]$ and all the rules of G to G', where S' is the start symbol of G' and S is the start symbol of G. So 0Y2F is a subset of 0Y3F. Theorem 79 of Baldwin's thesis shows that the language $\{a^{2^n}\}$, $n > 0$, is in ALG_3^1. Theorem 4-1 earlier in this chapter establishes that $ALG_3^1 \subseteq$ 1Y3F, so it must be true that $\{a^{2^n}\} \in$ 0Y3F. So every language in 0Y2F is in 0Y3F, but 0Y3F has a language that is not in 0Y2F (lemma 4-70). QED.

In summary, 0Y2F$_c$ is a subset of 0Y2F and is equivalent to the class of context-free languages. All 2-d forest grammars which do not contain deletion cycles yield languages in 0Y2F$_c$. Many grammars which do have deletion cycles have also been shown to yield 0Y2F$_c$ languages, provided the deletion cycles are regular or consistent. While 0Y2F may contain non-context-free languages, it is still much smaller than 0Y3F, the next class of string languages in the forest-yield hierarchy.

CHAPTER 5.

THREE-DIMENSIONAL FOREST YIELD LANGUAGES

The frontier operation on a 2-dimensional forest is the concatenation of the strings which are the frontiers of its subforests. When two strings are concatenated, there is no confusion about where to join them: the second string is attached to the end of the first. The frontier operation on a 3-d forest is a concatenation of the trees which are the yields of subforests. Again, the concatenation of two trees can be described as attaching the second tree to the end of the first. But that description is not complete, since the first tree may have several ends, or leaves. The selectors in a forest indicate where concatenation will occur and what subtrees will be attached. Two or more selectors may demand the same subtree, and in that case multiple copies of the subtree are attached. This gives 3-d forest grammars a copying power that 2-d forest grammars do not have.

Examples of 3-d Forest Grammars

A 3-d forest grammar can be written which yields the language $\{a^{2^n}\}$, $n \geqslant 0$. This grammar, called A2N, will illustrate the copying power of the frontier operation. As shown here, the grammar has only one non-terminal, $A \in N_2$. The selector set is $\Xi = \{y, z\}$, where y represents $<2, \lambda>$ and z is $<1, \lambda>$. The terminal set is $\Sigma = \{\#, a\}$, and the rules are the following:

1) $A \rightarrow \#[_3\, \#[_2\, y\, [_1\, y\,]]][_2\, A\,]$, and

2) $A \rightarrow \#[_2\, a\, [_1\, z\,]]$.

```
#---3-------2
    |       |
    #---2   #---3-------2
        |       |       |
        y---1   #---2   #---2
            |       |       |
            y       y---1   a---1
                        |       |
                        y       z
```

derived 3-d forest

```
#---2
    |
    #---2---------------1
        |               |
        #---2-------1   #---2
            |       |       |
            a---1   #---2   #---2-------1
                |       |       |       |
                z       a---1   a---1   #---2
                            |       |       |
                            z       z       a---1
                                                |
                                                z
```

2-d frontier

```
a---1
    |
    a---1
        |
        a---1
            |
            a---1
                |
                z
```

1-d frontier

FIGURE 5. 3-d forest and yields produced by A2N

Figure 5 contains a 3-d forest produced by A2N and its 2-d and 1-d frontiers. A derived forest yields the string a^{2^n}, where n is the number of times rule 1 is applied in the derivation. The derivation sequence for the forest in figure 5 is 1 1 2.

The grammar A2N has only λ-paths on its selectors, so the selectors are standard. The use of extended selectors in 3-d forest grammars does not affect the copying power of the frontier operation, but is does increase flexibility in specifying what subtrees will be copied, and it also introduces deletion cycles. This enhanced deletion power makes it possible for the frontier operation on 3-d forests to simulate list-processing operations such as the selection of an arbitrary element from a list or the division of a list into disjoint sublists. The forest $\#[_2 \#[_2 e_1][_1 \#[_2 \#[_2 e_2][_1 \cdots \#[_2 \#[_2 e_k]] \cdots]]]]$ can be used to represent a list of k elements, e_1 through e_k. The forest is structured so that a selector can extract a single element or a sublist with one or more of the leftmost elements deleted. For example, the selector $<2,22>$ extracts element e_1, and the selector $<2,21>$ extracts the rest of the list. The creation of forests which represent lists is illustrated by the 2-d grammar BLIST. The terminal set is $\Sigma = \{\#, b\}$ and the selector set is $\Xi = \{z\}$, where $z = <1,\lambda>$. There is only one non-terminal, $B \in N_2$. The rules are as follows:

1) $B \rightarrow \#[_2 \#[_2 b [_1 z]][_1 B]]$, and

2) $B \rightarrow \#[_2 \#[_2 b [_1 z]]]$.

```
#---2
    |
    #---2-------1
        |       |
        b---1   #---2
            |       |
            z       #---2-------1
                        |       |
                        b---1   #---2
                            |       |
                            z       #---2
                                        |
                                        b---1
                                            |
                                            z
```

FIGURE 6. 2-d tree produced by BLIST

Figure 6 contains a 2-d forest produced by BLIST with the derivation sequence 1 1 2. The forest represents a list of three b's.

Another example grammar, MERGE, illustrates the list-processing power of 3-d forest grammars. MERGE will have two 2-d subgrammars, ALIST and BLIST, which generate lists of a's and b's. The frontier operation on a forest produced by MERGE will have the effect of merging two lists into a single list of mixed a's and b's. The terminal set is $\Sigma = \{\#, a, b\}$, and the selector set is $\Xi = \{n, o, p, q, t, u, v, z\}$, where $n = <2, 211>$, $o = <2, 212>$, $p = <2, 2121>$, $q = <2, 22>$, $t = <2, 221>$, $u = <2, 21>$, $v = <2, 2>$, and $z = <1, \lambda>$. The non-terminals are $N_2 = \{S, M, P, Q, V, B, A\}$, where S is the start symbol. The rules are as follows:

```
#-3---2
  |   |
  #-2 #-3---2
    |   |   |
    u-1 #-2 #-3---2
      |   |   |   |
      #-2 v-1 #-2 #-2
        |   |   |   |
        v-1 p-1 t-1 #-2-------1
          |   |   |   |       |
          n   #-2 u-1 #-2---1 #-2
                |   |   |   |   |
                o-1 #-2 a-1 #-2 #-2---1
                  |   |   |   |   |   |
                  n   q   z   #-2 b-1 #-2
                                |   |   |
                                a-1 z   #-2
                                  |       |
                                  z       b-1
                                            |
                                            z
```

FIGURE 7. 3-d tree produced by MERGE

1) $S \rightarrow \#[_3\, Q\,][_2\, M\,]$,

2) $Q \rightarrow \#[_2\, u\,[_1\, \#[_2\, v\,[_1\, n\,]]]]$,

3) $Q \rightarrow \#[_2\, v\,[_1\, \#[_2\, u\,[_1\, n\,]]]]$,

4) $M \rightarrow \#[_3\, P\,][_2\, M\,]$,

5) $M \rightarrow \#[_3\, V\,][_2\, \#[_2\, A\,[_1\, B\,]]]$,

6) $P \rightarrow \#[_2\, t\,[_1\, u\,[_1\, \#[_2\, q\,[_1\, n\,]]]]]$,

7) $P \rightarrow \#[_2\, v\,[_1\, p\,[_1\, \#[_2\, o\,[_1\, n\,]]]]]$,

8) $V \rightarrow \#[_2\, t\,[_1\, u\,[_1\, \#[_2\, q\,]]]]$,

9) $V \rightarrow \#[_2\, v\,[_1\, p\,[_1\, \#[_2\, o\,]]]]$,

10) $B \rightarrow \#[_2 \#[_2 b [_1 z]][_1 B]]$,

11) $B \rightarrow \#[_2 \#[_2 b [_1 z]]]$,

12) $A \rightarrow \#[_2 \#[_2 a [_1 z]][_1 A]]$, and

13) $A \rightarrow \#[_2 \#[_2 a [_1 z]]]$.

```
#---2
    |
    #---2-------1
        |       |
        #---2   #---2
            |       |
            b---1   #---2-------1
                |       |       |
                z       #---2   #---2
                            |       |
                            a---1   #---2-------1
                                |       |       |
                                z       b---1   #---2
                                            |       |
                                            z       #---2
                                                        |
                                                        a---1
                                                            |
                                                            z
```

2-d frontier

```
b---1
    |
    a---1
        |
        b---1
            |
            a---1
                |
                z
```

1-d frontier

FIGURE 8. Frontiers of the tree in figure 7

Rules 10 and 11 constitute the subgrammar BLIST, and rules 12 and 13 form a similar subgrammar ALIST. Non-terminal M generates the list-processing cycle. The cycle is designed so that an intermediate 2-d tree of the form $\#[_2\, a\ \mathit{list}\, [_1\, b\ \mathit{list}\, [_1\, \mathit{mixed}\ \mathit{list}\ ]]]$ is maintained throughout the frontier operation. This structure is initially set up by application of rule 5, the terminating cycle step. Each application of rule 4 in the cycle corresponds to an operation on the intermediate tree when the frontier is taken. Each such cycle step introduces a subtree derived from P by rule 6 or 7. The subtree of rule 6 takes an a off the a-list and attaches it to the front of the mixed list. The subtree of rule 7 takes a b off the b-list and attaches it to the mixed list. The application of rule 1 corresponds to the final step of the frontier operation. This step attaches the remainder of the a-list and the remainder of the b-list as the first two elements of the mixed list. Figure 7 contains a 3-d tree derived from S, and figure 8 contains its 2-d and 1-d frontiers. The derivation sequence for the tree in figure 7 is 1 2 4 7 5 8 10 11 12 13.

In list-processing applications, the need sometimes arises to turn a complex list into a simple list. A complex list is one whose elements can themselves be lists. The grammar A2N produces forests whose 2-d yields can be considered to represent complex lists of a's. Figure 9 shows complex and simple list representations with 2-d trees. A 3-d grammar SQUASH can be written which creates complex lists of a's as 2-d subtrees in its derived forests. The frontier operation then transforms the complex lists into simple lists. SQUASH has terminal set $\Sigma = \{\#, a\}$ and selector set $\Xi = \{s, t, u, v, w, x, y, z\}$, where $s = <2, 2221>$,

```
      #---2
          |
          #---2----------------1
              |                |
              #---2--------1   #---2
                  |        |       |
                  a---1    #---2   #---2--------1
                      |        |       |        |
                      z        a---1   a---1    #---2
                                   |       |        |
                                   z       z        a---1
                                                        |
                                                        z
```

forest representing a complex list

```
#---2
    |
    #---2--------1
        |        |
        a---1    #---2
            |        |
            z        #---2--------1
                         |        |
                         a---1    #---2
                             |        |
                             z        #---2--------1
                                          |        |
                                          a---1    #---2
                                              |        |
                                              z        #---2
                                                           |
                                                           a---1
                                                               |
                                                               z
```

forest representing a simple list

FIGURE 9. Tree representations of lists

$t = <2, 221>$, $u = <2, 21>$, $v = <2, 2>$, $w = <2, 22>$, $x = <2, 222>$,

$y = <2, \lambda>$, and $z = <1, \lambda>$. The non-terminal set is $N_2 = \{R, W, T, A\}$ with start symbol R. The rules are the following:

1) $R \rightarrow \#[_3\, T\,][_2\, R\,]$,

2) $R \rightarrow \#[_3\, \#[_2\, u\,[_1\, \#[_2\, v\,]]]][_2\, W\,]$,

3) $T \rightarrow \#[_2\, t\,[_1\, \#[_2\, w\,[_1\, u\,]]]]$,

4) $T \rightarrow \#[_2\, \#[_2\, x\,[_1\, \#[_2\, s\,[_1\, t\,]]]][_1\, u\,]]$,

5) $W \rightarrow \#[_3\, \#[_2\, w\,[_1\, \#[_2\, t\,[_1\, u\,]]]]][_2\, W\,]$,

6) $W \rightarrow A$,

7) $A \rightarrow \#[_3\, \#[_2\, y\,[_1\, y\,]]][_2\, A\,]$, and

8) $A \rightarrow \#[_2\, a\,[_1\, z\,]]$.

```
#--3-----2
  |      |
   #--2  #--3-----2
      |     |     |
      t--1  #--2  #--3-----2
         |     |     |     |
         #--2  t--1  #--2  #--3-----2
            |     |     |     |     |
            w--1  #--2  u--1  #--2  #--3-----2
               |     |     |     |     |     |
               u     w--1  #--2  w--1  #--2  #--3-----2
                        |     |     |     |     |     |
                        u     v     #--2  y--1  #--2  #--2
                                       |     |     |     |
                                       t--1  y     y--1  a--1
                                          |           |     |
                                          u           y     z
```

FIGURE 10. 3-d forest produced by SQUASH

Rules 7 and 8 constitute the subgrammar A2N. The non-terminal R generates the list-processing cycle. Rule 2 is the terminating cycle step. An intermediate 2-d tree of the form $\#[_2$ *old list* $[_1$ *new list* $]]$ is maintained throughout the cycle. Each time rule 1 is applied in the cycle, a non-terminal T is introduced. T is expanded by rule 3 or rule 4. Whenever a subtree produced by rule 3 is encountered during the frontier operation, the first element of the old list is moved to the front of the new list. Whenever a subtree produced by rule 4 is encountered, the first element of the old list is split into two elements and the new list is unchanged. If the first element of the old list is not itself a list in this situation, a path error will occur. The non-terminal W generates a cycle which repeatedly divides the first element of the old list into two pieces. When the first element is a simple element, application of rule 2 makes this element the initial new list, and the intermediate tree is in the proper form for application of the R cycle. Figure 10 contains a 3-d forest produced by SQUASH and figure 11 shows its 2-d frontier. The derivation sequence for the forest is 1 3 1 3 2 5 6 7 7 8. The old list is partially or completely simplified during the frontier operation, but no a's are added or deleted in the process. The resulting 2-d forest has the same number of a's as the 2-d forest derived from non-terminal A.

The four example grammars just introduced can be composed to form a grammar which will help relate 0Y3F to other known classes of languages. The grammar BA2N might be described as MERGE (SQUASH (A2N), BLIST). It yields the subset of $\{a \mid b\}^+$ such that each string in the subset has 2^n a's, $n > 0$, and an

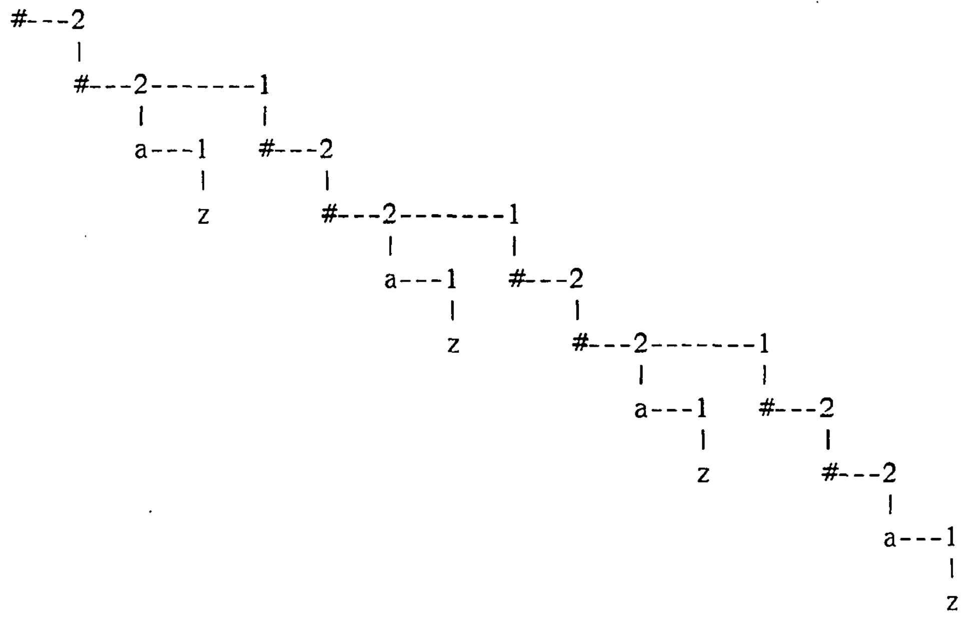

FIGURE 11. 2-d frontier of forest in figure 10

arbitrary number of b's. The terminal set of BA2N is $\Sigma = \{\#, a, b\}$, and the selector set is $\Xi = \{n, o, p, q, s, t, u, v, w, x, y, z\}$, where the selectors are defined as in the preceding examples. The non-terminal set is $N_2 = \{S, M, P, Q, R\ T, V, W, A, B\}$, with start symbol S. The rules are as follows:

1) $S \rightarrow \#[_3 Q][_2 M]$,

2) $Q \rightarrow \#[_2 u [_1 \#[_2 v [_1 n]]]]$,

3) $Q \rightarrow \#[_2 v [_1 \#[_2 u [_1 n]]]]$,

4) $M \rightarrow \#[_3 P][_2 M]$,

5) $M \rightarrow \#[_3 V][_2 \#[_2 R [_1 B]]]$,

6) $P \rightarrow \#[_2 t [_1 u [_1 \#[_2 q [_1 n]]]]]$,

7) $P \rightarrow \#[_2 v [_1 p [_1 \#[_2 o [_1 n]]]]]$,

8) $V \rightarrow \#[_2 t [_1 u [_1 \#[_2 q]]]]$,

9) $V \rightarrow \#[_2 v [_1 p [_1 \#[_2 o]]]]$,

10) $R \rightarrow \#[_3 T][_2 R]$,

11) $R \rightarrow \#[_3 \#[_2 u [_1 \#[_2 v]]]][_2 W]$,

12) $T \rightarrow \#[_2 t [_1 \#[_2 w [_1 u]]]]$,

13) $T \rightarrow \#[_2 \#[_2 x [_1 \#[_2 s [_1 t]]]][_1 u]]$,

14) $W \rightarrow \#[_3 \#[_2 w [_1 \#[_2 t [_1 u]]]]][_2 W]$,

15) $W \rightarrow A$,

16) $A \rightarrow \#[_3 \#[_2 y [_1 y]][_2 A]$,

17) $A \rightarrow \#[_2 a [_1 z]]$,

18) $B \rightarrow \#[_2 \#[_2 b [_1 z]][_1 B]]$, and

19) $B \rightarrow \#[_2 \#[_2 b [_1 z]]]$.

The yield of BA2N is a language which is not an IO macro language. It can be used to prove that 1Y3F is larger than ALG_3^1.

THEOREM 5-1. ALG_3^1 is a proper subset of 1Y3F.

PROOF. Theorem 4-1 gives us $ALG_3^1 \subseteq$ 1Y3F. Fischer (1968) has shown that the language of the preceding example, L(BA2N), is an OI macro language which is not an IO macro language. Baldwin (1983) has shown the equivalence of ALG_3^1 with the IO macro languages. So L(BA2N) is in 1Y3F, but not in ALG_3^1. QED.

Three-Dimensional Normal Form

The goal of this chapter is to identify a subclass of 0Y3F which is larger than the IO macro languages, but still within the context-sensitive languages. A normal form can be defined for 3-d forest grammars such that every language in 0Y3F is $Y_0(G)$ where G is a normal form grammar. The normal form prevents the generation of some non-frontierable forests. The first step in transforming a 3-d grammar to normal form is conversion to a short-rule grammar.

DEFINITION 5-2. A 3-d forest grammar $G = <\Sigma, \Xi, N, R, S>_3^k$ is a *short-rule* grammar if and only if every rule in R fits one of the following forms for $A, B, C, E \in N$, and $a \in \Sigma \bigcup \Xi$:

1) $A \to a[_3 B][_2 C][_1 D]$, 5) $A \to a[_2 C][_1 D]$,
2) $A \to a[_3 B][_2 C]$, 6) $A \to a[_2 C]$,
3) $A \to a[_3 B][_1 C]$, 7) $A \to a[_1 D]$, or
4) $A \to a[_3 B]$, 8) $A \to a$.

LEMMA 5-3. Let G be a grammar $<\Sigma, \Xi, N, R, S>_3^k$. There is a short-rule grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ such that $L(G) = L(G')$.

PROOF. The construct method for G' is quite similar to the method in the proof of lemma 4-6, so it will only be summarized here. Only non-terminals can be inside brackets in the right-hand sides of rules, so it G has a rule $A \to \alpha[_r \beta]$ where β is not a non-terminal, a new non-terminal T is invented and $A \to \alpha[_r \beta]$ is replaced by $A \to \alpha[_r T]$ and $T \to \beta$. If G has a rule with a non-terminal outside of brackets, $A \to B[_r \alpha]$, then put the rules for B in short-rule form and replace $A \to B[_r \alpha]$ with $A \to \beta_1[_r \alpha], \ldots, A \to \beta_m[_r \alpha]$ where $\beta_1, \ldots, \beta_m$ are the right-hand sides of rules for B. This process involves only textual substitution of non-terminals and right-hand sides of rules. The forests derived from the grammar are not changed. So $L(G) = L(G')$. QED.

The transformation from short-rule to normal form is made by modifying or removing certain kinds of rules. Any rules containing a selector in Ξ_3 can be removed. Rules of the form $A \rightarrow a[_3 B]$ can be eliminated unless A is the start symbol, and rules like $A \rightarrow a[_3 B][_2 C][_1 D]$ can be replaced by virtue of the next lemma.

LEMMA 5-4. Suppose there are α, β, γ, δ, and ζ in $H_3^1(\Sigma, \Xi)$ such that $\alpha =$ $\#[_3 \gamma][_2 \delta][_1 \zeta]$ and $\beta = \#[_3 \#[_3 \gamma][_2 \delta]][_1 \zeta]$. Then $fr_2(\alpha) = fr_2(\beta)$.

PROOF. This lemma follows from the definition of the frontier function:

$$\begin{aligned} fr_2(\alpha) &= fr_2(\#[_3 \gamma][_2 \delta])[_1 fr_2(\zeta)] \\ &= fr_2(\#[_3 \#[_3 \gamma][_2 \delta]])[_1 fr_2(\zeta)] \\ &= fr_2(\#[_3 \#[_3 \gamma][_2 \delta]][_1 \zeta]) \\ &= fr_2(\beta). \end{aligned}$$

QED.

DEFINITION 5-5. Let G be a grammar $<\Sigma, \Xi, N, R, S>_3^k$. G is in *3-dimensional normal form*, 3DNF, if and only if every rule in R fits one of the forms below. Assume that $B \in N_2$, $C, D \in N_1 \bigcup N_2$, $a \in \Sigma$, $b \in \Sigma \bigcup \Xi_1$, and $c \in \Sigma \bigcup \Xi_1 \bigcup \Xi_2$.

1) $S \rightarrow a[_3 B]$ for $S \in N_3$,

2) $A \rightarrow a[_3 B][_2 C]$ where $A \in N_1 \bigcup N_2$,

3) $A \rightarrow a[_2 C]$ where $A \in N_1 \bigcup N_2$,

4) $A \rightarrow c$ where $A \in N_1 \bigcup N_2$,

5) $A \rightarrow a[_3 B][_1 C]$ where $A \in N_1$,

6) $A \rightarrow a[_2 C][_1 D]$ where $A \in N_1$,

7) $A \rightarrow b[_1 C]$ where $A \in N_1$.

The next lemma and theorem complete the conversion of a short-rule grammar to 3DNF. The construction in the proof of the lemma shows how to eliminate nodes which have three subtrees. The final step in the proof of the theorem is the removal of unit productions (of the form $A \rightarrow B$).

LEMMA 5-6. Let G be a short-rule grammar $<\Sigma, \Xi, N, R, S>_3^k$. There is a grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ such that every rule in R' is either 3DNF or $A \rightarrow B$ where A , $B \in N_2$, and $Y_2(G) = Y_2(G')$.

PROOF. Construct G' by the following steps:

1) Set $\Sigma' = \Sigma$, $\Xi' = \Xi_1 \bigcup \Xi_2$, and $N' = N \bigcup \{S'\}$.

2) If R has $S \rightarrow a[_3 A]$ where S is the start symbol of G, put $S' \rightarrow \#[_3 S]$ in R' and S' in N'_3. Otherwise add a rule $S' \rightarrow S$, and put S' in N'_i where $S \in N_i$.

3) If R has $A \rightarrow a[_3 B]$, remove it and put $A \rightarrow B$ in R', and move A to N'_2.

4) If R has a rule which contains an element of Ξ_3, remove it from R.

5) If R has a rule $A \rightarrow a[_3 B][_2 C][_1 D]$, remove it from R and put $A \rightarrow a[_3 T][_1 D]$ and $T \rightarrow \#[_3 B][_2 C]$ in R'. Add newly invented non-terminal T to N'_2.

6) Add the remaining rules of R to R'.

The lemma is established by proving propositions (A) and (B) below.

(A) If G contains non-terminal A such that $A \Rightarrow^* \alpha$ and $fr_2(\alpha) = \beta$, then G' has A such that $A \Rightarrow^* \alpha'$, and $fr_2(\alpha') = \beta$.

Proof of (A) by induction on the number of steps in the derivation of α.

Base. $A \Rightarrow \alpha$ in one step.

There must be a rule $A \rightarrow a$ in R. If $a \in \Xi_3$ then $fr_2(a)$ is undefined, and R' has no corresponding rule. If $a \in \Sigma \bigcup \Xi_1 \bigcup \Xi_2$, then R' has the same rule.

Inductive hypothesis. Assume (A) is true when $A \Rightarrow^* \alpha$ in $n - 1$ steps.

Inductive step. Show (A) is true when $A \Rightarrow^* \alpha$ in n steps.

Case 1. The first derivation step is $A \rightarrow a\,[_3\, B\,]$.

In this case, $\alpha = a\,[_3\, \gamma]$ where $B \Rightarrow^* \gamma$ and $fr_2(\alpha) = fr_2(\gamma) = \beta$. The inductive hypothesis establishes that G' also has B such that $B \Rightarrow^* \gamma'$ and $fr_2(\gamma') = \beta$. Construction step 3 puts a rule $A \rightarrow B$ in R', so $A \Rightarrow B \Rightarrow^* \gamma'$ in G' and $fr_2(\gamma') = \beta$.

Case 2. The first step is $A \rightarrow a\,[_3\, B\,][_2\, C\,][_1\, D\,]$.

We have $\alpha = a\,[_3\, \zeta][_2\, \gamma][_1\, \delta]$ where $B \Rightarrow^* \zeta$, $C \Rightarrow^* \gamma$, and $D \Rightarrow^* \delta$. The inductive hypothesis can be applied to establish that G' has B, C, and D such that $B \Rightarrow^* \zeta'$, $C \Rightarrow^* \gamma'$, $D \Rightarrow^* \delta'$, $fr_2(\zeta') = fr_2(\zeta)$, $fr_2(\gamma') = fr_2(\gamma)$, and $fr_2(\delta') = fr_2(\delta)$. Step 5 of the construction adds to R' the rules $A \rightarrow a\,[_3\, T\,][_1\, D\,]$ and $T \rightarrow \#[_3\, B\,][_2\, C\,]$. So in G', $A \Rightarrow^* \alpha' = a\,[_3\, \#[_3\, \zeta][_2\, \gamma]][_1\, \delta]$, and $fr_2(\alpha') = fr_2(\alpha)$ according to lemma 5-4.

Case 3. The first step is $A \rightarrow \sigma$, where σ is $a\,[_3\, B\,][_2\, C\,]$, $a\,[_3\, B\,][_1\, C\,]$, $a\,[_2\, B\,][_1\, C\,]$, $a\,[_2\, B\,]$, or $a\,[_1\, B\,]$.

For each situation, apply the inductive hypothesis on the forests derived from

B and C, and observe that R' has the same rule for A and R does by construction step 6.

To conclude the proof of (A), note that if there is $S \Rightarrow^* \alpha$ such that $fr_2(\alpha) = \beta$ in G, then $S \Rightarrow^* \alpha'$ such that $fr_2(\alpha') = \beta$ is in G', and step 2 has added to G' a rule $S' \rightarrow \#[_3 S]$ or $S' \rightarrow S$. So the start symbol of G' also derives a forest whose 2-d frontier is β.

(B) If G' has A such that $A \Rightarrow^* \alpha'$, $fr_2(\alpha') = \beta$, and $A \in N$, then G has A such that $A \Rightarrow^* \alpha$ and $fr_2(\alpha) = \beta$. If G' has T such that $T \Rightarrow^* \alpha'$, $fr_2(\alpha') = \beta$, T is not in N, and $T \neq S'$, then G has A such that $A \Rightarrow^* a[_3 B][_2 C][_1 D]$ and $fr_2(a[_3 \zeta[[_2 \gamma]) = \beta$, where $B \Rightarrow^* \zeta$ and $C \Rightarrow^* \gamma$. If $T = S'$, then G has $S \Rightarrow^* \alpha$ and $fr_2(\alpha) = \beta$.

Proof of (B) by induction on the number of steps in the derivation of α'.

The proof is similar to the proof of (A). QED.

THEOREM 5-7. Let G be a grammar $<\Sigma, \Xi, N, R, S>_3^k$. There is a 3DNF grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ such that $Y_2(G) = Y_2(G')$.

PROOF. Apply the constructions of lemma 5-3 and lemma 5-6 to G, and then eliminate productions of the form $A \rightarrow B$ by the conventional substitution method. The result is G' such that G' is 3DNF and $Y_2(G') = Y_2(G)$. QED.

3-d Increasing Grammars

If a 3-d forest grammar yields a string language that is within the context-sensitive class, then there must be a linear relationship between the length of a yielded word w and the number of derivation steps required to produce a forest which yields w. If a grammar G is in 3DNF, then each rule application introduces exactly one terminal or selector. The number of steps required to derive a 3-d forest β in $L(G)$ is the same as $|\beta|$, the number of nodes in β. A linear relationship between $|\alpha|$ and $|\beta|$ where $A \Rightarrow^* \beta$ and $fr_2(\beta) = \alpha$ will be shown first for 3-d grammars which are increasing. A grammar will be called increasing if the 2-d yield grows with each step of the frontier operation on every derived forest.

DEFINITION 5-8. Let G be a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$. G is an *increasing* grammar if and only if $|fr_2(\beta)| > |fr_2(\gamma)|$ and $|fr_2(\beta)| > |fr_2(\delta)|$ for every $\beta = \#[_3\, \gamma][_2\, \delta]$ such that $A \Rightarrow^* \beta$ and $A \in N$.

The requirement that $|fr_2(\beta)|$ be greater than $|fr_2(\gamma)|$ in the above definition is not too stringent, since it is always the case that $|fr_2(\beta)| \geqslant |fr_2(\gamma)|$. The other requirement, that $|fr_2(\beta)| > |fr_2(\delta)|$, is more significant. As a result of this restriction, an increasing 3-d grammar has no decreasing cycles. The next lemma shows a non-linear relationship between the size of a derived forest and its 2-d frontier for increasing grammars.

THEOREM 5-9. Suppose G is an increasing 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ which has non-terminal $A \in N$. If $\alpha \in Y_2(G_A)$ and $|\alpha| = l$, then there is a forest β such that $A \Rightarrow^* \beta$, $fr_2(\beta) = \alpha$, $|\beta| \leqslant 2^l - 1$ when $A \in N_1 \bigcup N_2$, and

$|\beta| \leqslant 2^l$ when $A \in N_3$.

PROOF. The theorem can be proven by induction on $|\alpha| = l$.

Base. $l = 1$.

If $A \in N_2$ or $A \in N_1$, then it must be the case that $\alpha = c$ and G has a rule $A \rightarrow c$ where $c \in \Sigma \bigcup \Xi_1 \bigcup \Xi_2$. So $\alpha = \beta = C$ and $|\alpha| = 1 = 2^1 - 1 = |\beta|$. If $A \in N_3$, then the derivation starts $A \rightarrow a\,[_3\, B\,]$ where $B \in N_2$ and $B \Rightarrow^* \beta'$. The case of $B \in N_2$ was just considered, showing $|\beta'| = 1$. So $|\beta| = |a\,[_3\, \beta']| = 2^1$.

Inductive hypothesis. Assume the lemma is true for $l < n$.

Inductive step. Show the lemma is true from $l = n$.

Case 1. $\alpha = a\,[_1\, \alpha']$.

In this case, β must be derived $A \rightarrow a\,[_1\, C\,]$ where $A \in N_1$, $C \Rightarrow^* \beta'$, $C \in N_1 \bigcup N_2$, and $fr_2(\beta') = \alpha'$. By the inductive hypothesis, $|\beta'| \leqslant 2^{l-1} - 1$. But then $|\beta| = 1 + 2^{l-1} - 1 = 2^{l-1} \leqslant 2^l - 1$.

Case 2. $\alpha = a\,[_2\, \alpha_1][_1\, \alpha_2]$.

β can be $b\,[_3\, \beta_1][_1\, \beta_2]$ or $a\,[_2\, \beta_1][_1\, \beta_2]$. If β is $b\,[_3\, \beta_1][_1\, \beta_2]$, then it must be derived by $A \rightarrow b\,[_3\, B\,][_1\, C\,]$ where $B \Rightarrow^* \beta_1$, $C \Rightarrow^* \beta_2$, $B \in N_2$, and $C \in N_1 \bigcup N_2$. It must also be true that $fr_2(\beta_2) = \alpha_2$ and $fr_2(\beta_1) = a\,[_2\, \alpha_1]$. The inductive hypothesis can be applied to establish that $|\beta_1| \leqslant 2^{l-1} - 1$ and $|\beta_2| \leqslant 2^{l-2} - 1$. But then

$$\begin{aligned} |\beta| &= 1 + |\beta_1| + |\beta_2| \\ &\leqslant 1 + 2^{l-1} - 1 + 2^{l-2} - 1 \end{aligned}$$

$$\leqslant 2^{l-1} + 2^{l-2} - 1$$
$$\leqslant 2^{l-1} + 2^{l-1} - 1$$
$$\leqslant 2 \times 2^{l-1} - 1$$
$$\leqslant 2^{l} - 1.$$

If β is $a[_2\,\beta_1][_1\,\beta_2]$, it is derived $A \rightarrow a[_2\,B][_1\,C]$ where $B, C \in N_1 \bigcup N_2$, $B \Rightarrow^* \beta_1$, and $C \Rightarrow^* \beta_2$. It must be true that $fr_2(\beta_1) = \alpha_1$ and $fr_2(\beta_2) = \alpha_2$, so the inductive hypothesis is applied to give $|\beta_1| \leqslant 2^{l-1} - 1$ and $|\beta_2| \leqslant 2^{l-1} - 1$. But then

$$|\beta| = 1 + |\beta_1| + |\beta_2|$$
$$\leqslant 1 + 2^{l-1} + 2^{l-2} - 1$$
$$\leqslant 2 \times 2^{l-1} - 1$$
$$\leqslant 2^{l} - 1.$$

Case 3. $\alpha = a[_2\,\alpha']$.

In this case, β can be $a[_2\,\beta']$, $b[_3\,\beta_1][_2\,\beta_2]$, or $b[_3\,\beta']$. If β is $a[_2\,\beta']$, then it must be derived using $A \rightarrow a[_2\,B]$ where $B \Rightarrow^* \beta'$, $B \in N_1 \bigcup N_2$, and $fr_2(\beta') = \alpha'$. The inductive hypothesis establishes that $|\beta'| \leqslant 2^{l-1} - 1$. But then $|\beta| = 1 + 2^{l-1} - 1 = 2^{l-1} < 2^{l} - 1$.

If β is $b[_3\,\beta_1][_2\,\beta_2]$, it is derived $A \rightarrow b[_3\,B][_2\,C]$ for $B, C \in N_1 \bigcup N_2$, $B \Rightarrow^* \beta_1$ and $C \Rightarrow^* \beta_2$. $fr_2(\beta) = \alpha$, so it must be true that $fr_2(\beta_1) = \alpha_1$, $fr_2(\beta_2) = \alpha_2$, and $\alpha = subs_2(\alpha_1, \alpha_2)$. Since G is increasing, $|\alpha| > |\alpha_1|$ and $|\alpha| > |\alpha_2|$. The inductive hypothesis can be applied to establish that $|\beta_1| \leqslant 2^{l-1} - 1$ and $|\beta_2| \leqslant 2^{l-1} - 1$. But then

$$|\beta| = 1 + |\beta_1| + |\beta_2|$$

$\leqslant 1 + 2^{l-1} + 2^{l-2} - 1$

$\leqslant 2 \times 2^{l-1} - 1$

$\leqslant 2^l - 1.$

If β is $b[_3\, \beta']$, then it is derived with $A \rightarrow b[_3\, B]$ where $B \Rightarrow^* \beta'$ and $B \in N_2$. By the previous discussion, $|\beta'| \leqslant 2^l - 1$. So $|\beta| \leqslant 2^l$. QED.

As a result of the previous theorem, we can determine whether an increasing 3-d grammar G yields a 2-d forest β by deriving all the forests of size less than $2^{|\beta|}$ and then taking their 2-d frontiers. This will be helpful in improving the non-linear relationship between $Y_3(G)$ and $Y_2(G)$ to a linear relationship.

COROLLARY 5-10. Suppose G is an increasing grammar $<\Sigma, \Xi, N, R, S>_3^k$, and let F_c be the set of forests such that $\alpha \in F_c$ only when $\alpha \in Y_2(G)$ and $|\alpha| \leqslant c$ for some constant c. There is a 2-d grammar G' such that $L(G') = F_c$.

PROOF. As a result of theorem 5-9 above, $Y_2(G)$ is a recursive set. To enumerate the forests in $Y_2(G)$ with size less than or equal to c, we just generate all the 3-d trees in $L(G)$ of size $\leqslant 2^c - 1$ and take their frontiers. This set F_c is finite for fixed c, so we can certainly write a 2-d regular forest grammar to derive its forests. There could be a separate rule for every forest in the language. QED.

Grammars with Deletion Constants

With the addition of another constraint on a grammar, the non-linear relationship in theorem 5-9 between the size of a derived forest and the size of its 2-d frontier can be replaced with a linear relationship. The new constraint is called the *deletion constant.* It places a constant upper bound on the number of symbols that are deleted by any single step of the frontier operation. Note that the deletion constant does not limit the total number of symbols deleted during the frontier operation, since an arbitrary number of steps can occur. In 2DNF grammars, the deletion constant is no larger than the longest path on any selector. This is also true with 3DNF grammars for which no entire subforests are thrown away during the frontier operation. Even when a 3-d grammar deletes entire subforests, it will have a deletion constant if the size of the deleted subforest in any single frontier step is independent of the size of the subforest which is not deleted.

DEFINITION 5-11. Let G be a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$. The *deletion constant* of G is the smallest constant c such that $|fr_2(\beta)| \geqslant |fr_2(\gamma)| + |fr_2(\delta)| - c$ for every α in $Y_2(G)$ and β in $L(G)$ where $\beta = a[_3\, \gamma][_2\, \delta]$ is the smallest forest such that $fr_2(\beta) = \alpha$. If no such constant can be found, then G has no deletion constant.

Given any constant c, a grammar can be written so that any forest whose 2-d yield has less than c nodes can be derived without introducing and 3-nodes or 3-2 nodes. Such a grammar is said to be c-augmented.

DEFINITION 5-12. A 3DNF grammar G is *c-augmented* if it is increasing, has deletion constant c, and every $\alpha \in Y_2(G)$ such that $|\alpha| \leqslant c$ is derivable by a 2-d subgrammar of G.

LEMMA 5-13. If G is an increasing, 3DNF grammar with deletion constant c, then there is a c-augmented 3DNF grammar G' such that $Y_2(G) = Y_2(G')$.

PROOF. Put all the rules of G in G'. If A yields α in G and $|\alpha| \leqslant c$, then add 2-d rules to G' for A to derive α directly, using the method described in the proof of corollary 5-10. Certainly $Y_2(G) = Y_2(G')$, since G' has all the rules of G, and the only rules added derive forests which are in $Y_2(G)$. Also, any $\alpha \in Y_2(G)$ is derivable in G' without the use of any 3-d rules. QED.

LEMMA 5-14. Suppose G is a c-augmented 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$. If $\alpha \in Y_2(G_A)$ where $A \in N$ and $|\alpha| = l$, then there is a β such that $A \Rightarrow^* \beta$, $fr_2(\beta) = \alpha$, and $|\beta| = d$ where

$$d \leqslant \begin{cases} l & \text{for } l \leqslant c, \\ 2(c+1)(l-c)-1 & \text{for } l > c \text{ and } A \in N_1 \bigcup N_2, \text{ and} \\ 2(c+1)(l-c) & \text{for } l > c \text{ and } A \in N_3 \end{cases}$$

PROOF. Proof by induction on $l = |\alpha|$.

Base. $l = 1$.

Case 1. $l \leqslant c$.

In this case, α is derivable without 3-d steps. $A \Rightarrow \alpha = \beta = fr_2(\beta)$. So $|fr_2(\beta)| = |\alpha| = l$.

Case 2. $l > c$ and $A \in N_1 \bigcup N_2$.

The only possible derivation of α is $A \Rightarrow a = \alpha$, where $a \in \Sigma \bigcup \Xi_1 \bigcup \Xi_2$.

Since l is 1, c must be 0. So $\beta = fr_2(\beta) = \alpha$, $|\beta| = 1$, and

$$\begin{aligned} 1 &\leqslant 2(c+1)(l-c)-1 \\ &\leqslant 2(0+1)(l-0)-1 \\ &\leqslant 2-1 \\ &\leqslant 1. \end{aligned}$$

Case 3. $l > c$ and $A \in N_3$.

The derivation of β starts with $A \rightarrow a[_3 B]$, $B \in N_2$, $B \Rightarrow^* \beta'$, and $fr_2(\beta') = \alpha$. Case 2 above establishes that $|\beta'| = 1 = 2(c+1)(l-c)-1$. So $|\beta| = 1 + |\beta'| = 2(c+1)(l-c)$.

Inductive hypothesis. Assume the lemma is true for $1 < l < n$.

Inductive step. Show the lemma is true for $l = n$.

Case 1. $\alpha = a[_1 \alpha']$.

The derivation of β must be $A \Rightarrow a[_1 C] \Rightarrow^* a[_1 \beta']$, where $A \in N_1$, $fr_2(\beta') = \alpha'$, and $C \in N_1 \bigcup N_2$. If $l-1 \leqslant c$, then $|\beta'| = l-1$ and $|\beta| = l$. This satisfies the lemma if $l \leqslant c$. If $l = c+1$, then we need to verify that $|\beta| = l \leqslant (2c+2)(l-c)-1 \leqslant 2l-1$. But this is true for $l \geqslant 1$.

If $l-1 > c$, then the inductive hypothesis gives $|\beta'| \leqslant (2c+2) \times (l-1-c)-1$. So

$$\begin{aligned} |\beta| &\leqslant |\beta'| + 1 \\ &\leqslant (2c+2)(l-1-c) \\ &\leqslant 2cl - 2c^2 + 2l - 2c - 2 - 2c \\ &\leqslant 2cl - 2c^2 + 2l - 2c - 1 \\ &\leqslant (2c+2)(l-c)-1. \end{aligned}$$

Case 2. $\alpha = a[_2 \alpha_1][_1 \alpha_2]$ and $\beta = b[_3 \beta_1][_1 \beta_2]$.

β is derived with a rule $A \rightarrow b[_3 B][_1 C]$ where $A, C \in N_1 \bigcup N_2$, $B \in N_2$, $B \Rightarrow^* \beta_1$, and $C \Rightarrow^* \beta_2$. It must also be true that $fr_2(\beta_2) = \alpha_2$ and $fr_2(\beta_1) = a[_2 \alpha_1]$. Let $l = l_1 + l_2$ where $l_1 = |a[_2 \alpha_1]|$ and $l_2 = |\alpha_2|$. The inductive hypothesis can be applied to determine the maximum size of β_1 and β_2. There are four possible situations, depending on the values of l_1, l_2, and c. We can assume that $l > c$. Otherwise, β could be derived without $A \rightarrow b[_3 B][_1 C]$.

a) If $l_1 \leqslant c$ and $l_2 \leqslant c$, then $|\beta_1| = l_1$, $|\beta_2| = l_2$, and $|\beta| = 1 + l_1 + l_2 = 1 + l$. We are given that $l > c$, and we deduce that $l \leqslant 2c$, since $l = l_1 + l_2$. It remains to show that $1 + l \leqslant (2c+2)(l-c) - 1$ when $c < l \leqslant 2c$. It is clear that $l - c$ is at least 1, so we must only verify that $1 + l \leqslant 2c + 2 - 1 = 2c + 1$, which is certainly true when $l \leqslant 2c$.

b) If $l_1 > c$ and $l_2 \leqslant c$, then $|\beta_1| \leqslant (2c+2)(l_1 - c)$, $|\beta_2| = l_2$, and

$$\begin{aligned}
|\beta| &= |\beta_1| + |\beta_2| + 1 \\
&\leqslant (2c+2)(l_1 - c) + l_2 + 1 \\
&\leqslant 2cl_1 - 2c^2 + 2l_1 - 2c + c + 1 \\
&\leqslant 2c\,(l-1) - 2c^2 + 2\,(l-1) - 2c + c + 1 \\
&\leqslant 2cl - 2c^2 + 2l - 2c - 2 - c + 1 \\
&\leqslant (2c+2)(l-c) - 1 - c \\
&\leqslant (2c+2)(l-c) - 1.
\end{aligned}$$

c) If $l_1 \leqslant c$ and $l_2 > c$, then $|\beta_1| = l_1$, $|\beta_2| \leqslant (2c+2)(l_2 - c) - 1$, and

$$|\beta| = |\beta_1| + |\beta_2| + 1$$

$$\leqslant l_1 + (2c+2)(l_2 - c)$$

$$\leqslant c + 2cl_2 - 2c^2 + 2l_2 - 2c$$

$$\leqslant c + 2c\,(l-1) - 2c^2 + 2\,(l-1) - 2c$$

$$\leqslant 2cl - 2c^2 + 2l - 2c - 2 - 2c + c$$

$$\leqslant (2c+2)(l-c) - 1 - c - 1$$

$$\leqslant (2c+2)(l-c) - 1.$$

d) If $l_1 > c$ and $l_2 > c$, then $|\beta_1| \leqslant (2c+2)(l_1 - c)$, $|\beta_2| \leqslant (2c+2)(l_2 - c) - 1$, and

$$|\beta| = |\beta_1| + |\beta_2| + 1$$

$$\leqslant (2c+2)(l_1 - c) + (2c+2)(l_2 - c) - 1 + 1$$

$$\leqslant 2cl_1 - 2c^2 + 2l_1 - 2c + 2cl_2 - 2c^2 + 2l_2 - 2c - 1 + 1$$

$$\leqslant 2c\,(l_1 + l_2) - 2c^2 + 2\,(l_1 + l_2) - 2c - 1 - 2c^2 - 2c + 1$$

$$\leqslant (2c+2)(l-c) - 1 - 2c^2 - 2c + 1$$

$$\leqslant (2c+2)(l-c) - 1.$$

Case 3. $\alpha = a\,[_2\,\alpha_1][_1\,\alpha_2]$ and $\beta = a\,[_2\,\beta_1][_1\,\beta_2]$.

β is derived $A \Rightarrow a\,[_2\,B\,][_1\,C\,] \Rightarrow^* a\,[_2\,\beta_1][_1\,\beta_2]$ with $B, C \in N_1 \bigcup N_2$. It must be true that $fr_2(\beta_1) = \alpha_1$ and $fr_2(\beta_2) = \alpha_2$. Let $l = 1 + l_1 + l_2$ where $l_1 = |\alpha_1|$ and $l_2 = |\alpha_2|$. The inductive hypothesis is applied to limit the sizes of β_1 and β_2. Again, there are four possible situations:

a) If $l_1 \leqslant c$ and $l_2 \leqslant c$, then $|\beta_1| = l_1$, $|\beta_2| = l_2$, and $|\beta| = 1 + l_1 + l_2 = l$. If $l > c$, it must be shown that $l \leqslant (2c+2)(l-c) - 1$. The logic is similar to case 2a.

b) If $l_1 > c$ and $l_2 \leqslant c$, then $|\beta_1| \leqslant (2c+2)(l_1 - c) - 1$, $|\beta_2| = l_2$, and

$|\beta| = |\beta_1| + |\beta_2| + 1$. This is shown to be smaller than $(2c+2)(l-c)-1$ by a method similar to that of case 2b.

c) If $l_1 \leqslant c$ and $l_2 > c$, then $|\beta_1| = l_1$, $|\beta_2| \leqslant (2c+2)(l_2-c)-1$, and $|\beta| = |\beta_1| + |\beta_2| + 1$. This is similar to case 2c.

d) If $l_1 > c$ and $l_2 > c$, then $|\beta_1| \leqslant (2c+2)(l_1-c)-1$, $|\beta_2| \leqslant (2c+2)(l_2-c)-1$, and $|\beta| = |\beta_1| + |\beta_2| + 1$. This is similar to case 2d.

Case 4. $\alpha = a[_2\,\alpha']$ and $\beta = a[_2\,\beta']$.

β is derived $A \Rightarrow a[_2\,B] \Rightarrow^* a[_2\,\beta']$ with $B \in N_1 \bigcup N_2$. It must be true that $fr_2(\beta') = \alpha'$, and $|\alpha'| = l-1$. This case is very similar to case 1.

Case 5. $\alpha = a[_2\,\alpha']$ and $\beta = b[_3\,\beta_1][_2\,\beta_2]$.

β is derived $A \Rightarrow b[_3\,B][_2\,C] \Rightarrow^* b[_3\,\beta_1][_2\,\beta_2]$ with $B \in N_2$ and $C \in N_1 \bigcup N_2$. $fr_2(\beta) = \alpha$, so it must be true that $fr_2(\beta_1) = \alpha_1$ and $fr_2(\beta_2) = \alpha_2$, and $\alpha = subs_2(\alpha_1, \alpha_2)$. Let $l_1 = |\alpha_1|$ and $l_2 = |\alpha_2|$. G is an increasing grammar, so $l > l_1$ and $l > l_2$. G has deletion constant c, so $l \geqslant l_1 + l_2 - c$. The inductive hypothesis can be applied to limit the sizes of β_1 and β_2. There are four possible situations:

a) If $l_1 \leqslant c$ and $l_2 \leqslant c$, then $|\beta_1| = l_1$, $|\beta_2| = l_2$, and $|\beta| = 1 + l_1 + l_2$. We also have $l > c$. Otherwise there would be no 3-2 step in the derivation of β. So $|\beta| = 1 + l_1 + l_2 \leqslant 2c+1$, and $(2c+2)(l-c)-1 \geqslant 2c+1$, since $l - c \geqslant 1$.

b) If $l_1 > c$ and $l_2 \leqslant c$, then $|\beta_1| \leqslant (2c+2)(l_1-c)-1$, $|\beta_2| = l_2$, and $|\beta| = |\beta_1| + |\beta_2| + 1$. This is shown to be smaller than

$(2c+2)(l-c)-1$ given that $l_2 \leqslant c$ and $l_1 \leqslant l-1$ by a method similar to that of case 2b.

c) If $l_1 \leqslant c$ and $l_2 > c$, then $|\beta_1| = l_1$, $|\beta_2| \leqslant (2c+2)(l_2-c)-1$, and $|\beta| = |\beta_1| + |\beta_2| + 1$. This is exactly parallel to case 2c.

d) If $l_1 > c$ and $l_2 > c$, then $|\beta_1| \leqslant (2c+2)(l_1-c)-1$, $|\beta_2| \leqslant (2c+2)(l_2-c)-1$, and

$$\begin{aligned} |\beta| &= |\beta_1| + |\beta_2| + 1 \\ &\leqslant (2c+2)(l_1-c) + (2c+2)(l_2-c) - 1 + 1 \\ &\leqslant 2cl_1 - 2c^2 + 2l_1 - 2c - 1 + 2cl_2 - 2c^2 + 2l_2 - 2c - 1 + 1 \\ &\leqslant 2c\ (l_1 + l_2 - c) - 2c^2 + 2(l_1 + l_2 - c) - 2c - 1 \\ &\leqslant 2cl - 2c^2 + 2l - 2c - 1 \\ &\leqslant (2c+2)(l-c) - 1. \end{aligned}$$

Case 6. $\alpha = a[_2\,\alpha']$ and $\beta = b[_3\,\beta']$.

β is derived $A \Rightarrow b[_3 B] \Rightarrow^* b[_3\,\beta']$ where $B \in N_2$. By cases 4 and 5, $|\beta'| \leqslant (2c+2)(l-c)-1$. So $|\beta| \leqslant (2c+2)(l-c)$. QED.

If a grammar is increasing and has a deletion constant, there is a linear relationship between the sizes of forests in the 3-d yield and the 2-d yield. Many useful 3-d forest grammars are not strictly increasing. So, given a non-increasing grammar G, we would like to know if there is an increasing grammar G' with the same yield as G. This topic is investigated in the next section.

3-d Grammars without Overlap Cycles

3-d forest grammars that do not have cycles of overlapping selectors can be transformed to grammars which are strictly increasing. To achieve this result, it will be necessary to define several terms similar to those used for analyzing 2-d grammars in chapter 4.

For many 3-d forest grammars, it can be determined whether the grammar is increasing by examination of its rules. The existence and value of a deletion constant can also be determined. In examining the rules, we must be able to predict the outcome of the 2-d frontier operation. This requires that we know which selectors will be applied in each frontier step.

Suppose β is a 3-d forest $\#[_3\, \delta][_2\, \gamma]$. There is a set of *outer selectors* in δ which will be applied when $fr_2(\beta)$ is taken. Outer selectors are analagous to the external selectors defined in chapter 4. They differ from external selectors in two respects. First, a 2-d forest has at most one external selector. A 3-d forest, however, has a finite set of outer selectors. Second, the outer selector set may contain selectors which will be truncated when the 2-d frontier operation is applied. For example, let $\beta = \#[_3 <2,21>] [_2\, \#[_2\, x\, [_1\, y\,]]]$ where $x, y \in \Xi_2$. Selectors x and y are the outer selectors of β, but only y is an external selector. x is eliminated when $fr_2(\beta)$ is taken. Thus, the outer selector set of a 3-d forest is a superset of the external selector set. Outer selectors are easily found. It is much more difficult to determine whether outer selectors will be external selectors.

DEFINITION 5-15. Let β be a forest in $H_3^1(\Sigma, \Xi)$. A selector $x \in \Xi_2$ is an *outer selector* of β when β contains x and the path from the root of β to x is ν where

1) $\nu \in \{1, 2\}^*$, or
2) if $\nu = \sigma 3 \pi$ for $\sigma, \pi \in \{1, 2, 3\}^*$,

   then $sel(<2, \sigma 2>, \beta)$ is undefined.

DEFINITION 5-16. Let β be a forest in $H_3^1(\Sigma, \Xi)$. The set *outsel*(β) is defined as $\{x \mid x$ is an outer selector of $\beta\}$.

With 2-d grammars, it was possible to construct a set of 1-d selectors, *exsel*(A), for each non-terminal A such that A derived β if and only if the external selector of β was in *exsel*(A). With 3-d grammars, a set of sets of outer selectors is associated with each non-terminal.

DEFINITION 5-17. Let G be a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$. The set *outsel*$(A)$ where $A \in N$ is defined as $\{s \mid s =$ *outsel*(β) where $A \Rightarrow^* \beta\}$.

Note that *outsel* is multiply defined. Hopefully, this will not cause confusion. If the argument of *outsel* is a 3-d forest, then *outsel* is a set of 2-d selectors. If the argument is a non-terminal in a 3-d grammar, then *outsel* is a set of sets of 2-d selectors.

LEMMA 5-18. Suppose G is a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$. If $A \in N$, then there is an effective procedure for constructing *outsel*(A).

PROOF. The algorithm given here constructs *outsel*(A) for every non-terminal in the grammar. The construction proceeds by taking a rule $A \rightarrow \beta$ from R, selecting the appropriate case depending on the form of β, and adding elements to *outsel*(A). The case statement is given as a procedure which is repeatedly invoked as rules are taken from R. Assume that A, B, and C are elements of N, $a \in \Sigma$, and $x \in \Xi_2$.

```
Procedure addel(A, β).
  Begin
    Case β of
      $a[_r B], r \geqslant 1$: add the elements of outsel(B) to outsel(A).
      $x[_1 B]$: add $\{x\} \bigcup s$ to outsel(A) for each s in outsel(B).
      $x$: add $\{x\}$ to outsel(A).
      $a[_r B][_1 C], r > 1$: add $s \bigcup t$ to outsel(A) for each s in
         outsel(B) and t in outsel(C).
      $a[_3 B][_2 C]$: add the elements of outsel(C) to outsel(A).
    End case.
  End addel.
```

The selection of rules from R is undertaken according to the following algorithm:

```
Repeat
  Take a rule $A \rightarrow \beta$ from R.
  If A is not in a cycle then
    Use the current method to form outsel(B) and outsel(C), where B
       and/or C are in β.
    Call addel(A, β).
    Remove $A \rightarrow \beta$ from R.
  Else if A is in a cycle $(A_1, A_2, \ldots, A_m)$ then
```

```
    Find every rule for $A_i$, $1 \leqslant i \leqslant m$.
    Use the current method to form *outsel* $(B)$ for each $B$ which is not
        in the cycle, but is in the right-hand side $\gamma$ of a rule
        $A_i \rightarrow \gamma$, $1 \leqslant i \leqslant m$.
    Repeat
        For $i = 1$ to $m$
            Call *addel* $(A_i, \gamma)$ for each rule of $A_i$.
    Until nothing new is added to any *outsel* set.
    Remove the rules $A_i \rightarrow \gamma$ from $R$, $1 \leqslant i \leqslant m$.
Until $R$ is empty.
```

The size of every *outsel* set is bounded by the number of possible subsets of Ξ_2, so the construction algorithm halts. It can be shown by induction on the size of a forest ζ that if $A \Rightarrow^* \zeta$, then *outsel* $(\zeta) \in$ *outsel* (A). It can also be shown by induction on the number of *addel* steps in the construction of *outsel* (A) that if *outsel* (A) contains selector set s, then $A \Rightarrow^* \zeta$ where $s =$ *outsel* (ζ). QED.

Definition 4-16 defined overlapping selectors for n-dimensional forests. The definition requires that the external selectors of a forest be known. If the external selectors are not known, the outside selectors can be used instead.

During the frontier operation on 3-d forests, entire subforests can be deleted. This is also true with 2-d forests, but it is relatively easy to remove such truncating steps from 2-d grammars. With 3-d grammars, however, the elimination of all truncating steps is more difficult. The following definition distinguishes between *complete* and *truncating* frontier steps.

DEFINITION 5-19. Let $\beta = \#[_3\,\gamma][_2\,\delta]$ be a forest in $H_3^1(\Sigma, \Xi)$. The 2-d frontier of β is *complete* if and only if $fr_2(\gamma)$ is complete, $fr_2(\delta)$ is complete, and for each subtree δ' of δ, the path from the root of δ to δ' is either

1) σ where $<2, \sigma> \in outsel(\gamma)$,
2) $\sigma 2\pi$ where $<2, \sigma> \in outsel(\gamma)$, or
3) σ where $<2, \sigma> \in outsel(\gamma)$, and $\delta' \in \Sigma$.

DEFINITION 5-20. Let G be a 3-d grammar $<\Sigma, \Xi, N, R, S>_3^k$. G is *complete* if for every $\alpha \in Y_2(G)$ there is a β such that $A \Rightarrow^* \beta$, $fr_2(\beta)$ is complete, $fr_2(\beta) = \alpha$, and $A \in N$. If G is not complete, then it is *truncating*.

LEMMA 5-21. Let G be a 3-d grammar $<\Sigma, \Xi, N, R, S>_3^k$. If G is complete, then G has deletion constant c, and c is no greater than the sum of the lengths of the paths on the selectors in Ξ_2.

PROOF. This lemma follows directly from the definition of a complete grammar. If a non-terminal derives a forest $\beta = \#[_3\,\gamma][_2\,\delta]$ and an entire subtree of δ is not selected for copying during the 2-d frontier operation, then $fr_2(\beta)$ is not complete, and neither is G. The only nodes that are deleted are the terminals along the selector paths, and so the number of deleted nodes cannot exceed the sum of the lengths of the selector paths for any single frontier step. QED.

During the frontier operation on a 2-d forest, only one selection/substitution process is active. This results from the fact that each subtree has only one external selector. Subtrees in 3-d forests, however, can have several external selectors. So several selection/substitution processes may be occurring simultaneously during the 2-d frontier operation. Some of the terminology used in the analysis of 2-d cycles will have to be broadened or generalized. Cycles, roots of cycles, essential non-terminals, and cycle steps will not need to be redefined, but it will be useful to speak of *full* cycle steps for 3-d grammars. A full cycle step is one expansion of a cycle from root to root.

DEFINITION 5-22. Let G be a grammar $<\Sigma,\Xi,N,R,S>_3^k$ which contains a cycle $\{A_1, A_2, \ldots, A_r\}$ with root A_1. A *full cycle step* is any structural form β derived from A_1 such that β contains A_1 and no non-terminals other that $A_2, \ldots, A_r$ are expanded in the derivation.

The leading and trailing non-terminals of a cycle were previously defined only for cyclic 2DNF grammars. The following definitions redefine these terms for 3-d cycles.

DEFINITION 5-23. Let G be a grammar $<\Sigma,\Xi,N,R,S>_3^k$ which contains a cycle whose root is A, and suppose β is a full cycle step derived from A which contains a non-terminal B. If the path from the root node of β to A is $\sigma 3\pi$ and the path to B is $\sigma 2\nu$, then B is a *leading* non-terminal of the cycle. If the path to A is $\sigma 2\pi$ and the path to B is $\sigma 3\nu$, then B is a *trailing* non-terminal of the cycle.

The concept of deletion cycles in 2-d grammars will be replaced with *overlap* cycles in 3-d grammars. The term "deletion" is abandoned because it is quite common in 3-d grammars for overlap cycles to show net growth in spite of the repeated deletion. This happens when an overlap cycle has several outside selectors, some overlapping and some non-overlapping.

DEFINITION 5-24. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_3^k$ which contains a cycle whose root is A. The cycle is an *overlap cycle* if and only if there is a full cycle step β derived from A such that

1) β has leading non-terminals $B_1, B_2, \ldots, B_l$,
2) $<2, \pi_i> \in outsel(B_i)$ for $1 \leqslant i \leqslant l$,
3) $B_i \Rightarrow^* \zeta_i$ and $sel(<2, \sigma_i>, fr_2(\zeta_i)) = <2, \pi_i>$ for $1 \leqslant i \leqslant l$, and
4) $\sigma_1 \cdot \sigma_2 \cdot \ldots \cdot \sigma_l$ is a proper prefix of $\pi_2 \cdot \pi_3 \cdot \ldots \cdot \pi_l \cdot \pi_1$.

A set of integers called *gain* can be associated with each full cycle step, and, depending on the values in the *gain* sets, cycles can be classified as increasing, decreasing, 0-gain, or mixed.

DEFINITION 5-25. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_3^k$ which contains a cycle whose root is A, and β is a full cycle step such that $A \Rightarrow^* \beta$. The integer g belongs to $gain(A)$ if and only if $g = |fr_2(\delta)| - |fr_2(\delta')|$ where $\delta \in H_3^1(\Sigma, \Xi)$, $\beta \Rightarrow^* \delta$, and δ' is the largest subforest of δ such that $A \Rightarrow^* \delta'$.

DEFINITION 5-26. Suppose G is a grammar $<\Sigma, \Xi, N, R, S>_3^1$ which contains a cycle whose root is A. The cycle is *increasing* if $g > 0$ for every $g \in gain(A)$, *decreasing* if $g < 0$ for every $g \in gain(A)$, *0-gain* if $gain(A) = \{0\}$, and *mixed* otherwise.

It will eventually be shown that any 3-d forest grammar which does not have overlapping selectors can be converted to an increasing grammar which has the same string yield. This will be accomplished by eliminating subforests which will be truncated during the frontier operation. Before truncation can be eliminated, however, a grammar must be partitioned according to the outside selector sets of the non-terminals.

LEMMA 5-27. Let G be a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ which has non-terminal A. There is an effective procedure for constructing a subgrammar $G_A : s$ such that A' in $G_A : s$ derives β if and only if A derives β and $outsel(\beta) = s$.

PROOF. Given a set of selectors, $s = \{<2,p_1>, <2,p_2>, \ldots, <2,p_k>\}$, $G_A : s = <\Sigma', \Xi', N', R', S'>_3^k$ can be constructed from the subgrammar G_A according to the steps below. Assume $A, B, C, D \in N$, $a, b \in \Sigma \bigcup \Xi_1$, and $x \in \Xi_2$.

1) Set $\Sigma' = \Sigma$, $\Xi' = \Xi$, $N' = N$, and $R' = R$.

2) Put A' in N' and $A' \rightarrow A_s$ in R'.

3) Add non-terminals B_t to N' for every non-terminal B of G_A and every t which is a subset of s.

4) Select a non-terminal B_t from N' for which there are no rules and make rules for it as follows:

a) If R has $B \rightarrow a\,[_m C\,]$ for $m \leqslant 3$, put $B_t \rightarrow a\,[_m C_t\,]$ in R'.

b) If R has $B \rightarrow x\,[_1 C\,]$ and $x \in t$, then add $B_t \rightarrow x\,[_1 C_t\,]$ and $B_t \rightarrow x\,[_1 C_u\,]$ to R, where $u = t - \{x\}$. If x is not in t, then do not add a rule to R'.

c) If R has $B \rightarrow a$ and $t = \phi$, put $B_t \rightarrow a$ in R'.

d) If R has $B \rightarrow x$ and $t = \{x\}$, put $B_t \rightarrow x$ in R'. If $t \neq \{x\}$, then add nothing to R'.

e) If R has $B \rightarrow a\,[_3 C\,][_2 D\,]$, put $B_t \rightarrow a\,[_3 C\,][_2 D_t\,]$ in R'.

f) If R has $B \rightarrow a\,[_m C\,][_1 D\,]$ for $m = 2$ or 3, put $B_t \rightarrow a\,[_m C_u\,][_1 D_v\,]$ in R' for every u and v such that $u \bigcup v = t$.

g) If none of cases a) through f) result in the addition of a rule for B_t, remove B_t from N'.

5) Repeat step 4 until all the non-terminals in N' have rules.

It is clear that the algorithm above halts. Each non-terminal B_t is processed by step 4 only once. The lemma can be proven by induction on the size of β. The induction is not shown here. QED.

In order to eliminate truncating frontier steps from a grammar, it is necessary to trace a 2-d selector path in a 3-d forest. The *trace* function below accomplishes this, provided the forest to be traced does not have overlapping selectors.

DEFINITION 5-28. The function $trace(x, \beta)$: $\Xi_2 \times H_3^1(\Sigma, \Xi) \rightarrow H_3^1(\Sigma, \Xi)$ is defined as follows:

$trace(<2, \lambda>, \beta) = \beta$,

$trace(x, \#[_3\, \gamma]) = trace(x, \gamma)$,

$trace(x, \#[_3\, \gamma][_2\, \delta]) = trace(x, \gamma)$,

$trace(<2, k\, \sigma>, \alpha[_m\, \gamma]) = trace(<2, k\, \sigma>, \alpha)$ for $m < k$, and

$trace(<2, k\, \sigma>, \alpha[_k\, \gamma]) = trace(<2, \sigma>, \gamma)$.

LEMMA 5-29. Suppose G is a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ with no overlapping selectors. There is a grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ such that $Y_2(G) = Y_2(G')$ and G' is complete.

PROOF. The grammar G' can be constructed by the steps below. Assume $A, B, C, D, E \in N$, and $a, \# \in \Sigma \bigcup \Xi$.

1) Set $G' = G$ and partition G' according to outside selector sets.

2) Find a rule $A \rightarrow a\, [_3\, B\,][_2\, C\,]$ in R'.

3) Let $s = outsel(B)$. Generate all the m distinct structural forms β_i which have

the following characteristics for $1 \leqslant i \leqslant m$:

a) $C \Rightarrow^* \beta_i$,

b) no non-terminal D in β_i is expanded unless $D = trace(<2, \sigma>, \beta_i)$, where σ is a prefix of the path on some $x \in s$, and

c) no rule in a 3-d cycle which does not produce a 2-node is applied more than once in deriving β_i.

4) For each β_i generated in step 3, if $\beta_i = a[_3 E][_1 \gamma]$, $a[_3 \gamma][_1 E]$, $a[_2 E][_1 \gamma]$, $a[_2 \gamma][_1 E]$, or $a[_1 E]$, where $trace(x, \beta_i) \neq E$ for any x in s, then delete E and the arc pointing to it in β_i.

5) For each β_i, add a rule $T_i \rightarrow \#[_3 B][_2 \beta_i]$ to R' where T_i is a new non-terminal. Also, replace $A \rightarrow a[_3 B][_2 C]$ in R' with i rules $A \rightarrow T_i$, $1 \leqslant i \leqslant m$. If $s = 0$ (and $m = 0$), replace $A \rightarrow a[_3 B][_2 C]$ with $A \rightarrow B$.

6) Repeat steps 2 through 5 for every rule $A \rightarrow a[_3 B][_2 C]$ in R'.

7) Remove unreachable non-terminals and rules from G'.

Since G has no overlapping selectors, it is possible to expand the non-terminal C in a rule $A \rightarrow a[_3 B][_3 C]$ until all the paths of the set of outside selectors of B are visible. Then it is evident what subforests will be truncated, and the rules can be modified so that the truncated forests are never derived. The algorithm above accomplishes this without making any other changes to the grammar, so any β in $Y_2(G)$ is also in $Y_2(G')$, and vice versa. This can be proven rigorously by induction on the size of β. The induction is not shown here. QED.

Any 3-d grammar without overlapping selectors can also be modified to make a grammar with the same string yield which is strictly increasing. One approach of this modification would be expanding the right-hand sides of grammar rules and taking partial frontiers to write new rules that combine the decreasing steps with larger increasing steps. A less complicated approach (but also more wasteful) is shown here which takes advantage of the fact that 2-nodes disappear during the 1-d frontier operation. Using this approach, extra 2-nodes are added to any rules which produce decreasing frontier steps. These additional nodes increase the size of the 2-d yield, but they are eliminated during the 1-d frontier operation.

LEMMA 5-30. Let G be a complete 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ without overlapping selectors. There is an increasing grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ such that $Y_1(G) = Y_1(G')$.

PROOF. The grammar G' can be constructed by the steps below. Assume $A, B, C \in N$, and $a \in \Sigma \bigcup \Xi$.

1) Set $G' = G$ and partition G' according to the outside selector sets.

2) Find a rule $A \rightarrow a[_3 B][_2 C]$ in R'.

3) Let $s = outsel(B)$. Generate all the m distinct structural forms β_i as in step 3 in the proof of lemma 5-29.

4) If $trace(x, \beta_i) = a$ is in $\Sigma \bigcup \Xi$ for every x in s, then replace one such a in β_i with $\#[_2 a]$. Do this for every $1 \leqslant i \leqslant m$.

5) Choose one of the outside selectors x in s and replace it in the rules for B in G_B with $\#[_2 \#[_2 \ldots \#[_2 x] \ldots]]$ to form G'_B, where the number of 2-nodes added is the sum of the lengths of the paths on every selector in s.

6) Replace $A \rightarrow a[_3 B][_2 C]$ with rules $A \rightarrow \#[_3 B'][_2 \beta_i]$, $1 \leqslant i \leqslant m$.

7) Repeat steps 2 through 6 until every $A \rightarrow a[_3 B][_2 C]$ rule has been modified.

It is apparent that the above algorithm halts. The largest sum of path-lengths in any outside selector set is a constant, and no selectors are modified by the algorithm. So step 5 does not have to be repeatedly applied for any single grammar rule. It is also apparent that the changes made will not affect the string yield of the grammar, since $fr_1(\alpha) = fr_1(\#[_2 \alpha])$ for any set α. The 2-d yield will contain larger forests, but the 1-d yield is the same. So $Y_1(G) = Y_1(G')$.

It remains to be verified that G' is increasing. Suppose G' generates $\beta = \#[_3 \gamma][_2 \delta]$. It is always the case that $|fr_2(\beta)| \geqslant |fr_2(\gamma)|$, and $|fr_2(\beta)| = |fr_2(\gamma)|$ only occurs when every subtree selected from $fr_2(\delta)$ during the frontier operation has size 1. Step 4 above ensures that at least one selected subtree has size 2. So $|fr_2(\beta)| > |fr_2(\gamma)|$.

It happens that $|fr_2(\beta)| \leqslant |fr_2(\delta)|$ when a subforest of δ is deleted during the frontier operation, or when the number of interior nodes in δ along the selection paths is larger than the number of non-selector nodes in γ, so that the frontier operation deletes more interior nodes than it adds. But G and G' are complete grammars, and step 5 increased the number of nodes in γ to exceed the sum of the path-lengths of the outside selectors. So no subforests are truncated, and the number of interior nodes added always exceeds the number deleted, giving $|fr_2(\beta)| > |fr_2(\delta)|$. QED.

If a 3-d forest grammar contains overlapping selectors, but no overlap cycles, then the selector paths can be changed so that they do not overlap one another. Suppose $\beta = \#[_3 \#[_2 x]][_2 \#[_3 \#[_2 y]][_2 y]]$, where x is a selector $<2, 221>$ and y is $<2, 121>$. The selector x overlaps y. As the path of x is traced, y is encountered after traversing one 2-arc. We can form β' by replacing x with $x' = <2, 2>$ and y with $y' = <2, 12121>$. The path of y' is formed by attaching the unused portion of the x path to the end of the y path. The selector x' does not overlap y' in the newly formed β', and $fr_2(\beta') = fr_2(\beta)$. This method will be used to prove that if a grammar has no overlap cycles, then there is a complete, increasing grammar with the same yield.

LEMMA 5-31. Let $\delta = \alpha_1[_{l_1}\alpha_2[_{l_2} \cdots \alpha_m [_{l_m} \beta] \ldots]]$ be a forest in $H_n^1(\Sigma, \Xi)$. If

$sel(<n, \pi>, sel(<n, \nu>, \delta)) = \beta$ where $\pi, \nu \in \{1, 2, \ldots, n\}^+$, then

$sel(<n, \nu\pi>, \delta) = \beta$.

PROOF. If $sel(<n, \pi>, sel(<n, \nu>, \delta)) = \beta$, then it must be true that $\nu = l_1 \cdot l_2 \cdot \ldots \cdot l_j$, $\pi = l_{j+1} \cdot \ldots \cdot l_m$, and $sel(<n, \nu>, \delta) = \alpha_{j+1}[_{l_{j+1}} \cdots$ $\alpha_m [_{l_m} \beta] \ldots]$ where $l_{j+1} = n$. But since it is also true that $sel(<n, l_1 \cdot \ldots \cdot l_m>, \delta) = \beta$, we have $sel(<n, \nu\pi>, \delta) = \beta$. QED.

THEOREM 5-32. Suppose G is a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ without overlap cycles. There is a complete, increasing grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ such that $Y_2(G) = Y_2(G')$.

PROOF. The grammar G' can be constructed by the steps below. Assume that $A, B, C \in N$, and $a \in \Sigma \bigcup \Xi$.

1) Set $G' = G$ and partition G' according to outside selector sets.

2) Find a rule $A \rightarrow a\,[_3\,B\,][_2\,C\,]$ in R'.

3) Let $s = outsel(B)$. Generate all the m distinct structural forms β_i as in step 3 in the proof of lemma 5-29.

4) Find β_i such that $trace(<2, \sigma>, \beta_i) = y$ where $y \in \Xi_2$ and σ is a proper prefix of a path on one selector x in s. Suppose $y = <2, \nu>$ and $x = <2, \sigma\pi>$. Then let $y' = <2, \nu\pi>$ and $x' = <2, \sigma>$. Replace y in β_i with y' to form β'_i, and replace the outside x in G_B with x' to form G_{B_i}. Add a rule $T_i \rightarrow \#[_3\,B_i\,][_2\,\beta'_i\,]$ where T_i is a new non-terminal.

5) Find β_i such that $trace(<2, \sigma>, \beta_i) = y$ where $j > 1$, $y \in \Xi_2$, and σ is a proper prefix on r selectors x_j in s, $1 \leqslant j < r$. Suppose $y = <2, \nu>$ and $x_j = <2, \sigma\pi_j>$. Then let $y'_j = <2, \nu\pi_j>$ and $x'_j = <2, \sigma 21^{j-1}>$. Replace y in β_i with $\#[_2\,y'_1[_1\,y'_2[_1 \ldots [_1\,y'_r\,] \ldots]]]$ to form β'_i. Replace the outside x_j in G_B with x'_j to form G_{B_i}. Add a rule $T_i \rightarrow \#[_3\,B_i\,][_2\,\beta'_i\,]$, where T_i is a new non-terminal.

6) Add a rule $T_i \rightarrow \#[_3\,B\,][_2\,\beta_i\,]$ for every β_i that was not processed by step 4 and step 5, and replace $A \rightarrow a\,[_3\,B\,][_2\,C\,]$ with rules $A \rightarrow T_i$, $1 \leqslant i \leqslant m$.

7) Repeat steps 2 through 6 until no overlapping selectors are left in G'.

The above algorithm is guaranteed to halt if there are no overlap cycles, and the resulting grammar G' has no overlapping selectors. Every β in $Y_2(G)$ is also in $Y_2(G')$, and vice versa. This can be proven rigorously by induction on the size of β. The induction is not shown here. The proof appeals to the previous lemma to establish that the selector compositions in steps 4 and 5 do not affect the yields of the derived forests. If G' is not complete, it can be converted to a complete grammar by the method of lemma 5-29. QED.

3-d Explicit Grammars

For an increasing 3-d forest grammar, the size of a derived 3-d forest is linearly related to the size of its 2-d yield. It must also be shown that a linear relationship exists between the size of the 2-d yield and the 1-d yield of a forest. To eliminate some of the complexity of the frontier operation, a class of grammars called *explicit* forest grammars will be defined.

The 1-d yield of a 3-d forest is the result of a two-pass frontier operation. The first pass does selection and substitution mandated by the selectors in Ξ_2. The second pass uses the selectors in Ξ_1. In explicit grammars, λ is the only path allowed for 1-d selectors, so that all the deletion occurs during the first pass of the frontier operation. Restrictions on the placement of 1-d selectors are also made to prevent substitution of trees whose string yield is λ during the 1-d frontier operation. The definitions that follow formally define explicit forests and grammars.

DEFINITION 5-33. Suppose $\beta \in H_2^1(\Sigma, \Xi)$. β is *explicit* if and only if β has no subforests $b[_2 z]$, $b[_2 z][_1 \gamma]$, $b[_2 \gamma][_1 z]$, or y, where $z = <1, \lambda>$, $\gamma \in H_2^1(\Sigma, \Xi)$, $b \in \Sigma \bigcup \Xi_2$, and $y \in \Sigma \bigcup \Xi_1 - \{z\}$.

DEFINITION 5-34. Let G be a 3-d forest grammar $<\Sigma, \Xi, N, R, S>_3^k$. G is *explicit* if and only if every forest in $Y_2(G)$ is explicit.

It is an open question whether explicit 3-d grammars retain all the power of non-explicit grammars. It will be shown, however, that explicit grammars are more powerful than grammars with only standard selectors. Explicit grammars have properties which make it easier to establish a linear relationship between the sizes of the 1-d and 2-d frontiers of a derived forest. These properties are described below, first as properties of 2-d forests, and then as properties of 3-d forests grammars.

The length of the string yield of a 2-d forest can be predicted by counting the 1-nodes in the forest. The *slen* function does this, and if the 1-d frontier of a 2-d forest β is defined, then $slen(\beta) = |fr_1(\beta)|$.

DEFINITION 5-66. The function $slen(\beta)$: $H_2^1(\Sigma, \Xi) \rightarrow N$ is defined as follows:

$slen(z) = 0$ for $z \in \Xi_1$,

$slen(x) = 1$ for $x \in \Sigma \bigcup \Xi_2$,

$slen(a[_1\gamma]) = 1 + slen(\gamma)$ for $a \in \Sigma \bigcup \Xi_2$,

$slen(a[_2\gamma]) = slen(\gamma)$ for $a \in \Sigma$, and

$slen(a[_2\gamma][_1\delta]) = slen(\gamma) + slen(\delta)$ for $a \in \Sigma$.

LEMMA 5-67. Suppose $\beta \in H_2^1(\Sigma, \Xi)$ and β is explicit. If $\beta \neq z$ where $z = <1, \lambda>$, then $slen(\beta) > 0$.

PROOF. The smallest possible explicit non-z forest has 2 nodes: $a[_1 z]$, $a[_1 x]$, $x[_1 z]$, $x[_1 x]$, $a[_2 x]$, or $x[_2 x]$ where $a \in \Sigma$ and $x \in \Xi_2$. The string length is 1 or 2 in each case.

Every larger forest contains at least one of these 2-node forests, and since non-λ paths on 1-d selectors are prohibited, no deletion occurs during the 1-d frontier operation. Therefore, every larger forest also has string length greater than zero. QED.

LEMMA 5-68. If $\beta = a\,[_2\,\gamma][_1\,\delta] \in H_2^1$ is explicit, then *slen* $(\beta) >$ *slen* (γ) and *slen* $(\beta) >$ *slen* (δ).

PROOF. Neither γ nor δ can be $<1, \lambda>$, since β is explicit. So by the previous lemma, *slen* $(\gamma) > 0$ and *slen* $(\delta) > 0$. *slen* (β) is just *slen* (γ) + *slen* (δ), so *slen* $(\beta) >$ *slen* (γ) and *slen* $(\beta) >$ *slen* (δ). QED.

As a result of lemma 5-67, $<1, \lambda>$ is the only explicit forest whose string frontier is λ. Lemma 5-68 guarantees that substitution steps in the 1-d frontier operation always result in larger strings. These results on 2-d forests can be duplicated for 3-d forests.

LEMMA 5-69. If $\beta \in H_3^1(\Sigma, \Xi)$ is explicit and $\beta \neq z$ where $z = <1, \lambda>$, then *slen* $(fr_2(\beta)) > 0$.

PROOF. Proof by induction on h, the number of 3-nodes and 3-2-nodes in β.

Base. $h = 0$.

In this case, $fr_2(\beta) = \beta$, and lemma 5-67 establishes the desired result.

Inductive hypothesis. Assume the lemma is true for $h < n$.

Inductive step. Show the lemma is true for $h = n$.

Case 1. $\beta = a\,[_3\,\beta_1]$ or $a\,[_3\,\beta_1][_1\,\beta_2]$.

It is not possible that $\beta_1 = z$, since z is not in $H_3^2(\Sigma, \Xi)$. So we can apply the inductive hypothesis to establish that *slen* $(fr_2(\beta_1)) > 0$. But $fr_2(\beta) = fr_2(\beta_1)$ or $fr_2(\beta_1)[_1\,fr_2(\beta_2)]$, so *slen* $(fr_2(\beta)) > 0$.

Case 2. $\beta = a[_3\,\beta_1][_2\,\beta_2]$.

Let $fr_2(\beta_1) = \alpha_1$, $fr_2(\beta_2) = \alpha_2$, and $fr_2(\beta) = subs_2(\alpha_1, \alpha_2)$. β_1 cannot be z. The inductive hypothesis establishes that $slen(\alpha_1) > 0$. If α_1 has no 2-d selectors, then $fr_2(\beta) = fr_2(\alpha_1)$, and the lemma is proven. Suppose α_1 has a 2-d selector x, and $\alpha_3 = sel(x, \alpha_2)$. α_3 cannot be z, since z is not in $H_2^2(\Sigma, \Xi)$. According to lemma 5-67 then, $slen(\alpha_3) > 0$. So $slen(fr_2(\beta)) = slen(subs_2(\alpha_1, \alpha_2)) \geqslant slen(\alpha_1) > 0$.

Case 3. $\beta = a[_k\,\beta_1]$ or $a[_2\,\beta_1][_1\,\beta_2]$ and β_1 or β_2 has a 3-d subforest.

Let the largest 3-d subforest of β_1 be β'_1, the largest 3-d subforest of β_2 be β'_2, $fr_2(\beta'_1) = \alpha'_1$, and $fr_2(\beta'_2) = \alpha'_2$. Cases 1 and 2 demonstrate that $slen(\alpha'_1) > 0$ and $slen(\alpha'_2) > 0$. But α'_1 (and α'_2) is a subtree of $fr_2(\beta)$, so $slen(fr_2(\beta)) > 0$. QED.

LEMMA 5-70. If $\beta \epsilon H_3^1(\Sigma, \Xi)$, β is explicit, and $fr_2(\beta) = \alpha = a[_2\,\gamma][_1\,\delta]$, then $slen(\alpha) > slen(\gamma)$ and $slen(\alpha) > slen(\delta)$.

PROOF.

Case 1. $\beta = a[_2\,\beta_1][_1\,\beta_2]$.

We have $fr_2(\beta) = \alpha$, $fr_2(\beta_1) = \gamma$, and $fr_2(\beta_2) = \delta$. Since β is explicit, neither β_1 nor β_2 is z. By lemma 5-69, $slen(\gamma) > 0$ and $slen(\delta) > 0$. But $slen(\alpha) = slen(\gamma) + slen(\delta)$, so $slen(\alpha) > slen(\gamma)$ and $slen(\alpha) > slen(\delta)$.

Case 2. $\beta = a[_3\,\beta_1][_1\,\beta_2]$.

In this case, $fr_2(\beta) = \alpha$, $fr_2(\beta_1) = a[_2\,\gamma]$, and $fr_2(\beta_2) = \delta$. Since neither β_1 nor β_2 can be z, $slen(\delta) > 0$ and $slen(a[_2\,\gamma]) > 0$ (by lemma 5-69). It follows that $slen(\gamma) > 0$. But $slen(\alpha) = slen(\gamma) + slen(\delta)$, so $slen(\alpha) > slen(\gamma)$ and $slen(\alpha) > slen(\delta)$.

Case 3. $\beta = a[_3\, \beta_1][_2\, \beta_2]$.

This is impossible: $fr_2(\beta)$ must be a tree.

Case 4. $\beta = a[_3\, \beta_1]$.

Now we have $fr_2(\beta) = fr_2(\beta_1) = \alpha$. The lemma is established by analysis of β_1, to which case 1 or 2 will eventually be applied. QED.

The previous two lemmas can be applied to forests which are produced by explicit 3-d forest grammars. This will help establish a linear relationship between the 2-d and 1-d yields. It is still possible, however, for an arbitrarily large 2-d forest to yield a short string. Consider, for example, the explicit tree $\beta = \#[_2\, \#[_2\, \#[_2\, \ldots \#[_2\, a\, [_1\, z\,]] \ldots]]]$. The 1-d frontier of β is just $a\,[_1\, z\,]$, regardless of how many 2-nodes β has. It is certainly possible to write a grammar with a cycle with arbitrarily increases the number of consecutive 2-nodes in a forest. But it is also true that if there were such a cycle, it need never be applied more than a fixed number of times. Arbitrarily repeated cycle steps add nothing to the 1-d yield of the grammar. If we are interested in the smallest β which yields a particular string, then we can establish a constant upper bound on the number of consecutive 2-nodes in β by analyzing the rules of the grammar. This is formalized in the next lemma.

LEMMA 5-71. Suppose G is an explicit 3DNF grammar $< \Sigma, \Xi, N, R, S >_3^k$ such that $A \in N$, $A \Rightarrow^* \beta$, $fr_2(\beta) = \gamma$, and β is the smallest forest derived from A

such that $fr_1(\beta) = w$. Then γ has no more than 2^c consecutive 2-nodes on any path from its root to a leaf, where c is the number of non-terminals in N.

PROOF. Let γ have the form $a_1[_2 a_2[_2 \ldots a_m[_2 \gamma'] \ldots]]]$, with m consecutive 2-nodes. The forest β which yields γ is a forest with γ' as its rightmost subforest and 3-, 3-2-, or 2-nodes everywhere else. If m is greater than c, then some non-terminal has been applied more than once in generating the 3-2 portion of β, and there is a smaller forest with the same yield. The largest possible subforest without such repetition has 2^c nodes or less, and the largest possible number of consecutive 2-nodes in the 2-d yield of a forest of 2^c nodes is 2^c. So γ has no more than 2^c consecutive 2-nodes. QED.

If a 2-d forest is explicit and there is a constant bound on the number of consecutive 2-nodes, then the size of a forest and its 1-d frontier are linearly related. The next lemma formalizes this result, and the theorem that follows extends it to sets of 2-d forests which are the yields of explicit 3-d forest grammars.

LEMMA 5-72. Let β be a forest over $H_2^1(\Sigma, \Xi)$ such that β is explicit and there are no more than m consecutive 2-nodes on a path from its root to a leaf. If γ is a subforest of β, $\gamma \neq z$, and $slen(\gamma) = l$, then $|\gamma| \leqslant m(2l-1) + 3l - 1$.

PROOF. The lemma is proved by induction on l.

Base. $l = 1$.

The largest possible γ is $a_1[_2 a_2 \ldots a_m[_2 b[_1 z]] \ldots]]$ and $fr_1(\gamma) = b[_1 z]$. If there were any 2-1-nodes, then l would be at least 2 (by lemma 5-68). So

$l = 1$ and $|\gamma| = m+2 \leqslant m+3-1$.

Inductive hypothesis. Assume the lemma is true for $l < n$.

Inductive step. Show the lemma is true for $l = n$.

The most expensive way to add a 1-node, b, to an existing forest γ' is to insert m consecutive 2-nodes at every opportunity. If γ' is the largest possible forest whose frontier has $n-1$ nodes, then γ is no larger than $a_1[_2 a_2[_2 \ldots$ $a_m[_2 \#[_2 a_1[_2 a_2[_2 \ldots a_m[_2 b[_1 z]] \ldots]]][_1 \gamma']] \ldots]]$. So

$$
\begin{aligned}
|\gamma| &\leqslant m+1+m+2+m(2(l-1)-1)+3(l-1)-1 \\
&\leqslant 2m+3+2ml-3m+3l-3-1 \\
&\leqslant 2ml-m+3l-1 \\
&\leqslant m(2l-1)+3l-1.
\end{aligned}
$$

QED.

THEOREM 5-73. Suppose G is an explicit 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ such that $A \in N$, $A \Rightarrow^* \beta$, $fr_2(\beta) = \gamma$, and β is the smallest forest derived from A such that $fr_1(\beta) = w$. If l is the length of w and c is the number of nonterminals in N, then $|\gamma| \leqslant 2^c(2l-3)+3l-4$.

PROOF. Consider $Y_2(G)'$, the subset of $Y_2(G)$ which contains only the smallest forests which yield a string in $Y_1(G)$. Each of these forests is explicit and has a constant bound of 2^c on the number of consecutive 2-nodes (by lemma 5-71). If γ in $Y_2(G)'$ yields w, then $slen(\gamma) = |w|-1$. The value $slen(\gamma)$ corresponds to $|fr_1(\gamma)|$ when the 1-d frontier is defined, provided we subtract 1 from the size of w. We need to subtract 1 because the *slen* function does not count the $<1, \lambda>$ at the end of the 1-d frontier. For each γ in $Y_2(G)'$, we can apply lemma 5-72 to get

$$\begin{aligned}|\gamma| &\leqslant 2^c(2(l-1)-1)+3(l-1)-1\\ &\leqslant 2^c(2l-2-1)+3l-3-1\\ &\leqslant 2^c(2l-3)+3l-4.\end{aligned}$$

QED.

It was shown in lemma 5-29 that every 3-d forest grammar with standard selectors has a yield-equivalent complete grammar. It can now be shown that every standard 3-d grammar has a yield-equivalent explicit grammar. A grammar which is complete allows no truncation during the 2-d frontier operation, but truncation may still occur during the 1-d frontier. This happens if a forest has a leaf which is a terminal, not a selector. If terminal leaves are eliminated, then no truncation will occur. The following two lemmas prove for 2 and 3 dimensions that a forest which contains a terminal leaf has no external 1-d selector.

LEMMA 5-74. If $\beta \in H_2^1(\Sigma, \Xi)$ and β has a leaf which is a terminal, then β has no external selector.

PROOF. This can be proven by induction on $|\beta| = l$.

Base. $l = 1$.

β is just a terminal a, so β has no external selector.

Inductive hypothesis. Assume the lemma is true for $l = n-1$.

Inductive Step. Show the lemma is true for $l = n$.

If $\beta = a[_k\, \beta_1]$ for $k = 1$ or 2, then β_1 has a terminal leaf. The inductive hypothesis establishes that β_1 has no external selector, so β has no external selector.

If $\beta = a\,[_2\,\beta_1][_1\,\beta_2]$, then either β_1 or β_2 has a terminal leaf. If β_1 has a terminal leaf, then, by the inductive hypothesis, it has no external selector and $fr_1(\beta) = fr_1(\beta_1)$. So β has no external selector. If β_1 has no terminal leaves and β_2 has a terminal leaf, then the inductive hypothesis establishes that β_2 has no external selector. $fr_1(\beta)$ is formed by attaching a suffix of $fr_1(\beta_2)$ to the end of $fr_1(\beta_1)$, so β has no external selector. QED.

LEMMA 5-75. If $\beta \in H_3^1(\Sigma, \Xi)$, $fr_2(\beta)$ is complete, and β has a leaf which is a terminal, then $fr_2(\beta)$ has no external selector.

PROOF. This can be rigorously proven by induction on the size of β. The most interesting case is $\beta = a\,[_3\,\beta_1][_2\,\beta_2]$. If β_1 has a terminal leaf, then so does $fr_2(\beta_1)$ and $fr_2(\beta)$. The fact that $fr_2(\beta)$ is complete assures us that the terminal leaf will not be truncated during the 2-d frontier operation. But if $fr_2(\beta)$ has a terminal leaf, then , by lemma 5-74, it has no external selector. The logic is similar if β_2 has a terminal leaf. QED.

If a subforest has no external 1-d selector, then any subforest connected to its root by a 1-arc will be truncated during the 1-d frontier operation. A 3-d grammar can be modified to remove the subforests that will be truncated from the grammar rules. Once these subforests have been removed, selectors $<1, \lambda>$ can be attached to the terminal leaves, and the modified grammar will have the same yield as the original grammar.

LEMMA 5-76. Let G be a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ such that Ξ contains only standard selectors. There is a 3DNF grammar G' such that G' is complete, G' has no productions $A \rightarrow a$ where $A \in N$ and $a \in \Sigma$, and $Y_1(G) = Y_1(G')$.

PROOF. G' can be constructed according to the steps below. Assume A, B, C, D, E, F, $I \in N$, a, b, $c \in \Sigma$, and $x \in \Xi$.

1) Set $G = G'$.

2) Partition G' according to outside selectors, make G' complete, and restore it to 3DNF.

3) Partition G' again to isolate subforests with terminal leaves. This process replaces each non-terminal A in G' with A_Σ and A_X.

a) Replace $A \rightarrow x$ with $A_X \rightarrow x$.

b) Replace $A \rightarrow a$ with $A_\Sigma \rightarrow a$.

c) Replace $a[_l B]$, $l \leqslant 3$, with $A_\Sigma \rightarrow a[_l B_\Sigma]$ and $A_X \rightarrow a[_l B_X]$.

d) Replace $a[_l B][_r C]$, $1 < l \leqslant 3$, $1 \leqslant r < 3$, with $A_\Sigma \rightarrow a[_l B_\Sigma][_r C_\Sigma]$, $A_\Sigma \rightarrow a[_l B_\Sigma][_r C_X]$, $A_\Sigma \rightarrow a[_l B_X][_r C_\Sigma]$, and $A_X \rightarrow a[_l B_X][_r C_X]$.

4) Find a rule $A_\Sigma \rightarrow a[_l B][_r C_\Sigma]$ in R'.

5) Let $s = outsel(B)$, and form all the m distinct structural forms β_i derived from C_Σ by the method of step 3 in the proof of lemma 5-29. Form rules $T_i \rightarrow a[_3 B][_2 \beta_i]$ and $A_\Sigma \rightarrow T_i$ for each $1 \leqslant i \leqslant m$, and remove $A_\Sigma \rightarrow a[_3 B][_2 C_\Sigma]$.

6) If there is a rule $A_\Sigma \rightarrow a[_l B_\Sigma][_1 C]$, then replace it with $A_\Sigma \rightarrow a[_l B_\Sigma]$.

7) If there is a rule $T_i \rightarrow a[_3 B][_2 \beta_i]$ and G_B contains a rule $D \rightarrow y[_1 F]$ or $D \rightarrow$

$b[_l E][_1 F]$ where the forest derived from E contains y, $y \in s$, and $sel(y, \beta_i) = c$ or $c[_2 l_\Sigma]$, replace the rule for D with $D \to y$ or $D \to b[_l E]$ to form G_{B_i}. Then replace $T_i \to a[_3 B][_2 \beta_i]$ with $T_i \to a[_3 B_i][_2 \beta_i]$.

8) Remove useless rules from G', return it to 3DNF, and repeat steps 2 through 6 until no further changes can be made.

9) Replace every rule $A \to a$ in R' with $A \to a[_1 z]$, where $z = <1, \lambda>$.

Each pass through the algorithm has the net effect of eliminating some forests with 1-arcs pointing to them, so the algorithm certainly halts. The goal of the algorithm is to eliminate rules which produce subforests that will be truncated during the 1-d frontier operation. Then the selector $<1, \lambda>$ is attached to any remaining terminal leaf, so the resulting grammar is complete and produces forests which have only selectors as leaves.

Step 2 is a partitioning step which does not affect the yield of the grammar. A non-terminal A_Σ derives forests which have at least one terminal as a leaf. A_X derives forests which have only selectors as leaves. Steps 4 and 5 also rewrite some rules without affecting the yield of the grammar.

Steps 6 and 7 eliminate subforests which will be truncated during the 1-d frontier operation. Lemma 5-75 can be cited to prove that the 1-d yields will not be affected.

The steps are repeated until no further changes occur. At this point, the grammar has been effectively pruned so that no subforests are produced which will be truncated during the 2-d or 1-d frontier operations. Step 9 attaches the selector

$<1, \lambda>$ to any terminal leaves which remain, so the requirements of the lemma are satisfied. QED.

Once terminal leaves have been eliminated from a grammar, it can be further modified so that it will not produce subforests of the form $a\,[_2\, z\,]$, $a\,[_2\, z\,][_1\, \beta]$, or $a\,[_2\, \beta][_1\, z\,]$, where $z = <1, \lambda>$. When these modifications have been made, the resulting grammar is explicit.

THEOREM 5-77. Suppose G is a 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ which has only standard selectors. There is a complete, explicit grammar $G' = <\Sigma', \Xi', N', R', S'>_3^k$ with standard selectors such that $Y_1(G) = Y_1(G')$.

PROOF. Construct G' according to the steps below. Assume $A, B, C, D, E, F \in N'$, $a \in \Sigma'$, $x \in \Xi'_2$, and $z \in \Xi'_1$.

1) Set $G' = G$, make G' complete, remove terminal leaves, and partition G' according to outside selectors.

2) Partition G' again to form A_Σ, A_X, and A_Z for every A in N'. Rules are invented as follows:

   a) Replace $A \rightarrow z$ with $A_Z \rightarrow z$.

   b) Replace $A \rightarrow x$ with $A_X \rightarrow x$.

   c) Replace $A \rightarrow a\,[_1\, B\,]$ with $A_\Sigma \rightarrow a\,[_1\, B_\Sigma]$, $A_\Sigma \rightarrow [_1\, B_X\,]$, and $A_\Sigma \rightarrow a\,[_1\, B_Z\,]$.

   d) Replace $A \rightarrow x\,[_1\, B\,]$ with $A_X \rightarrow x\,[_1\, B_X\,]$, $A_x \rightarrow [_1\, B_Z\,]$, and $A_\Sigma \rightarrow x\,[_1\, B_\Sigma]$.

   e) Replace $A \rightarrow a\,[_r\, B\,]$ where $r > 1$ with $A_\Sigma \rightarrow a\,[_r\, B_\Sigma]$, $A_X \rightarrow a\,[_r\, B_X\,]$, and $A_Z \rightarrow a\,[_r\, B_Z\,]$.

   f) Replace $A \rightarrow a\,[_r\, B\,][_1\, C\,]$ where $r > 1$ with $A_\Sigma \rightarrow a\,[_r\, B_\Sigma][_1\, C_\Sigma]$, $A_\Sigma \rightarrow$

$a[_r B_\Sigma][_1 C_X]$, $A_\Sigma \rightarrow a[_r B_\Sigma][_1 C_Z]$, $A_\Sigma \rightarrow a[_r B_X][_1 C_\Sigma]$, $A_\Sigma \rightarrow a[_r B_X][_1 C_\Sigma]$, $A_X \rightarrow a[_r B_X][_1 C_X]$, $A_X \rightarrow a[_r B_X][_1 C_Z]$, and $A_Z \rightarrow a[_r B_Z][_1 C_Z]$.

g) Replace $A \rightarrow a[_3 B][_2 C]$ with $A_\Sigma \rightarrow a[_3 B_\Sigma][_2 C_\Sigma]$, $A_\Sigma \rightarrow a[_3 B_\Sigma][_2 C_X]$, $A_\Sigma \rightarrow a[_3 B_\Sigma][_2 C_Z]$, $A_\Sigma \rightarrow a[_3 B_X][_2 C_\Sigma]$, $A_X \rightarrow a[_3 B_X][_2 C_X]$, and $A_Z \rightarrow a[_3 B_X][_2 C_Z]$.

3) Find a rule $A \rightarrow a[_3 B][_2 C]$ (where A, B, and C can have any subscript) in R'.

4) Let $s = outsel(B)$, and form all the m distinct structural forms β_i derived from C as in step 3 of the proof of lemma 5-29. Form rules $T_i \rightarrow a[_3 B][_2 \beta_i]$ and $A \rightarrow T_i$ for each $1 \leqslant i \leqslant m$, and remove $A \rightarrow a[_3 B][_2 C]$.

5) If $trace(x, \beta_i) = b[_2 D_Z]$ for an x in s such that G_B has a rule $E \rightarrow x$, remove D_Z from β_i and replace the x in G_B which selects D_Z with z to form G_{B_i}. If G_B has a rule $E \rightarrow x[_1 F]$ for the x which selects D_Z, replace it with $E \rightarrow F$. Then, replace $T_i \rightarrow a[_3 B][_2 \beta_i]$ with $T_i \rightarrow a[_3 B_i][_2 \beta_i]$.

6) Repeat steps 3, 4, and 5 for each rule $A \rightarrow a[_3 B][_2 C]$.

7) If R' has a rule $A_Z \rightarrow \beta$, replace it with $A_Z \rightarrow z$.

8) If R' has a rule $A \rightarrow a[_r B][_1 C_Z]$, replace it with $A \rightarrow a[_r B]$.

9) If R' has a rule $A \rightarrow a[_r B_Z][_1 C]$, replace it with $A \rightarrow C$.

The purpose of the algorithm is to isolate and eliminate rules which produce only trees whose 1-d frontiers are z. Step 2 partitions the grammar so that a non-terminal A_Σ derives forests which contain at least one terminal 1-node. A_X

derives forests which have only selectors as 1-nodes and leaves, and A_Z derives forests which have only z as leaves and no 1-nodes. The 1-d frontier of any forests derived from A_Z is z.

Steps 3, 4, and 5 expand the right-hand sides of rules for 3-2 non-terminals so that 2-d selectors which select z-forests can be eliminated. The modifications made in step 5 do not affect the 1-d yield, since $fr_1(a[_2 z][_1 \beta]) = fr_1(\beta)$. The replacement of $x[_1 E]$ with E shortens the path to E. This would cause a problem if some selector of a containing forest had a path to E, but this is impossible with standard selectors. Standard 2-d selectors have paths in 1^* and cannot penetrate a tree which results from a 2-d frontier step.

Step 7 replaces any z-forest with a single z. Steps 8 and 9 eliminate z-forests without affecting the 1-d yield. The resulting grammar is complete and it produces no subforests $a[_2 z]$, $a[_2 z][_1 \beta]$, or $a[_2 \beta][_1 z]$. QED.

A Subclass of 1Y3F within Context-Sensitive

It has been demonstrated that for many 3-d forest grammars, there is a linear size relationship between forests in the 3-d yield and the 2-d yield. It has also been shown that many grammars have a linear size relationship between the 2-d yield and the 1-d yield. We now define a subclass of 1Y3F for languages which exhibit both relationships.

DEFINITION 5-78. $1Y3F_l$ is a subset of 1Y3F. A language L belongs to $1Y3F_l$ when $L = Y_1(G)$ for some 3-d forest grammar G, and if $w \in Y_1(G)$, then there is β in $Y_3(G)$ such that $fr_1(\beta) = w$ and $|\beta| \leqslant f(|w|)$ for some linear function f.

THEOREM 5-79. If G is a 3-d forest grammar which is increasing, explicit, and has a deletion constant, then $Y_1(G) \in 1Y3F_l$.

PROOF. G has a deletion constant and is increasing, so by lemma 5-14, if $\alpha \in Y_2(G)$, then there is a β in $Y_3(G)$ such that $|\beta| \leqslant f(|\alpha|)$ for a linear function f, and $fr_2(\beta) = \alpha$. G is explicit, so by lemma 5-73, if $w \in Y_1(G)$, then there is an α in $Y_2(G)$ such that $fr_1(\alpha) = w$ and $|\alpha| \leqslant g(|w|)$ for a linear function g. So if $w \in Y_1(G)$, then there is β in $Y_3(G)$ such that $fr_1(\beta) = w$ and $|\beta| \leqslant h(|w|)$ where h is the linear function which is the composition of f and g. QED.

Any explicit 3-d forest grammar with strictly increasing cycles yields a language in $1Y3F_l$. The cycles in the grammar can even be overlap cycles as long as they show a net gain. Certain 0-gain overlap cycles can also be included in a grammar which yields a $1Y3F_l$ language. If a 0-gain cycle is consistently deleting in one of its selection processes, then the number of possible cycle steps is limited by the size of the pre-cycle forest. If too many cycle steps are applied, a path error occurs when the frontier is taken.

DEFINITION 5-80. Let G be a 3-d forest grammar $<\Sigma, \Xi, N, R, S>_3^k$ which contains a cycle whose root is A, $A \in N$. If $A \Rightarrow^* \beta$ and no other cycle non-terminals are used to derive β, then β is an *input forest* of the cycle.

DEFINITION 5-81. Let G be a 3DNF grammar with a deletion constant which contains a cycle. The cycle is *linear* if it is a 0-gain overlap cycle such that the maximum number of cycle steps which can be applied without introducing a path error for an input forest β is $f(|fr_2(\beta)|)$, where f is a linear function.

THEOREM 5-82. If G is an explicit 3DNF grammar $<\Sigma, \Xi, N, R, S>_3^k$ which has a deletion constant and every cycle in G is increasing or linear, then $Y_1(G) \epsilon$ 1Y3F$_l$.

PROOF. Suppose G has a linear cycle whose root is A, B is outside the cycle, and B derives the input forest β for the cycle, $A, B \epsilon N$. Suppose also that if $\alpha \epsilon Y_2(G_B)$, then $Y_3(G_B)$ has β such that $fr_2(\beta) = \alpha$ and $|\beta| \leqslant g(|\alpha|)$. G is explicit, so if $w \epsilon Y_1(G)$, then $Y_2(G)$ has δ such that $fr_1(\delta) = \alpha$ and $|\delta| \leqslant f(|w|)$. The A-cycle is linear, so there is a linear function h such that if $A \Rightarrow^* \gamma$ and β is the input tree which is a subforest of γ, then the maximum number of cycle steps is $h(|fr_2(\beta)|)$. Finally, assume γ is the smallest forest derivable from A such that $fr_1(fr_2(\gamma)) = fr_1(\delta) = w$. Since G has a deletion constant and the A-cycle is 0-gain, we know that $|\delta| = |\alpha|$ and each step of the cycle adds no more than a constant number of nodes to the derived 3-d forest. If this constant is c and the number of steps is s, then

$$\begin{aligned}|\gamma| &\leqslant |\beta| + s \times c\\ &\leqslant g(|\alpha|) + h(|\alpha|) \times c\\ &\leqslant g(|\delta|) + h(|\delta|) \times c\\ &\leqslant g(f(|w|)) + h(f(|w|)) \times c.\end{aligned}$$

Thus, we have established that $|\gamma| \leqslant g'(|w|)$ where g' is a linear composition of

f, g, and h. Such a composite function can be constructed for each linear cycle in G. Since the number of cycles in G is finite, we can establish a composite function for the whole grammar. We do not have to worry about an infinite composition of functions if there is a cycle of cycles. A cycle of linear cycles can be analyzed as a single linear cycle. A cycle of increasing cycles can be analyzed as a single increasing cycle. A mixed cycle of linear and increasing cycles is neither increasing nor linear, so it is ruled out by the requirements of the theorem. QED.

Linear overlap cycles are significant because the example grammars at the beginning of this chapter employ them to perform list-processing operations. The grammar MERGE performs the task of merging two lists into one. The cycle generated by non-terminal M does the merging. Each step in this cycle deletes two interior nodes and adds two interior nodes without truncating any subforests. So the cycle has a net gain of 0. Each step of the cycle removes an element from one of two input lists. So the largest possible number of cycle steps is no greater than the sum of the elements in the two lists, and this sum is smaller than the number of nodes in the 2-d input forest.

The grammar SQUASH can be subjected to a similar analysis. It performs the task of converting a complex list of elements to a simple list. The grammar has consecutive overlap cycles generated by non-terminals R and W. These cycles operate on a complex list of elements $\#[_2\, a\, [_1\, z\,]]$, which is initially a full binary tree. The W-cycle repeatedly splits the leftmost subtree into two subtrees and attaches them to the front of the list. Each steps adds two interior nodes and deletes two interior nodes without truncating and subtrees. The W-cycle terminates when the first element of the old list is moved to become the first element of the new list. Then, the R-cycle continues the processing of the old list by either splitting the first element of the old list or by moving an element from the old list to the new list. Each step of the R-cycle adds the same number of interior nodes as it deletes, and no step truncates any subforests. The total number of element-splitting steps over the two cycles is no larger than the number of interior nodes in the original binary tree. If a full binary tree has n leaves, then it has $n-1$ interior nodes. So the number of splitting steps is no more than $n-1$. The number of element-moving steps is no more than the number of leaves in the original tree. The number of cycle steps for R and W combined is less than $2n$, where n is the number of leaves in the original tree, and the number of leaves is half the total number of nodes in the tree. So the number of cycle steps is linearly bounded in terms of the size of the 2-d frontier of the input tree.

The grammar BA2N yields the subset of $\{a \mid b\}^+$ such that each string has 2^n a's, $n > 0$. It can now be demonstrated that the language yielded by BA2N is in $1Y3F_l$.

LEMMA 5-83. $Y_1(BA2N) \in 1Y3F_l$.

PROOF. The grammar BA2N is a composite of the grammars SQUASH, MERGE, BLIST, and A2N. Inspection of the grammar rules verifies that BA2N is explicit. No rule generates any of the forbidden subforests. The overlap cycles in BA2N come from SQUASH and MERGE. By the analysis in the paragraphs preceding this lemma, these cycles are linear and complete. All the other cycles in the grammar are complete, non-overlapping, and increasing. It follows from theorem 5-82 that $Y_1(\text{BA2N}) \in \text{1Y3F}_l$. QED.

THEOREM 5-84. ALG_3^1 is a proper subset of 1Y3F_l.

PROOF. If L belongs to ALG_3^1, then there is a 3-d forest grammar G such that $Y_1(G) = L$ and G has only standard selectors. There is a grammar G' such that $Y_1(G') = Y_1(G)$, G' is complete and explicit (theorem 5-77), and G' is increasing (lemma 5-30). By theorem 5-79, then, $L \in \text{1Y3F}_l$. Lemma 5-83 shows that $Y_1(\text{BA2N})$ is in 1Y3F_l, but $Y_1(\text{BA2N})$ is not in ALG_3^1 (see theorem 5-1). So ALG_3^1 is a proper subset of 1Y3F_l. QED.

Baldwin (1983) has shown that a linear-bounded grammar can be written which simulates the frontier operation on a forest in $H_n^k(\Sigma, \Xi)$ with standard selectors. This grammar can be used for forests with extended selectors if the *search* routine on page 241 of Baldwin's dissertation is modified to accommodate them. Below is a *search* subgrammar written specifically for 3-d forests. It demonstrates that the required modification in Baldwin's grammar will not violate the linear space bound. The non-terminal set for this subgrammar is $\{<sp_1>, <sp_2>, \ldots, <sp_m>, <rc>, <lb>, <llb>, <move>\}$. The non-terminals $<sp_i>$ are the search non-terminals. There is one for every suffix p of a path on a 2-d selector of the input forest. $<rc>$ is a non-terminal which carries a search non-terminal to the right over a 2-d tree. $<lb>$ and $<llb>$ are symbols representing left brackets that have been passed over during a carry operation. The initial configuration for the search is $<sp>\beta$ where β is the 2-d forest to be traversed. The final configuration is $\beta_1 <move> \beta_2 \beta_3$, where β_2 is the tree selected for copying. The steps which precede and follow these configurations are unchanged from Baldwin's dissertation. Let the symbol a represent any element of Σ. The rules for *search* are given below.

1) $<sp>a \rightarrow a <sp>$ if $p \neq \lambda$,

2) $<sp>a \rightarrow <move>a$ if $p = \lambda$,

3) $<sp>[_2 \rightarrow <sp><rc>[_2$ if $p = 1p'$,

4) $<sp>[_2 \rightarrow [_2<sp'>$ if $p = 2p'$,

5) $<sp>[_1 \rightarrow [_1<sp'>$ if $p = 1p'$,

6) $<sp><rc>[_2 \rightarrow [_2<llb><sp><rc>$,

7) $<llb><sp><rc>a \rightarrow a <llb><sp><rc>$,

8) $<llb><sp><rc>[_j \rightarrow [_j <llb><lb><sp><rc>$,

9) $<llb><sp><rc>] \rightarrow] <sp>$,

10) $<lb><sp><rc>a \rightarrow a <lb><sp><rc>$,

11) $<lb><sp><rc>[_j \rightarrow [_j <lb><lb><sp><rc>$,

12) $<lb><sp><rc>] \rightarrow] <sp><rc>$,

13) $<llb>a \rightarrow a <llb>$,

14) $<llb>] \rightarrow]<llb>$,

15) $<llb>[_j \rightarrow [_j <llb>$,

16) $<lb>a \rightarrow a <lb>$,

17) $<lb>] \rightarrow]<lb>$, and

18) $<lb>[_j \rightarrow [_j <lb>$.

The first 5 rules consume the path p, switching from $<sp>$ to $<sp'>$ when appropriate. Rules 6 through 12 carry an $<sp>$ to the right over a subtree. Each time a left bracket is passed over, an $<lb>$ (or $<llb>$ for the first left bracket) is added. Whenever a right bracket is passed over, an $<lb>$ is erased. When a right bracket is encountered and only an $<llb>$ marker is present, the carry operation is complete. Rules 13 through 18 are auxiliary to the carry operation. They just move the left bracket markers to the right. It is clear from the examination of the rules above that the number of symbols required to implement the *search* operation on β is $2 + |\beta| + b$, where b is the number of left brackets in β. So the space required for the operation is linear in the size of the input tree.

THEOREM 5-85. If $L = Y_1(G)$ and $L \in$ 1Y3F$_l$ then there is a linear bounded automaton M which accepts L.

PROOF. Construct M by incorporating in its finite control the 3-d forest grammar G and the linear bounded grammar which simulates the frontier operation. We know there is a function f such that if $w \in L$, then $G \Rightarrow^* \beta$, $fr_1(\beta) = w$, and $|\beta| \leqslant f(|w|)$. We also know that the frontier simulator requires space not to exceed $g(|w| + |\beta|)$, where g is also a linear function. This space bound is expressed in terms of $|w| + |\beta|$ because the 1-d frontier of β may be smaller or larger that β. If $fr_1(\beta)$ is smaller than β, then the simulator operates in $g(|\beta|)$ space. If the frontier of β is larger, then the simulator operates in $g(|w|)$ space. In either case, $g(|w| + |\beta|)$ is a safe upper bound. M operates according to the steps below:

1) Start with w on the tape.

2) Mark a working area on the tape of size $g(|w|+|\beta|)+f(|w|+|\beta|)$ next to w.

3) Generate the next largest β derivable from the start symbol of G (or the smallest β if this is the first time this step is executed). Write β in the working area, replacing what was there before.

4) If β will not fit in the working area, then reject w and stop.

5) Simulate the frontier operation on β, replacing it with $w' = fr_1(\beta)$.

6) If the working area is not large enough for the simulation of the frontier, then go to step 4.

7) Compare w and w'. If they are the same, then accept w and stop. If they are not the same, then go to step 3.

M is guaranteed to halt, accepting or rejecting w. The amount of tape required is on the order of $|w|+g(|w|+f(|w|))$. QED.

COROLLARY 5-86. If G is a 3-d forest grammar such that $Y_1(G) \in 1Y3F_l$, then there is a context-sensitive grammar G' such that $L(G') = Y_1(G)$.

PROOF. This follows immediately from the previous theorem and the fact that a context-sensitive grammar exists for any language which is accepted by a linear bounded automaton. QED.

This chapter has defined a subclass of 1Y3F whose languages are recognizable in linear space. 3-d forest grammars must be restricted to achieve this result, but some interesting grammars which simulate list processing can be written within the restrictions. Grammars with decreasing cycles have not been considered here. It is an open question whether 1Y3F $\equiv$ 1Y3F$_l$.

CHAPTER 6.
OUTSIDE-IN LANGUAGES IN 0Y3F

The 3-d forest grammar BA2N of the previous chapter is an example of a $03YF_l$ language which yields an outside-in macro language. This chapter demonstrates that $0Y3F_l$ contains some simple OI macro languages as well as all the IO macro languages.

DEFINITION 6-1. A *simple* macro grammar is an IO or OI macro grammar $<\Sigma, F, V, \rho, S, P>$ such that each cycle has only one non-terminal and every rule has one of the following forms for $A, B \in F$, $\alpha_i \in (F \bigcup \Sigma)^+$, $x_i \in V$, and $\sigma_i \in F \bigcup \Sigma \bigcup V$, $1 \leqslant i \leqslant m$:

1) $A \rightarrow B(\alpha_1, \alpha_2, \ldots \alpha_m)$,
2) $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$,
3) $A(x_1, x_2, \ldots, x_m) \rightarrow \sigma$, or
4) $A \rightarrow \alpha$.

The definitions of IO and OI macro grammars are defined by Fischer (1968). A simple macro grammar does not have any non-terminals which are arguments of other non-terminals. Baldwin (1983) has demonstrated that ALG_3^1 corresponds to the class of IO macro languages, and it immediately follows that simple IO macro $\subset$ IO macro $\subset 0Y3F_l$.

The IO Conversion Method

IO macro grammars can be converted directly to 3-d forest grammars. The direct conversion method replaces a macro rule $A(x, y, z) \rightarrow B(ax, by, cz)$ with a forest rule $A \rightarrow \#[_3 B][_2 \#[_2 a [_1 x]][_1 \#[_2 b [_1 y]][_1 \#[_2 c [_1 z]]]]]$, for selectors

$x = \langle 2, \lambda \rangle$, $y = \langle 2, 1 \rangle$, and $z = \langle 2, 11 \rangle$. Consider a macro grammar M whose rules are the following:

1) $S \rightarrow A(a)$,

2) $A(x) \rightarrow A(xx)$, and

3) $A(x) \rightarrow xx$.

The language $L(M)$ is the set $\{a^{2^n}\}$, $n \geqslant 1$. We can convert this directly to a 3-d forest grammar, A2N', whose rules are shown below. Let $x = \langle 2, \lambda \rangle$ and $z = \langle 1, \lambda \rangle$.

1) $S \rightarrow \#[_3 A\,][_2 \#[_2 a\,[_1 z\,]]]$,

2) $A \rightarrow \#[_3 A\,][_2 \#[_2 x\,[_1 x\,]]]$, and

3) $A \rightarrow \#[_2 x\,[_1 x\,]]$.

Figure 12 contains a derivation of the grammar A2N' and its frontiers. Contrast these forests with those of figure 5. The grammars A2N and A2N' have the same 1-d yield, but they produce different shaped forests. During the 2-d frontier operation on the forest of A2N', multiple copies of 2-d selectors are made before the a's replace the selectors. With the corresponding forest of A2N, no 2-d selectors are copied. Rather, the a's are copied with each step of the frontier operation. These examples illustrate that if a simple 3-d grammar generates its yield by copying 2-d selectors, there is a simple 3-d grammar with the same yield which does not involve the copying of 2-d selectors.

Another method for converting simple macro grammars to 3-d forest grammars is presented here which can be generalized to accommodate OI as well as IO

```
#---3---------------2
    |               |
    #---3-------2   #---2
        |       |       |
        #---2   #---2   a---1
            |       |       |
            y---1   y---1   z
                |       |
                y       y
```

derived 3-d forest

```
#---2
    |
    #---2---------------1
        |               |
        #---2-------1   #---2
            |       |       |
            a---1   #---2   #---2-------1
                |       |       |       |
                z       a---1   a---1   #---2
                            |       |       |
                            z       z       a---1
                                                |
                                                z
```

2-d frontier

```
a---1
    |
    a---1
        |
        a---1
            |
            a---1
                |
                z
```

1-d frontier

FIGURE 12. 3-d forest and yields produced by A2N'.

grammars. This method will be called the IO construction method. The resulting grammar will produce forests which look like those of A2N instead of A2N'. It will be useful to have a function which turns a string into a 1-d forest and converts macro variables to 2-d selectors. This function is call *imap*.

DEFINITION 6-2. Suppose M is a macro grammar with terminal set Σ and variable set V. The function $imap(\sigma)$: $\Sigma \bigcup V^{+} \rightarrow H_1^1(\Sigma, \Xi)$ is defined as

$imap(a\ \sigma) = a[_1\ imap(\sigma)]$ and

$imap(a) = a \quad$ for $a \in \Sigma$,

$imap(x_j\ \sigma') = <2, 21^{j-1}>[_1\ imap(\sigma')]$ and

$imap(x_j) = <2, 21^{j-1}>$ for $x_j \in V$.

Let M be a simple IO macro grammar. A 3-d forest grammar G = $<\Sigma, \Xi, N, R, S>_3^k$ can be constructed such that $Y_0(G) = L(M)$ by the steps below:

1) Set Σ equal to the terminal set of M, and add # to Σ.

2) Set N equal to the non-terminal set of M, and make S the same as the start symbol of M.

3) Set $\Xi_2 = \{<2,2>, <2,21>, \ldots, <2, 21^{m-1}>\}$, where m is the cardinality of the variable set of M. Also, set $\Xi_1 = \{<1, \lambda>\}$ and $\Xi_3 = \phi$.

4) Add rules to R as follows:

   a) If M has a rule $A \rightarrow \sigma$, $\sigma \in (\Sigma \bigcup N)^{+}$, and no non-terminal in σ has any arguments, then put a corresponding rule $A \rightarrow \#[_2\ \sigma']$ in R, where σ' is the 1-d forest such that $\sigma' = imap(\sigma)$.

   b) If M has $S \rightarrow A(\alpha_1, \alpha_2, \ldots \alpha_m)$, where the α's are strings of terminals and non-terminals, put $A \rightarrow\ \#[_2\ \#[_2\ \alpha'_1][_1\ \#[_2\ \alpha'_2][_1\ \ldots\ \#[_2\ \alpha'_m]\ldots]]]$ in R where $\alpha'_i = imap(\alpha_i)$ for each $1 \leqslant i \leqslant m$.

c) If M has $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$, where the σ's are string of terminals, non-terminals, and variables, put $B \rightarrow$ $\#[_3 \#[_2 \#[_2 \sigma'_1][_1 \#[_2 \sigma'_2][_1 \ldots \#[_2 \sigma'_m] \ldots]]]][_1 A]$ in R, where $\sigma'_i =$ $imap(\sigma_i)$, $1 \leqslant i \leqslant m$.

d) If M has $A(x_1, x_2, \ldots, x_m) \rightarrow \sigma$ where σ is a string of terminals, non-terminals, and variables, then add the rule $S \rightarrow \#[_3 \#[_2 \sigma']][_2 A]$, where $\sigma' = imap(\sigma)$.

The construction method above is the inversion of Baldwin's direct method. The inversion of a 3-d grammar relies on a 3-d generalization of lemma 4-48 to establish that different shaped 3-d forests can have the same yield. Specifically, $fr_1(\#[_3 \#[_3 \alpha][_2 \beta]][_2 \gamma]) = fr_1(\#[_3 \alpha][_2 \#[_3 \beta][_2 \gamma]])$, where α, β, and γ are in $H_3^1(\Sigma, \Xi)$ and have no overlapping selectors. The lemmas and culminating theorem that follow prove that the IO construction method produces a 3-d forest grammar which is yield-equivalent to the original macro grammar. The first lemma establishes the correspondence of the macro substitution operation and the 2-d frontier operation.

LEMMA 6-3. Suppose 3-d forest grammar G is constructed from a simple IO macro grammar M by the IO construction method, and M has a derivation $S \Rightarrow^* A(\gamma_1, \gamma_2, \ldots, \gamma_m) \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$, where $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$ is the last type 2 rule applied (see definition 6-1). Suppose also that 3-d forest grammar G has a derivation $B \Rightarrow^* \zeta$ and $\zeta =$ $\#[_3 \#[_2 \#[_2 \pi_1][_1 \#[_2 \pi_2][_1 \ldots \#[_2 \pi_m] \ldots]]]][_2 \gamma']$, where $fr_2(\gamma') =$

$\#[_2 \#[_2 \gamma'_1][_1 \#[_2 \gamma'_2][_1 \ldots \#[_2 \gamma'_m] \ldots]]]$, $fr_2(\zeta) = \#[_2 \#[_2 \beta'_1][_1 \#[_2 \beta'_2][_1 \ldots$ $\#[_2 \beta'_m] \ldots]]]$, $\sigma'_i = imap(\sigma_i)$, $\sigma'_i \Rightarrow^* \pi_i$, and $fr_1(\gamma'_i) = imap(\gamma_i)$, $1 \leqslant i \leqslant m$. Then $\beta_i \Rightarrow^* w_i$ if and only if $fr_1(\beta'_i) = w'_i$, where w'_i is a string of terminals and $w'_i = imap(w_i)$.

PROOF. β_i is the string of terminals and non-terminals which results when $(\gamma_1, \gamma_2, \ldots, \gamma_m)$ is substituted into σ_i. Let σ_i have the form $\rho_1 \nu_1 \rho_2 \nu_2 \ldots$ $\rho_l \nu_l \rho_{l+1}$, where the ρ's are strings of terminals and non-terminals and the ν's are variables. When the substitution is performed, each ν_j, $1 \leqslant j \leqslant l$, selects γ_h where ν_j refers to the h th macro argument. So $\beta_i = \rho_1 \gamma_{\nu_1} \rho_2 \gamma_{\nu_2} \ldots \rho_l \gamma_{\nu_l} \rho_{l+1}$. If the ρ's contain non-terminals, they are context-free non-terminals which have no arguments. They can be expanded to obtain a terminal string $\beta_i \Rightarrow^* u_1 \gamma_{\nu_1} u_2 \gamma_{\nu_2} \ldots$ $u_l \gamma_{\nu_l} u_{l+1}$.

The frontier operation on ζ gives $subs_2(fr_2(\#[_2 \#[_2 \pi_1][_1 \#[_2 \pi_2][_1 \ldots \#[_2 \pi_m]$ $\ldots]]])$, $fr_2(\gamma')) = \#[_2 \#[_2 \beta'_1][_1 \#[_2 \beta'_2][_1 \ldots \#[_2 \beta'_m] \ldots]]]$, where β_i is $subs_2(\pi_i, fr_2(\gamma'))$, $\sigma'_i = imap(\sigma_i)$, and $\sigma'_i \Rightarrow^* \pi_i$. σ'_i is a bracketed version of σ_i with the same ρ's, and the ν's converted to selectors. Let σ'_i be represented as $\rho'_1[_1 x_1[_1 \ldots \rho'_l[_1 x_l[_1 \rho'_{l+1}]] \ldots]]]]$. Each selector x_j is $<2, 21^{h-1}>$, where ν_j in σ_i refers to the h th macro argument. σ'_i is expanded before the 2-d frontier operation to give $\pi_i = u'_1[_1 x_1[_1 \ldots u'_l[_1 x_l[_1 u'_{l+1}]] \ldots]]]]$. The rules in G used to derive u'_j from ρ'_j correspond exactly to rules in M which derive u_j from ρ_j. So $fr_2(u'_j)$ is a bracketed version of u_j. When $fr_2(\gamma')$ is substituted into π_i, each x_j is replaced with $\#[_2 \gamma_h]$, where $x_j = <2, 21^{h-1}>$. The frontier substitution, therefore, has the same effect as the macro substitution. The result is $\beta'_i =$ $u'_1[_1\, \gamma'_{x_1}[_1 \ldots u'_l\,[_1\, \gamma'_{x_l}[_1\, u'_{l+1}]] \ldots]]]]$, and $fr_1(\beta'_i)$ is w'_i, a bracketed version of w_i. QED.

LEMMA 6-4. Suppose 3-d forest grammar G is constructed from a simple IO macro grammar M by the IO construction method. M has a derivation $S \Rightarrow^* A(\gamma_1, \gamma_2, \ldots, \gamma_m) \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$, where $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$ is the last type 2 rule applied, and $\beta_i \Rightarrow^* w_i$ $1 \leqslant i \leqslant m$, if and only if G has a derivation $B \Rightarrow^* \zeta$, $\zeta = \#[_3\, \#[_2\, \#[_2\, \pi_1][_1\, \#[_2\, \pi_2][_1 \ldots \#[_2\, \pi_m] \ldots]]]][_2\, \delta]$, $fr_2(\zeta) = \#[_2\, \#[_2\, \beta'_1]\,[_1\, \#[_2\, \beta'_2][_1 \ldots \#[_2\, \beta'_m] \ldots]]]$, and $fr_1(\beta'_i) = w'_i$, where $\sigma'_i = \mathit{imap}(\sigma_i)$, and $w'_i = \mathit{imap}(w_i)$, $1 \leqslant i < m$.

PROOF. The lemma is established by proving the propositions (A) and (B) below.

(A) If $S \Rightarrow^* A(\gamma_1, \gamma_2, \ldots, \gamma_m) \Rightarrow B(\beta_1, \beta_2, \ldots, \beta_m)$ and $\beta_i \Rightarrow^* w_i$, $1 \leqslant i \leqslant m$, then $B \Rightarrow^* \zeta$, $fr_2(\zeta) = \#[_2\, \#[_2\, \beta'_1][_1\, \#[_2\, \beta'_2][_1 \ldots \#[_2\, \beta'_m] \ldots]]]$, and $fr_1(\beta'_i) = w'_i$.

Proof of (A) by induction on n, the number of type 2 macro rules applied to derive $B(\beta'_1, \beta'_2, \ldots, \beta'_m)$.

Base. $n = 1$.

The macro derivation is $S \Rightarrow A(\alpha_1, \alpha_2, \ldots, \alpha_m) \Rightarrow^* A(\gamma_1, \gamma_2, \ldots, \gamma_m) \Rightarrow B(\beta_1, \beta_2, \ldots, \beta_m)$, using rules $S \rightarrow A(\alpha_1, \alpha_2, \ldots, \alpha_m)$ and $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$. By steps 4b and 4c of the IO construction method, G has rules $B \rightarrow \#[_3\, \#[_2\, \#[_2\, \sigma'_1][_1\, \#[_2\, \sigma'_2][_1 \ldots \#[_2\, \sigma'_m] \ldots]]]][_2\, A]$ and $A \rightarrow \#[_2\, \#[_2\, \alpha'_1][_1\, \#[_2\, \alpha'_2][_1 \ldots \#[_2\, \alpha'_m] \ldots]]]$. If some σ'_i

contains non-terminals, then σ'_i can be expanded to π_i using rules supplied by step 4a of the construction. Similarly, if some α'_i has non-terminals, it can be expanded to γ'_i by rules corresponding to those in M which expand α_i to γ_i. So $B \Rightarrow^* \zeta$ where $\zeta = \#[_3 \#[_2 \#[_2 \pi_1][_1 \#[_2 \pi_2][_1 \ldots \#[_2 \pi_m] \ldots]]]]$ $[_2 \#[_2 \gamma'_1][_1 \#[_2 \gamma'_2][_1 \ldots \#[_2 \gamma'_m] \ldots]]]$, and $fr_1(\gamma'_i) = imap(\gamma_i)$. Let $fr_2(\zeta)$ be $\#[_2 \#[_2 \beta'_1][_1 \#[_2 \beta'_2][_1 \ldots \#[_2 \beta'_m] \ldots]]]$. We are given that $\beta_i \Rightarrow^* w_i$, so we can apply lemma 6-3 to get $fr_1(\beta'_i) = w'_i$, $1 \leqslant i \leqslant m$.

Inductive hypothesis. Assume (A) is true for $n < r$.

Inductive step. Show (A) is true for $n = r$.

The macro derivation is $S \Rightarrow^* A(\alpha_1, \alpha_2, \ldots, \alpha_m) \Rightarrow^* A(\gamma_1, \gamma_2, \ldots, \gamma_m)$ $\Rightarrow B(\beta_1, \beta_2, \ldots, \beta_m)$, and the last type 2 rule applied is $A(x_1, x_2, \ldots, x_m) \rightarrow$ $B(\sigma_1, \sigma_2, \ldots, \sigma_m)$. By step 4c of the IO method, G has a rule $B \rightarrow$ $\#[_3 \#[_2 \#[_2 \sigma'_1][_1 \#[_2 \sigma'_2][_1 \ldots \#[_2 \sigma'_m] \ldots]]]][_2 A]$. By the inductive hypothesis, $A \Rightarrow^* \delta$, $fr_2(\delta) = \#[_2 \#[_2 \alpha'_1][_1 \#[_2 \alpha'_2][_1 \ldots \#[_2 \alpha'_m] \ldots]]]$, and $fr_1(\alpha'_i) = imap(\gamma'_i)$. If any σ'_i has non-terminals, it can be expanded to π_i using rules supplied by step 4a of the IO method. So $B \Rightarrow^* \zeta$ where $\zeta =$ $\#[_3 \#[_2 \#[_2 \pi_1][_1 \#[_2 \pi_2][_1 \ldots \#[_2 \pi_m] \ldots]]]][_2 \delta]$. Let $fr\,2(\zeta) =$ $\#[_2 \#[_2 \beta'_1][_1 \#[_2 \beta'_2][_1 \ldots \#[_2 \beta'_m] \ldots]]]$, and apply lemma 6-3 to establish that $fr_1(\beta'_i) = imap(w_i)$, $1 \leqslant i \leqslant m$.

(B) If $B \Rightarrow^* \zeta$, $fr_2(\zeta) = \#[_2 \#[_2 \beta'_1][_1 \#[_2 \beta'_2][_1 \ldots \#[_2 \beta'_m] \ldots]]]$, and $fr_1(\beta'_i) =$ w'_i, then $S \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$ and $\beta_i \Rightarrow^* w_i$, $1 \leqslant i \leqslant m$.

The proof of (B) is similar to the proof of (A). QED.

THEOREM 6-5. Suppose a 3-d forest grammar G is constructed from a simple IO macro grammar M by the IO construction method. M has a derivation $S \Rightarrow^* \alpha$ where α is a string of terminals if and only if G has a derivation $S \Rightarrow^* \beta$ and $fr_1(\beta) = imap(\alpha)$.

PROOF.

First, assume that M has $S \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m) \Rightarrow^* \alpha$, the last macro rule applied is $B(x_1, x_2, \ldots, x_m) \rightarrow \sigma$, and $\beta_i \Rightarrow^* w_i$, $1 \leqslant i \leqslant m$. By lemma 6-4, G has $B \Rightarrow^* \zeta$, $fr_2(\zeta) = \#[_2 \#[_2 \beta'_1][_1 \#[_2 \beta'_2][_1 \ldots \#[_2 \beta'_m] \ldots]]]$, and $fr_1(\beta'_i) = imap(w_i)$. By construction step 4d of the IO method, G has a rule $S \rightarrow \#[_3 \#[_2 \sigma']][_2 B]$. So $S \Rightarrow^* \beta$ where $\beta = \#[_3 \#[_2 \sigma']][_2 \zeta]$. By an argument similar to that in the proof of lemma 6-3, it can be shown that $fr_1(\beta) = \alpha'$ where $\alpha' = imap(\alpha)$.

Now assume G has $S \Rightarrow^* \#[_3 \#[_2 \sigma']][_2 B] \Rightarrow^* \beta$ and $fr_1(\beta) = imap(\alpha)$. Then M must have a rule $B(x_1, x_2, \ldots, x_m) \rightarrow \sigma$. Suppose B can be expanded in G to give $\#[_3 \#[_2 \#[_2 \sigma'_1][_1 \#[_2 \sigma'_2][_1 \ldots \#[_2 \sigma'_m] \ldots]]]][_2 A] \Rightarrow^* \zeta$, where $fr_2(\zeta) = \#[_2 \#[_2 \gamma'_1][_1 \#[_2 \gamma'_2][_1 \ldots \#[_2 \gamma'_m] \ldots]]]$, and $fr_1(\gamma'_i) = imap(w_i)$, $1 \leqslant i \leqslant m$. By lemma 6-4, m has $S \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$ and $\beta \Rightarrow^* w_i$. As discussed in the proof of lemma 6-3, the substitution of $(\beta_1, \beta_2, \ldots, \beta_m)$ into σ has the same effect as the substitution of $fr_2(\zeta)$ into σ'. So $imap(\alpha) = \alpha' = fr_1(\beta)$. QED.

The OI Conversion Method

A generalization of the IO conversion method can be used to construct 3-d forest grammars which simulate simple OI macro grammars. OI processing allows non-terminals to be copied before they are expanded. This is simulated by generating a list of possible expansions of each argument. If two copies of some argument are required, then two distinct elements are taken from the list of possible expansions. This method will be called the OI construction method.

As before, a function will be employed to add 1-brackets to strings and convert macro variables to 2-d selectors. This function is similar to *imap* , except that the mapping of variables to selectors is more complicated. The symbol x_{i_j} will represent the j th occurrence of the i th argument in the argument list of a macro non-terminal.

DEFINITION 6-6. Suppose M is a macro grammar with terminal set Σ and variable set V. The function $omap(\sigma)$: $(\Sigma \bigcup V)^+ \rightarrow H_1^1(\Sigma, \Xi)$ is defined as

$omap(a\ \sigma) = a\,[_1\, omap(\sigma)]$ and

$omap(a) = a \quad$ for $a \in \Sigma$,

$omap(x_{i_j}\ \sigma') = <2, 21^{i-1}22(12)^{j-1}> [_1\, omap(\sigma')]$, and

$omap(x_{i_j}) = <2, 21^{i-1}22(12)^{j-1}>$ for $x_{i_j} \in V$.

Let M be a simple OI macro grammar. A 3-d forest grammar $G = \ <\Sigma, \Xi, N, R, S>_3^k$ can be constructed such that $Y_0(G) = L(M)$ by the steps below:

1) Set Σ equal to the terminal set of M and add # to Σ.

2) Add A, A_1, and A_2 to N for each non-terminal A of M.

3) Add selectors to Ξ and rules to R as follows:

a) If M has a rule $A \rightarrow \sigma$, $\sigma \in (\Sigma \bigcup N)^+$, and no non-terminal in σ has any arguments, then put a corresponding rule $A \rightarrow \#[_2\, \sigma']$ in R, where $\sigma' =$ *omap*(σ).

b) If M has $S \rightarrow A(\alpha_1, \alpha_2, \ldots \alpha_m)$, where the α's are strings of terminals and non-terminals, put $A \rightarrow \#[_2\, \#[_2\, \#[_1\, T_1]]\, [_1\, \#[_2\, \#[_1\, T_2]]\, [_1\, \ldots \#[_2\, \#[_1\, T_m\,]]$ $\ldots]]]$ in R, where each T_i is a newly invented non-terminal. Add rules $T_i \rightarrow \#[_2\, \#[_2\, \alpha'_i\,][_1\, T_i\,]]$ and $T_i \rightarrow \#[_2\, \#[_2\, \alpha'_i\,]]$, for each $1 \leqslant i \leqslant m$, where $\alpha'_i =$ *omap*(α_i).

c) If M has $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$, where the σ's are string of terminals, non-terminals, and variables, put $B \rightarrow B_1$, $B_1 \rightarrow$ $\#[_3\, \#[_2\, \gamma]][_2\, B_1]$, $B_1 \rightarrow \#[_3\, \#[_2\, \gamma]][_2\, B_2]$, and $B_2 \rightarrow \#[_3\, \#[_2\, \delta]][_2\, A\,]$ in R. γ is the subforest $\#[_2\, o_1[_1\, \#[_2\, \#[_2\, \sigma'_1][_1\, n_1]]]]\, [_1\, \ldots \#[_2\, o_m\, [_1\, \#[_2\, \#[_2\, \sigma'_m\,]$ $[_1\, n_m\,]]]] \ldots]$, where $o_i = <2, 21^{i-1}2(21)^{p_i}>$, $n_i = <2, 21^{i-1}21>$, p_i is the number of occurrences of x_i over $(\sigma_1, \sigma_2, \ldots, \sigma_m)$, and $\sigma'_i =$ *omap*(σ_i), $1 \leqslant i \leqslant m$. δ is the subforest $\#[_2\, n_1[_1\, \#[_2\, z\,]]]$ $[_1\, \#[_2\, n_2[_1\, \#[_2\, z\,]]]\, [_1\, \ldots \#[_2\, n_m\, [_1\, \#[_2\, z\,]]] \ldots]]$.

d) If M has $A(x_1, x_2, \ldots, x_m) \rightarrow \sigma$ where σ contains no non-terminals with arguments, then add the rule $S \rightarrow \#[_3\, \#[_2\, \sigma']][_2\, A\,]$ to R, where $\sigma' =$ *omap*(σ).

This construction method is an extension of the method for IO grammars. Instead of maintaining a list of single arguments for a non-terminal, a list of lists of arguments is generated. When copying occurs. multiple distinct versions of an argument are selected. If a non-terminal derives a forest which will become an argument of another non-terminal, then an arbitrarily long list of distinct possible derivations is generated.

Step 3b initializes the argument list for a non-terminal A which has arguments $\alpha_1, \alpha_2, \ldots, \alpha_m$. New non-terminals T_i are added to generate a list for each argument which contains an arbitrary number of versions of the argument.

Step 3c produces rules to simulate an OI derivation step which will involve the copying of arguments. The new forests, which are created from the old arguments, are themselves arguments of another non-terminal, so multiple versions of each new forest are generated. List processing techniques are applied to create new argument lists from the old ones. Repeated occurrences of some x_i in the σ's of the macro rule are replaced with distinct selectors x_{i_j}. These selectors will retrieve distinct versions of argument i from the old argument lists. The work of repeatedly adding new elements to the argument lists is done by non-terminal B_1. Non-terminal B_2 initializes the structure by moving the new argument lists generated by the previous macro derivation step to the position of the old argument lists for the processing of the current macro derivation step. Step 3d terminates the OI processing by joining the first arguments from each of the argument lists, as required by σ.

The two lemmas below and the theorem that follows verify that the OI method produces a grammar which is yield-equivalent to the original. The first lemma analyzes the rules generated in step 3c of the method.

LEMMA 6-7. Suppose 3-d forest grammar G is generated from a simple OI macro grammar M by the OI construction method. Assume M has a derivation $S \Rightarrow^* A(\alpha_1, \alpha_2, \ldots, \alpha_m) \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$ with $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$ as the last type 2 rule applied, and terminal strings s_{i_j}, $1 \leqslant j \leqslant q$, are independently derived from α_i, $1 \leqslant i \leqslant m$. Let p_i be the number of occurrences of x_i over $(\sigma_1, \sigma_2, \ldots, \sigma_m)$. Also, assume G has a derivation $A \Rightarrow^* \zeta$ such that $fr_2(\zeta) = \#[_2\, \nu_1 [_2\, \zeta_1][_1\, \#[_2\, \nu_2 [_2\, \zeta_2][_1$ $\ldots \#[_2\, \nu_m [_2\, \zeta_m]] \ldots]]]]$, $\zeta_i = \#[_2\, \#[_2\, \delta_{i_1}]\, [_1\, \#[_2\, \#[_2\, \delta_{i_2}][_1$ $\ldots \#[_2\, \#[_2\, \delta_{i_q}]] \ldots]]]]$, and $fr_1(\delta_{i_j}) = omap\,(\sigma_{i_j})$. G also has a derivation $B_i \Rightarrow^* \beta$ which applies r rules for B_1, one rule for B_2, and which has ζ as a subforest. Then $fr_2(\beta) = \#[_2\, \#[_2\, \zeta'_1 [_1\, \gamma'_1]]\, [_1\, \#[_2\, \zeta'_2 [_1\, \gamma'_2]][_1$ $\ldots \#[_2\, \zeta'_m [_1\, \gamma'_m]] \ldots]]]]$, $\gamma'_i = \#[_2\, \#[_2\, \gamma_{i_1}]\, [_1\, \#[_2\, \#[_2\, \gamma_{i_2}][_1$ $\ldots \#[_2\, \#[_2\, \gamma_{i_r}]] \ldots]]]]$, for $1 \leqslant i \leqslant m$, ζ'_i is ζ_i with $r \times p_i$ elements removed from the front, and $fr_1(\gamma_{i_t}) = omap\,(w_{i_t})$ if and only if w_{i_t}, $1 \leqslant t \leqslant r$, are independently derived strings of terminals such that $\beta_i \Rightarrow^* w_{i_t}$.

PROOF. The lemma is established by proving propositions (A) and (B) below.

(A) If $\beta_i \Rightarrow^* w_{i_t}$, $1 \leqslant i \leqslant m$, $1 \leqslant t \leqslant r$, then $fr_2(\beta) = \ldots$ and $fr_1(\gamma_{i_t}) = omap\,(w_{i_t})$.

Proof of (A) by induction on r.

Base. $r = 1$.

The derivation of β is $B_1 \Rightarrow \#[_3\, \xi][_2\, \#[_3\, \theta\,][_2\, \zeta]]$ where $\xi =$ $\#[_2\, \#[_2\, o_1 [_1\, \#[_2\, \#[_2\, \sigma'_1][_1\, n_1]]]]\, [_1\, \ldots \#[_2\, o_m [_1\, \#[_2\, \#[_2\, \sigma'_m]\, [_1\, n_m]]]] \ldots]]$, and

$\theta = \#[_2 \#[_2 n_1[_1 \#[_2 z]]] [_1 \#[_2 n_2[_1 \#[_2 z]]] [_1 \ldots \#[_2 n_m [_1 \#[_2 z]]] \ldots]]]$. For each $1 \leqslant i \leqslant m$, $\sigma'_i = omap(\sigma_i)$, $o_i = <2, 21^{i-1}2(21)^{p_i}>$, and $n_i = <2, 21^{i-1}21>$. $fr_2(\beta) = \#[_2 \#[_2 \zeta'_1[_1 \#[_2 \#[_2 \gamma'_1]][_1 \#[_2 z]]]]][_1 \ldots \#[_2 \zeta'_m [_1 \#[_2 \#[_2 \gamma'_m] [_1 \#[_2 z]]]]] \ldots]]$. The tree ζ'_i is selected from ζ by composing selectors o_i and n_i. The path of n_i leads to the i th argument list ζ_i, and the path of o_i removes the first p_i elements from ζ_i. The forest γ_{i_1} is formed by substituting ζ into θ and then into σ'_i. σ'_i has selectors x_{h_j} of the form $<2, 21_{h-1}22(12)^{j-1}>$ for $1 \leqslant h \leqslant m$ and $1 \leqslant j \leqslant p_i$. The path x_{h_j} leads to the selector n_h. The composition of x_{h_j} and n_h forms the path $21^{h-1}21 \cdot 2(12)^{j-1}$, which selects δ_{h_j} from ζ.

It needs to be shown that if $\beta_i \Rightarrow^* w_{i_1}$, then $fr_1(\gamma_{i_1}) = omap(w_{i_1})$. β_i is the macro substitution of $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ into σ_i to form a string of terminals and non-terminals. Let β_i have the form $\rho_1\alpha_{v_1}\rho_2\alpha_{v_2} \ldots \rho_l \alpha_{v_l}\rho_{l+l}$, where $\sigma_i = \rho_1 v_1 \rho_2 v_2 \ldots \rho_l v_l \rho_{l+l}$. The ρ's are possibly empty strings of terminals and non-terminals, and the v's are possibly distinct macro variables. Each v is replaced by some α to form β_i from σ_i. The fully expanded β_i has the form $w_{i_1} = y_1 u_{v_1} y_2 u_{v_2} \ldots y_l u_{v_l} y_{l+l}$, where $\rho_f \Rightarrow^* y_f$ and $\alpha_{v_f} \Rightarrow^* u_{v_f}$ for $1 \leqslant f \leqslant l$. There are several possible versions of w_{i_1}, resulting from different expansions of the ρ's and α's. Each u_{v_f} is a version of s_h, where v_f refers to the h th macro argument.

Now consider $fr_1(\gamma_{i_1})$. γ_{i_1} is built from σ'_i, the bracketed version of σ_i. All the terminals and non-terminals of σ_i are in σ'_i. The non-terminals of σ'_i

are context-free, and they can be expanded just as in σ. Therefore, γ_{i_1} has subtrees which yield $y_1, \ldots y_{l+1}$. Between each pair of these subtrees is a subtree δ_{h_j}, selected from ζ using selector x_{h_j}. Selector x_{h_j} corresponds to the j th occurrence of a variable v in σ_i which refers to the h th macro argument. So if w_{i_1} has a substring $u_{v_j} = s_{h_j}$, where v_j is the j th occurrence of a selector which refers to the h th macro argument, then γ_{i_1} has a corresponding subtree δ_{h_j} selected by x_{h_j}. It is given that $fr_1(\delta_{h_j}) = omap(s_{h_j})$. So if $\beta_i \Rightarrow^*$ w_{i_1}, then there's a corresponding γ_{i_1} such that $fr_1(\gamma_{i_1}) = omap(w_{i_1})$.

Inductive hypothesis. Assume (A) is true for $r < n$.

Inductive step. Show (A) is true for $r = n$.

The derivation of β is $B_1 \Rightarrow^* \#[_3\, \xi][_2\, \beta']$ where $\xi =$ $\#[_2\, \#[_2\, o_1[_1\, \#[_2\, \#[_2\, \sigma'_1][_1\, n_1]]]]\, [_1 \ldots \#[_2\, o_m\, [_1\, \#[_2\, \#[_2\, \sigma'_m\,]\, [_1\, n_m\,]]]] \ldots]]$. β' is derived in $r - 1$ steps, so the inductive hypothesis an be applied to establish that $fr_2(\beta')$ is $\#[_2\, \#[_2\, \zeta''_1[_1\, \gamma''_1]]\, [_1\, \#[_2\, \zeta''_2[_1\, \gamma''_2]]\, [_1 \ldots \#[_2\, \zeta''_m\, [_1\, \gamma''_m\,]] \ldots]]]$, $\gamma''_i = \#[_2\, \#[_2\, \gamma_{i_1}][_1\, \#[_2\, \#[_2\, \gamma_{i_2}][_1 \ldots \#[_2\, \#[_2\, \gamma_{i_m}]] \ldots]]]]$, ζ''_i is ζ_i with $(r-1) \times p_i$ elements removed, and $fr_1(\gamma_{i_t}) = omap(w_{i_t})$, $2 \leqslant t \leqslant r$. To get $fr_2(\beta)$, $fr_2(\beta')$ is substituted into ξ. The selector o_i selects a sublist of ζ''_i with p_i elements removed, giving ζ'_i which is ζ_i with $r \times p_i$ elements missing. The selector n_i selects γ''_i. By an argument similar to that used for the inductive base above, the substitution into σ'_i results in γ_{i_1} such that $fr_1(\gamma_{i_1}) =$ $omap(w_{i_1})$. So $fr_2(\beta) = \#[_2\, \#[_2\, \zeta'_1[_1\, \gamma'_1]]\, [_1\, \#[_2\, \zeta'_2[_1\, \gamma'_2]]\, [_1 \ldots \#[_2\, \zeta'_m\, [_1\, \gamma'_m\,]]$ $\ldots]]]$ where $\gamma'_i = \#[_2\, \#[_2\, \gamma_1][_1\, \#[_2\, \#[_2\, \gamma_2][_1 \ldots \#[_2\, \#[_2\, \gamma_r\,]] \ldots]]]]$ and

$fr_1(\gamma_{i_t}) = omap(w_{i_t})$, for $1 \leqslant i \leqslant m$ and $1 \leqslant t \leqslant r$.

(B) If $fr_2(\beta) = \#[_2\, \#[_2\, \zeta'_1[_1\, \gamma'_1]]\, [_1\, \#[_2\, \zeta'_2[_1\, \gamma'_2]]\, [_1\, \ldots \#[_2\, \zeta'_m[_1\, \gamma'_m]] \ldots]]]$ where $\ldots$ and $fr_1(\gamma_{i_t}) = omap(w_{i_t})$, then $\beta_i \Rightarrow^* w_{i_t}$ for $1 \leqslant i \leqslant m$, and $1 \leqslant t \leqslant r$.

The proof of (B) is similar to the proof of (A). QED.

LEMMA 6-8. Suppose a 3-d forest G is constructed from a simple OI macro grammar M by the OI construction method. M has a derivation $S \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$ in which the last type 2 rule applied is $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$, and w_{i_t}, $1 \leqslant i \leqslant m$ and $1 \leqslant t \leqslant r$, are independently derived strings of terminals such that $\beta_i \Rightarrow^* w_{i_t}$ if and only if G has a derivation $B \Rightarrow^* \beta$ where $fr_2(\beta) = \#[_2\, \#[_2\, \zeta'_1[_1\, \gamma_1]]\, [_1\, \#[_2\, \zeta'_2[_1\, \gamma_2]]\, [_1\, \ldots \#[_2\, \zeta'_m[_1\, \gamma_m]] \ldots]]]$, $\gamma_i = \#[_2\, \#[_2\, \gamma_{i_1}][_1\, \#[_2\, \#[_2\, \gamma_{i_2}][_1 \ldots \#[_2\, \#[_2\, \gamma_{i_r}]] \ldots]]]]$, and $fr_1(\gamma_{i_t}) = omap(w_{i_t})$.

PROOF. The lemma is established by proving propositions (A) and (B) below.

(A) If M has $S \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$ in which $\ldots$, and $\ldots$ $\beta_i \Rightarrow^* w_{i_j}$, then G has $B \Rightarrow^* \beta$, $fr_2(\beta) = \ldots$, and $fr_1(\gamma_{i_j}) = omap(w_{i_j})$.

Proof of (A) by induction on r, the number of type 2 macro rules applied to derive $B(\beta_1, \beta_2, \ldots, \beta_m)$.

Base. $r = 1$.

The macro derivation is $S \Rightarrow A(\alpha_1, \alpha_2, \ldots, \alpha_m) \Rightarrow B(\beta_1, \beta_2, \ldots, \beta_m)$ using rules $S \rightarrow A(\alpha_1, \alpha_2, \ldots, \alpha_m)$ and $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$. The α's and β's are strings of terminals and context-free non-terminals. Let s_{i_j}, $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant q$, be terminal strings such that $\alpha_i \Rightarrow^* s_{i_j}$, and let

w_{i_t} be terminals strings such that $\beta_i \Rightarrow^* w_{i_t}$, $1 \leqslant t \leqslant r$. By steps 3b and 3c of the OI construction method, G has rules $A \rightarrow \#[_2 \#[_2 \#[_1 T_1]] [_1 \#[_2 \#[_1 T_2]] [_1 \ldots \#[_2 \#[_1 T_m]] \ldots]]]$, $T_i \rightarrow \#[_2 \#[_2 \alpha'_i][_1 T_i]]$, $T_i \rightarrow \#[_2 \#[_2 \alpha'_i]]$, $B \rightarrow B_1$, $B_1 \rightarrow \#[_3 \#[_2 \xi]][_2 B_1]$, $B_1 \rightarrow \#[_3 \#[_2 \xi]][_2 B_2]$, and $B_2 \rightarrow \#[_3 \#[_2 \theta]][_2 A]$, where $\xi = \#[_2 o_1[_1 \#[_2 \#[_2 \sigma'_1][_1 n_1]]]] [_1 \ldots \#[_2 o_m [_1 \#[_2 \#[_2 \sigma'_m] [_1 n_m]]]] \ldots]$, $\theta = \#[_2 n_1[_1 \#[_2 z]]] [_1 \#[_2 n_2[_1 \#[_2 z]]] [_1 \ldots \#[_2 n_m [_1 \#[_2 z]]] \ldots]]$, $o_i = <2, 21^{i-1}2(21)^{p_i}>$, $n_i = <2, 21^{i-1}21>$, and p_i is the number of occurrences of macro variable x_i in $(\sigma_1, \sigma_2, \ldots, \sigma_m)$. The rules for A can be used to derive $\zeta = \#[_2 \#[_2 \#[_1 \zeta'_1]] [_1 \#[_2 \#[_1 \zeta'_2]] [_1 \ldots \#[_2 \#[_1 \zeta'_m]] \ldots]]]$ where $\zeta'_i = \#[_2 \#[_2 \alpha'_{i_1}][_1 \#[_2 \#[_2 \alpha'_{i_2}][_1 \ldots \#[_2 \#[_2 \alpha'_{i_q}]] \ldots]]]]$, and $fr_1(\alpha'_{i_j}) = omap(s_{i_j})$, $1 \leqslant j \leqslant q$. Let $B \Rightarrow B_1 \Rightarrow^* \beta$, applying the B_1 rule r times, the B_2 rule once, and rules for A to derive ζ. Now apply lemma 6-7 to establish the desired result.

Inductive hypothesis. Assume (A) is true for $r < n$.

Inductive step. Show (A) is true for $r = n$.

The macro derivation is $S \Rightarrow A(\alpha_1, \alpha_2, \ldots, \alpha_m) \Rightarrow B(\beta_1, \beta_2, \ldots, \beta_m)$ where the last rule applied is $A(x_1, x_2, \ldots, x_m) \rightarrow B(\sigma_1, \sigma_2, \ldots, \sigma_m)$. The α's and β's are strings of terminals and context-free non-terminals. Let s_{i_j}, $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant q$, be terminal strings such that $\alpha_i \Rightarrow^* s_{i_j}$, and let w_{i_t} be terminals strings such that $\beta_i \Rightarrow^* w_{i_t}$, $1 \leqslant t \leqslant r$. By step 3c of the OI construction method, G has rules $B \rightarrow B_1$, $B_1 \rightarrow \#[_3 \#[_2 \xi]][_2 B_1]$, $B_1 \rightarrow \#[_3 \#[_2 \xi]][_2 B_2]$, and $B_2 \rightarrow \#[_3 \#[_2 \theta]][_2 A]$, where $\xi = \#[_2 o_1[_1 \#[_2 \#[_2 \sigma'_1][_1 n_1]]]]$

$[_1 \ldots \#[_2 o_m [_1 \#[_2 \#[_2 \sigma'_m] [_1 n_m]]]] \ldots]$, $\theta = \#[_2 n_1 [_1 \#[_2 z]]]$ $[_1 \#[_2 n_2 [_1 \#[_2 z]]] [_1 \ldots \#[_2 n_m [_1 \#[_2 z]]] \ldots]]$, $o_i = <2, 21^{i-1}2(21)^{p_i}>$, $n_i = <2, 21^{i-1}21>$, and p_i is the number of occurrences of macro variable x_i in $(\sigma_1, \sigma_2, \ldots, \sigma_m)$. The inductive hypothesis can be applied to establish that $A \Rightarrow^* \zeta$ such that $fr_2(\zeta) = \#[_2 \#[_2 \nu_1 [_1 \zeta_1]] [_1 \#[_2 \nu_2 [_1 \zeta_2]] [_1 \ldots \#[_2 \nu_m [_1 \zeta_m]] \ldots]]]$, $\zeta_i = \#[_2 \#[_2 \delta_{i_1}][_1 \#[_2 \#[_2 \delta_{i_2}][_1 \ldots \#[_2 \#[_2 \delta_{i_q}]] \ldots]]]]$, and $fr_1(\delta_{i_j}) = omap(s_{i_j})$, $1 \leqslant j \leqslant q$. There is a derivation $B \Rightarrow B_1 \Rightarrow^* \beta$, which applies the rule for B_1 r times, the B_2 rule once, and rules for A to derive ζ. Now lemma 6-7 can be applied to establish the desired result.

(B) If G has $B \Rightarrow^* \beta$, $fr_2(\beta) = \#[_2 \#[_2 \zeta'_1 [_1 \gamma_1]] [_1 \#[_2 \zeta'_2 [_1 \gamma_2]] [_1 \ldots \#[_2 \zeta'_m [_1 \gamma_m]] \ldots]]]$, ... and $fr_1(\gamma_{i_t}) = omap(w_{i_t})$, then M has $S \Rightarrow^* B(\beta_1, \beta_2, \ldots, \beta_m)$ in which ..., and $\beta_i \Rightarrow^* w_{i_t}$ for $1 \leqslant i \leqslant m$ and $1 \leqslant t \leqslant r$. The proof of (B) is similar to the proof of (A). QED.

THEOREM 6-9. Suppose a 3-d forest grammar G is constructed from simple OI macro grammar M by the OI construction method. M has a derivation $S \Rightarrow^* w$, where w is a terminal string, if and only if G has a derivation $S \Rightarrow^* \beta$ and $fr_1(\beta) = omap(w)$.

PROOF. First, assume M has $S \rightarrow B(\beta_1, \beta_2, \ldots, \beta_m) \Rightarrow^* w$, and the last macro rule applied is $B(x_1, x_2, \ldots, x_m) \rightarrow \sigma$. Let u_{i_j} be strings of terminals such that $\beta_i \Rightarrow^* u_{i_j}$, for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant r$. By lemma 6-8, G has a derivation $B \Rightarrow^* \zeta$ where $fr_2(\zeta) = \#[_2 \#[_2 \zeta'_1 [_1 \gamma_1]] [_1 \#[_2 \zeta'_2 [_1 \gamma_2]] [_1 \ldots \#[_2 \zeta'_m [_1 \gamma_m]] \ldots]]]$, $\gamma_i = \#[_2 \#[_2 \gamma_{i_1}][_1 \#[_2 \#[_2 \gamma_{i_2}][_1 \ldots \#[_2 \#[_2 \gamma_{i_r}]] \ldots]]]]$, and $fr_1(\gamma_{i_t}) =$

$omap(u_{i_t})$, $1 \leqslant t \leqslant r$. By step 3d of the OI method, G has a rule $S \rightarrow \#[_3 \#[_2 \sigma']][_2 B]$. So $S \Rightarrow^* \#[_3 \#[_2 \sigma']][_2 \zeta]$. By the same argument as in the proof of lemma 6-7, it can be shown that the frontier substitution of $fr_2(\zeta)$ into the expansion of σ' yields the same string as the macro substitution of $(\beta_1, \beta_2, \ldots, \beta_m)$ into σ.

A similar argument can be developed from the initial assumption that G has $S \Rightarrow^* \#[_3 \#[_2 \sigma']][_2 B] \Rightarrow^* \beta$. In both cases, we find that $fr_1(\beta) = omap(w)$. QED.

Consider the OI macro grammar M which has the following rules:

1) $S \rightarrow F(A)$,

2) $F(x) \rightarrow F(xx)$,

3) $F(x) \rightarrow xx$,

4) $A \rightarrow bA$,

5) $A \rightarrow Ab$, and

6) $A \rightarrow a$.

The language generated by M is the same as Y_0(BA2N) from the grammar BA2N of chapter 5. We can construct a new version of this grammar, BA2N', to illustrate the OI construction method. Much of the detail of the OI method is unneeded since the macro non-terminal F takes only one argument. Consequently, occurrences of the macro variable x are converted to selectors whose paths do not have the prefix $21^{i-1}2$. The rules of BA2N' are the following:

1) $F \rightarrow \#[_2 \#[_1 T]]$,

2) $T \rightarrow \#[_2 \#[_2 A][_1 T]]$,

3) $T \rightarrow \#[_2 \#[_2 A]]$,

4) $F \rightarrow F_1$,

5) $F_1 \rightarrow \#[_3 \#[_2 o [_1 \#[_2 \#[_2 x_1 [_1 x_2]][_1 n]]]]][_2 F_1]$,

6) $F_1 \rightarrow \#[_3 \#[_2 o [_1 \#[_2 \#[_2 x_1 [_1 x_2]][_1 n]]]]][_2 F_2]$,

7) $F_2 \rightarrow \#[_3 \#[_2 n [_1 \#[_2 z]]]][_2 F]$,

8) $S \rightarrow \#[_3 \#[_2 x_1 [_1 x_2]]][_2 F_2]$,

9) $A \rightarrow \#[_2 b [_1 A]]$,

10) $A \rightarrow \#[_2 A [_1 b [_1 z]]]$, and

11) $A \rightarrow \#[_2 a [_1 z]]$.

The selectors are $z = <1, \lambda>$, $o = <2, 22121>$, $n = <2, 21>$, $x_1 = <2, 22>$, and $x_2 = <2, 2212>$. Rules 1 through 3 come from step 3b of the OI method applied to rule 1 of M. Rules 4 through 7 result from applying step 3c to rule 2 of M. Rule 8 is produced when step 3d is applied to rule 3 of M. Rules 9 through 11 result from applying step 3a to rules 4, 5, and 6 of M.

The cycle introduced in rule 2 and 3 is strictly increasing. The cycle of rules 5 and 6 is a 0-gain cycle which penetrates deeper into an old argument list with each step as it generates a new argument list. The new argument list is shorter by half than the old list, since two elements are removed from the old list every time an element is added to the new list. When rules 4 and 7 are combined with rules 5 and 6, we have a nested cycle. Rule 7 changes a new list to an old list for further processing by the inside cycle. If l is the size of the initial argument list generated by T, then the maximum number of F_1-cycle steps is $l / 2 + l / 4 + l / 8 +$

$\ldots + l \mid l = l - 1$, and the maximum number of full F-cycle steps is $\log_2 l$. So it must be true that the number of derivation steps involving rules 4 through 7 does not exceed $3 \log_2 l + l - 1$, which is less than $4l$. The F-cycle, therefore, is a linear cycle.

If the T-cycle or F-cycle is applied more times than necessary, some sub-forests will be truncated when the frontier is taken. It is also true, however, that each forest in the 2-d yield is the frontier of a 3-d forest which is complete. So the grammar has a deletion constant of zero.

As a matter of convenience, non-explicit subtrees $\#[_2 z]$ are introduced by step 3c. These serve as end-of-list markers. An equivalent explicit grammar can be written without these by adding separate end-of-list rules. As a result of analyzing the grammar BA2N', we conclude that Y_1(BA2N) $\in$ 1Y3F$_l$. This result is generalized to hold for any 3-d forest grammar constructed by the IO or OI conversion method.

THEOREM 6-10. If G is a 3-d forest grammar constructed from an IO or OI macro grammar using either the IO or OI construction method, then $Y_1(G) \in$ 1Y3F$_l$.

PROOF. It needs to be established that there is G' such that $Y_1(G') = Y_1(G)$ and G' is explicit, G' has a deletion constant, and every cycle in G' is increasing or linear. Macro grammars can be written so that no arguments are deleted, so we will assume that the initial macro grammar is non-deleting. If the IO method is used, then the resulting grammar has no overlapping selectors, and previous theorems assure us that G' can be constructed to meet the above requirements.

Suppose the OI method is used. Non-explicit subtrees are introduced in step 3c, but they can be easily eliminated by adding separate rules to process the last elements on argument lists. Subforests may be truncated during the frontier operation if the argument lists are too long, but for every string yielded, there is a derivation of a forest whose argument lists are exactly the right lengths. So each string in the 1-d yield can be produced by a complete forest, and the grammar has deletion constant 0.

The cycles generated by step 3b of the OI method are strictly increasing. A cycle introduced by step 3c for a non-terminal B_1 generates new argument lists from old argument lists. For each new argument created, a certain number, say r, of arguments are removed from one of the old argument lists. We only need to consider the case where $r > 1$ for some argument list. If r never exceeds 1, then the grammar is non-copying, and an equivalent IO grammar can be written for the language. So the number of steps in each B_1-cycle (and the length of the new argument list) is no more than $l \,/\, r$, where l is the number of arguments in the longest argument list. Step 3c also creates a cycle for B which contains B_1 as a subcycle. At the beginning of each repetition of the B-cycle, the previous new argument lists become the current old argument lists. Since the new argument list is always shorter than the old list by a factor of $1 \,/\, r$, the maximum number of B-cycle repetitions is $\log_r l$. Each full B-cycle step applies 3 rules plus the number of B_1-cycle steps. So the maximum number of rules applied in the B-cycle is

$$\sum_{i=1}^{\log_r l} \left(3 + \frac{l}{r^i}\right)$$

$$= 3 \log_r l + \sum_{i=1}^{\log_r l} \frac{l}{r_i}$$

$$= 3 \log_r l + l \sum_{i=1}^{\log_r l} \frac{1}{r_i}$$

$$\leqslant 3 \log_r l + l$$

$$\leqslant 4\, l.$$

It has been established that G', the explicit equivalent of G, has a deletion constant and only increasing or linear cycles. By theorem 5-51, $Y_1(G') \epsilon$ $1Y3F_l$. QED.

The IO and OI conversion methods have been described for very simple macro languages to keep the level of detail at a managable level. These methods can be generalized to allow various-sized argument lists, larger cycles, multiple non-nested macro non-terminals in the right-hand sides of rules, and context-free non-terminals and terminal strings arbitrarily placed between macro non-terminals. The methods cannot be generalized in a straightforward manner to accommodate nested macro variables. Baldwin's conversion method assures us that every IO macro grammar has a corresponding 3-d forest grammar, but conversion methods for nested OI macro grammars have not been investigated. Thus, it remains an open question whether $1Y3F_l$ contains all or just some of the OI macro languages.

The OI conversion method can be adapted to operate on IO macro grammars as well as OI macro grammars. To simulate OI expansion, the OI method maps repeated occurrences of macro variables in a single rule to distinct 2-d selectors. If repeated macro variables are mapped to the same 2-d selectors, then the new grammar simulates IO expansion. This raises the possibility of using the adapted OI method to convert simple quoted macro grammars to 3-d forest grammars. The forest grammars that result, however, may have mixed 0-gain and increasing overlap cycles. So the present analysis does not guarantee that these 3-d grammars will yield languages in $1Y3F_l$.

CHAPTER 7.

CONCLUSIONS AND FURTHER WORK

The question which has motivated this research is the following: What is the simplest formal system that is capable of completely describing common high-level programming languages? Context-free languages are too simple; context-sensitive languages are too complicated. The answer is somewhere in between.

Several extensions of the context-free languages have been investigated which permit the copying and deletion of substrings. The ALG_n^1 language hierarchy is such an extension in which each level of the hierarchy has more copying power. A formal system must have the ability to copy substrings in order to model the passage of arguments to subroutines. ALG_3^1 has this ability, and it is unlikely that the increased copying power of higher levels of the hierarchy is useful for describing programming languages.

The 1YnF hierarchy defined in this thesis has the same copying power as the ALG_n^1 hierarchy, but the ability to delete subforests is enhanced by extended selectors. As a result of this modification, 1YnF broadens the ALG_n^1 language classes for $n >= 2$ (and perhaps for $n = 2$), and 1Y3F is more capable of describing programming languages than ALG_3^1. If 1Y3F is to be truly useful for this purpose, however, it must be proven that 1Y3F is not too broad.

The use of extended selectors allows 3-d forest grammars to specify cycles which delete arbitrarily long sequences of characters from yielded strings. There is little support in the Computer Science literature for analysis of this kind of deletion. Context-sensitive grammars are commonly defined with the restriction that the left-hand side of a rewriting rule must not be larger than the right-hand side. This characterization is somewhat misleading, for it seems to imply that deletion is not tolerated at all in context-sensitive langauges. It is known that deletion is tolerated if no more than a constant number of consecutive symbols are deleted from a derived word. Chapter 4 goes further than this, demonstrating that context-free languages tolerate the deletion of arbitrarily long substrings, provided there is a regular grammar which describes the deleted substrings, and the deleting and increasing operations are independent. There is, however, a subset of 1Y2F which has not been shown to be within context-free. These languages are generated by grammars which have complex cycles that intermix simultaneous decreasing/increasing steps with increasing steps. It seems likely that further analysis will show these grammars also to have context-free equivalents, or at worst context-sensitive equivalents.

When 3-d forest grammars are analyzed, it becomes apparent that even a restricted subclass of 1Y3F has more languages that ALG_3^1 . This subclass, 1Y3F$_l$, is shown in chapter 5 to be a subset of the context-sensitive languages. This is shown in spite of the fact that the grammars which yield 1Y3F$_l$ languages can have cycles which delete arbitrarily large subforests. The result in chapter 6 that 1Y3F$_l$ contains some OI as well as IO languages is particularly encouraging. All high-level programming languages exhibit both OI and IO phenomena. Evaluation of arithmetic expressions and parameter passage using call-by-value are common examples of IO phenomena. The passage of functions as parameters, call-by-name in Algol, and quoted arguments in Lisp are examples of OI phenomena. A formal system which fully describes high-level programming languages must clearly have both OI and IO capabilities. It remains to be investigated whether $1Y3F_l$ is powerful enough for this application.

Opportunities for further research in multidimensional forest languages are plentiful. With regard to 2-d forest languages, it needs to be determined whether $1Y2F \equiv CF$. If these classes are not equivalent, then it should be determined whether $1Y2F \subset CS$. For 3-d forest languages, it is unknown whether $1Y3F \equiv 1Y3F_l$. If they are not equivalent, then a study could be made to determine if $1Y3F - 1Y3F_l$ contains any interesting languages. It may be true that languages outside the linear subclass have no forseeable applications, and in that case, modifications of the forest grammar and frontier definitions might lead to a more precise characterization of $1Y3F_l$. In any case, it will be interesting to discover whether $1Y3F_l$ shares the properties that are known to hold for context-free and/or context-sensitive languages.

Further research will also be required to determine whether $1Y3F_l$ contains all the OI macro languages. If $1Y3F_l$ is to be useful for describing programming languages, then it will have to contain at least some of the nested OI and quoted macro languages. It is likely that $1Y3F_l$ can also be shown to contain the yields of top-down and bottom-up tree transducers on regular sets of trees, and perhaps even the yields of compositions of tree transducers.

The 1YnF languages share the copying power of the ALG_n^1 languages. Further work can be done to verify that 1YnF forms a proper hierarchy of languages parallel to ALG_n^1, and to generalize the results for 1Y3F to 1YnF, $n \geqslant 3$.

Finally, there are a number of other variations of the frontier function which raise intriguing questions. One such variation would allow whole forests to be copied during the frontier operation. With both standard and extended selectors, only trees can be selected for copying. If selectors were redefined to allow specification of a degree, k, as well as a dimension, n, then the selection path could be traced to retrieve a forest in $H_n^k(\Sigma, \Xi)$ for copying. This is a natural generalization of standard selectors. It is likely to produce a language hierarchy distinct from ALG_n^1 and perhaps from 1YnF as well.

BIBLIOGRAPHY

Aggarwal, Sarwan K., and James A. Heinen. 1979. "A General Class of Non-context-free Grammars Generating Context-free Languages." *Information and Control* 43: 187-194.

Aho, Alfred V. 1968. "Indexed Grammars." *Journal of the Association for Computing Machinery* 15: 647-676.

Aho, Alfred V. 1969. "Nested Stack Automata." *Journal of the Association for Computing Machinery* 16: 383-406.

Aho, Alfred V., and Jeffrey D. Ullman. 1977. *Principles of Compiler Design.* Addison-Wesley Publishing Company, Reading, Mass.

Bader, Christopher, and Arnaldo Moura. 1982. "A Generalization of Ogden's Lemma." *Journal of the Association for Computing Machinery* 29: 404-407.

Baker, Brenda S. 1974. "Non-context-free Grammars Generating Context-free Languages." *Information and Control* 24: 231-246.

Baker, Brenda S. 1975. "Tree Transducers and Tree Languages." *Information and Control* 37: 241-266.

Baker, Brenda S. 1978. "Generalized Syntax Directed Translation, Tree Transducers, and Linear Space." *SIAM Journal on Computing* 7: 376-391.

Baker, Brenda S. 1979. "Composition of Top-down and Bottom-up Tree Transducers." *Information and Control* 41: 186-213.

Baldwin, William A. 1983. "Hypertrees -- A Study in Language Specification." Ph. D. dissertation, Iowa State University.

Book, Ronald. 1972. "Terminal Context in Context-sensitive Grammars." *SIAM Journal on Computing* 1: 20-30.

Choffrat, Christian, and Kavel Culik II. 1983. "Properties of Finite and Pushdown Transducers." *SIAM Journal on Computing* 12: 300-315.

Engelfriet, Joost. 1975a. "Bottom-up and Top-down Tree Transformations -- a Comparison." *Mathematical Systems Theory* 9: 198-231.

Engelfriet, Joost. 1975b. "Tree Automata and Tree Grammars." Unpublished course notes. Department of Computer Science, University of Aarhus, Denmark.

Engelfriet, Joost. 1977. "Top-down Tree Transducers with Regular Look-ahead." *Information and Control* 10: 289-303.

Engelfriet, Joost. 1981. "Three Hierarchies of Transducers." *Mathematical Systems Theory* 15: 95-125.

Engelfriet, Joost, and Erik Meineche Schmidt. 1977. "IO and OI. Part I." *Journal of Computer and System Sciences* 15: 328-353.

Engelfriet, Joost, and Erik Meineche Schmidt. 1978. "IO and OI. Part II." *Journal of Computer and System Sciences* 16: 67-99.

Engelfriet, Joost, and S. Skyum. 1982. "The Copying Power of One-state Tree Transducers." *Journal of Computer and System Sciences* 25: 418-435.

Engelfriet, Joost, Grzegorz Rozenberg, and Giora Slutzki. 1980. "Tree Transducers, L-systems, and Two-way Machines." *Journal of Computer and System Sciences* 20: 150-202.

Engelfriet, Joost, Erik Meineche Schmidt, and Jan Van Leeuwen. 1980. "Stack Machines and Classes of Non-nested Macro-languages." *Journal of the Association for Computing Machinery* 27: 96-117.

Fischer, Michael J. 1968. "Grammars with Macro-like Productions." *IEEE Conference Record 9th Annual Symposium on Switching Automata Theory* 131-142.

Ginsburg, Seymour, and Sheila A. Greibach. 1966. "Mappings which Preserve Context-sensitive Languages." *Information and Control* 9: 563-582.

Ginsburg, Seymour, and Sheila A. Greibach. 1969. "Abstract Families of Languages." Pp. 1-32 in *Studies in Abstract Families of Languages*. Memoir no. 87, American Mathematical Society, Providence, R. I.

Ginsburg, Seymour, and Barbara Partee. 1969. "A Mathematical Model of Transformational Grammars." *Information and Control* 15: 297-334.

Ginsburg, Seymour, and Gene F. Rose. 1966. "Preservation of Languages by Transducers." *Information and Control* 9: 153-176.

Greibach, Sheila A. 1981. "Formal Languages: Origins and Directions." *Annals of the History of Computing* 3(1): 14-41.

Herman, Gabor T., and Grzegorz Rozenberg. 1975. *Developmental Systems and Languages*. North-Holland Publishing Company, Amsterdam.

Hopcroft, John E., and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, Reading, Mass.

Horowitz, Ellis, and Sartaj Sahni. 1976. *Fundamentals of Data Structures*. Computer Science Press, Inc., Woodland Hills, California.

Lewis, P. M., and R. E. Stearns. 1968. "Syntax Directed Transduction." *Journal of the Association for Computing Machinery* 15: 465-488.

Liu, L., and P. Weiner. 1973. "An Infinite Hierarchy of Intersections of Context-free Languages." *Mathematical Systems Theory* 7: 185-192.

Platek, Martin, and Petr Sgall. 1978. "A Scale of Context-sensitive Languages: Applications to Natural Language." *Information and Control* 24: 155-162.

Rounds, William C. 1969. "Context-free Grammars on Trees." *1st Theory of Computing* 143-148.

Rounds, William C. 1970. "Mappings and Grammars on Trees." *Mathematical Systems Theory* 4(3): 257-287.

Savitch, Walter J. 1973. "How to Make Arbitrary Grammars Look Like Context-free Grammars." *SIAM Journal on Computing* 2: 174-182.

Siromoney, Rani, and Kamala Krithivasan. 1974. "Parallel Context-free Languages." *Information and Control* 24: 155-162.

Strawn, George O. 1982. "Hypertrees and the Modified Chomsky Hierarchy." Unpublished manuscript. Department of Computer Science, Iowa State University, Ames, Iowa.

Thatcher, James W. 1973. "Tree Automata: An Informal Survey." Pp. 143-147 in *Currents in the Theory of Computing*. Prentice Hall, Englewood Cliffs, N. J.