HKUST Theoretical Computer Science Center Research Report HKUST-TCSC-2000-01

Normal Form Algorithms for Extended Context-Free Grammars*

Jürgen Albert† Dora Giammarresi‡ Derick Wood§

January 14, 2000

Abstract

We investigate the complexity of a variety of normal-form transformations for extended context-free grammars, where by extended we mean that the set of right-hand sides for each nonterminal in such a grammar is a regular set. The study is motivated by the implementation project GraMa which will provide a C++ toolkit for the symbolic manipulation of context-free objects just as Grail does for regular objects.

Our results generalize known complexity bounds for context-free grammars but do so in nontrivial ways. Specifically, we introduce a new representation scheme for extended context-free grammars (the symbol-threaded expression forest), a new normal form for these grammars (dot normal form) and new regular expression algorithms.

1 Introduction

In the 1960's, extended context-free grammars were introduced, based on Backus–Naur form, as a useful abbreviatory notation that made context-free grammars easier to write. More recently, the Standardized General Markup Language (SGML) [16] used a similar abbreviatory notation to define extended context-free grammars for documents. Currently, Extensible Markup Language (XML) [6], which is a simplified version of SGML, is being promoted as the markup language for the web, instead of HTML (a specific grammar or document type definition (DTD) specified using SGML). These developments led to the investigation of how notions applicable to context-free grammars could be carried over to extended context-free grammars. There does not appear to have been any consolidated effort to study extended context-free grammars in their own right. We begin such an investigation with the most basic

*This research was supported under a grant from the Research Grants Council of Hong Kong SAR. It was carried out while the first and second authors were visiting HKUST.

†Lerhstuhl für informatik II, Universität Würzburg, Am Hubland, D-97074 Würzburg, Germany. E-mail: albert@informatik.uni-wuerzburg.de.

‡Dipartimento di Matematica Applicata e Informatica, Università Ca' Foscari di Venezia, via Torino 155, 30173 Venezia Mestre, Italy. E-mail: dora@dsi.unive.it.

§Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong SAR. E-mail: dwood@cs.ust.hk.

problems for extended context-free grammars: reduction and normal-form transformations. There has been some related work that is more directly motivated by SGML issues; see the proof of decidability of structural equivalence for extended context-free grammars [4] and the demonstration that SGML exceptions do not add expressive power to extended context-free grammars [17].

We are currently designing a manipulation system toolkit GraMa for extended context-free grammars, pushdown machines and context-free expressions. It is an extension of Grail [20, 19], a similar toolkit for regular expressions and finite-state machines. As a result, we need to choose appropriate representations of grammars and machines that admit efficient transformation algorithms (as well as other algorithms of interest).

Earlier results on context-free grammars were obtained by Harrison and Yehudai [12, 13, 26] and by Hunt *et al.* [15] among others. Harrison's chapter on normal form transformations [12] provides an excellent survey of early results. Cohen and Gotlieb [5] suggested a specific representation for context-free grammars and demonstrated how it aided the programming of various operations on them.

We first define extended context-free grammars using the notion of production schemas that are based on regular expressions. In a separate paper [9], we discuss the algorithmic impact of basing the schemas on finite-state machines. Since finite-state machines and regular expressions are both first-class objects in Grail, they can be used interchangeably as we expect they will be in GraMa.

We next describe algorithms for the fundamental normal-form transformations in Section 3. Before doing so, we propose a representation for extended context-free grammars as regular expression forests with symbol threads. We then discuss some algorithmic problems for regular expressions before tackling the various normal forms. We close the presentation, in Section 4, with a brief discussion of our ongoing investigations.

2 Notation and terminology

We treat extended context-free grammars as context-free grammars in which the right-hand sides of productions are regular expressions. Let V be an alphabet. Then, we define a regular expression over V and its language in the usual way [1, 25] with the Kleene plus as an additional operator. The symbol λ denotes the null string.

An **extended context-free grammar** G is specified by a tuple (N, Σ, P, S), where N and Σ are disjoint finite alphabets of **nonterminal symbols** and **terminal symbols,** respectively, P is a finite set of **production schemas,** and the nonterminal S is the **sentence symbol.** Each production schema has the form $A \rightarrow E_A$, where A is a nonterminal and E_A is a regular expression over $V = N \cup \Sigma$ that does not contain the empty-set symbol. When $\beta = \beta_1 A \beta_2 \in V^*$, $A \rightarrow E_A \in P$, and $\alpha \in L(E)$, the string $\beta_1 \alpha \beta_2$ can be derived from the string β and we denote this fact by writing $\beta \Rightarrow \beta_1 \alpha \beta_2$. The **language L(G) of an extended context-free grammar G** is the set of terminal strings derivable from the sentence symbol of G. Formally, $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^+ w\}$, where $\Rightarrow^+$ denotes the transitive closure of the derivability relation.

Even though a production schema may correspond to an infinite number of ordinary context-free productions, it is known that extended and standard context-free grammars describe exactly the same languages; for example, see the texts of Salomaa [23] and of Wood [25].

We denote the size of a regular expression E by $|E|$ and define it as the number of symbols and operators in E. We denote the size of a set A also by $|A|$. To measure the complexity of any grammatical transformation we need to define the size of a grammar. There are two traditional measures of the size of a context-free grammar that we generalize to extended context-free grammars as follows. Given an extended context-free grammar $G = (N, \Sigma, P, S)$, we define the **size** $|G|$ **of G** to be

$$\sum_{A \in N} (1 + |E_A|)$$

and we define the **norm** $\| G \|$ **of G** to be

$$|G| \lg |N \cup \Sigma|.$$

Clearly, the norm is a more realistic measure of a grammar's size as it takes into account the size of the encoding of the symbols of the grammar. We use only the size measure however, since the extension of our results to the norm measure is straightforward.

3 Normal-form transformations

We introduce the notion of an expression forest that is a tree-based representation for the set of regular expressions that appear as right-hand sides of production schemas. Each production schema's right-hand side is represented as an expression tree in the usual way, internal nodes are labeled with operators and external nodes are labeled with symbols. In addition, we represent the nonterminal left-hand side of a production schema with a single node labeled with that nonterminal. The node also references the root of the expression tree of its corresponding right-hand side. In Fig. 3, we give an example forest of two regular expressions.

Since an extended context-free grammar has a number of production schemas that are regular expressions, we represent such grammars as an expression forest, where each tree in the forest corresponds to one production schema and each tree is named by its corresponding nonterminal. (The naming avoids the tree repetition problem.) We now add threads such that the **thread for symbol** X connects all appearances of the symbol X in the expression forest.

3.1 Reachability and usefulness

Recall that a symbol X is **reachable** if it appears in some string derived from the sentence symbol; that is, if there is a derivation $S \Rightarrow^* \alpha X \beta$ where α and β are (possibly null) strings over $\Sigma \cup N$.

As in standard context-free grammars, reachable symbols can be easily identified by means of a digraph traversal. More precisely, we construct a digraph whose vertices are symbols in $N \cup \Sigma$ and there is an edge from A to B if and only if B labels an external node of the expression tree named A. (We assume that the production schemas do not contain

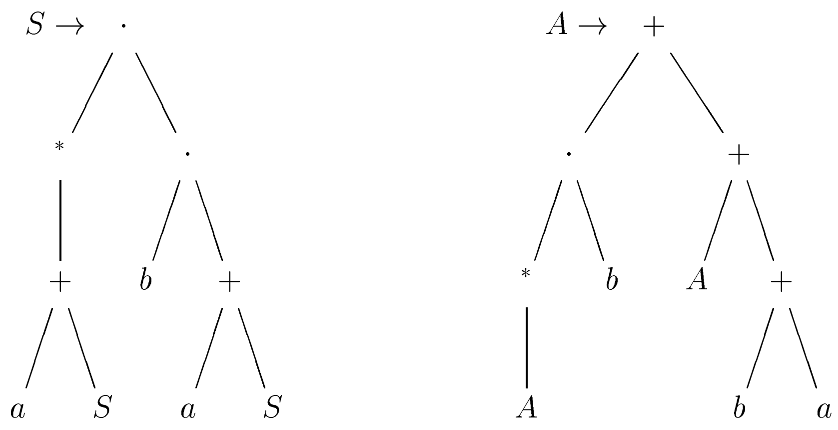

Figure 1: An expression forest for the extended context-free grammar with production schemas: $S \to (a+S)^*b(a+S)$ and $A \to A^*b + (A + (b+a))$. We have omitted the symbol threads for clarity.

the empty-set symbol.) Then, a depth-first traversal of this digraph starting from S gives all reachable symbols of the grammar. The times taken by the digraph construction and traversal are both linear in the size of the grammar.

A nonterminal symbol A is **useful** if there is a derivation $A \Rightarrow^+ \alpha$, where α is a terminal string. The set of useful symbols can be computed recursively as follows. Compute all symbols B such that $L(E_B)$ contains a string of terminal symbols (possibly the null string). All such B are useful symbols. Then, a symbol A is useful if $L(E_A)$ contains a string of terminals and the currently detected reachable symbols, and so on until no newly useful symbols are identified. We can formalize this inductive process with a marking algorithm such as described by Wood [24] for context-free grammars. The major difference between previous work and the approach taken here is that we want to obtain an efficient algorithm. Yehudai [26] designed an efficient algorithm for determining usefulness for context-free grammars; our approach can be viewed as a generalization of his algorithm.

To explain the marking algorithm, we assume that we have one bit available at each node of the expression forest to indicate the marking. We initialize these bits in a preorder traversal of the forest as follows: The bits of all nodes are set to zero (unmarked) except for nodes that are labeled with a Kleene star symbol, a terminal symbol or a null-string symbol—the bits of these nodes are set to one (marked). In the algorithm, whenever a node u is marked, it is useful and it satisfies the condition: *The language of the subtree rooted at u contains a string that is completely marked.* A Kleene-star node is marked since its subtree's language contains the null string; that is, a Kleene-star node is always useful.

After completing the initial marking, we bubble markings up the trees in a propagation phase as follows: Repeatedly examine newly marked nodes as follows until no newly marked nodes are obtained. For each newly marked node u, where $p(u)$ is u's parent if u is not the root, perform one of the following actions:

if $p(u)$ **is a plus node** and $p(u)$ is not marked, then mark $p(u)$.

if $p(u)$ **is a dot node,** $p(u)$ is not marked and u's sibling is marked, then mark $p(u)$.

if $p(u)$ **is a Kleene-plus node,** then mark $p(u)$.

if $p(u)$ **is a Kleene-star node,** it is already marked.

if u **is a root node** and the expression tree's nonterminal symbol is not marked, then mark the expression tree's nonterminal symbol.

If there are newly marked nonterminals after this initial round, then we mark all their appearances in the expression forest and repeat the propagation phase which bubbles the markings of newly marked symbols up the trees. If there are no newly marked nonterminals, then the algorithm terminates.

The algorithm has, therefore, a number of rounds and at the beginning of each round it marks all appearances of newly discovered useful nonterminals (discovered in the previous round) and then bubbles the newly marked nonterminals up the trees. As long as a round marks new nodes, the propagation process is repeated. To implement this process efficiently, we construct, at the beginning of each round, a queue of newly marked nodes. Note that the queue is a catenation of appearance lists after the first round. The algorithm then repeatedly deletes a newly marked node from the queue and, using the preceding propagation rules, it may also add newly marked nodes to the queue. A round terminates when the queue is empty.

Observe that each node of the expression forest is visited at most twice because a dot node can be visited twice. Thus, the marking algorithm runs in $O(|G|)$ time and space.

Recall that a grammar G is **reduced** if all its symbols are both useful and reachable. As for standard context-free grammars, to reduce a grammar we first identify all useful symbols and then select (together with the corresponding schemas) those that are also reachable.

More formally, after identifying the useless nonterminals (terminals are always useful), we first remove their production schemas from G. Second, we remove all productions (not schemas) that contain a useless nonterminal in their right-hand sides. In both steps we have to ensure that the threads are maintained correctly. In the first step, we need not only to remove the production schemas, but also to reconnect the threads of of all symbol appearances that are removed. We can use a traversal of each schema to identify the symbols in it and remove their appearances from the appropriate threads. In the second step, we use the threads of useless symbols to remove the corresponding productions. We simply replace each useless symbol with the empty-set symbol and remove it from its thread, and then apply the empty-set removal algorithm for regular expressions to each production schema. Thus, we obtain the equivalent grammar $\bar{G} = (\bar{N}, \bar{\Sigma}, \bar{P}, S)$.

We next identify the unreachable symbols of $\bar{G}$ and then remove the production schemas of the unreachable nonterminals and, once more, maintain the threads correctly. Observe that an unreachable terminal symbol can only appear in production schemas of unreachable nonterminals and that reachable symbols can only appear in production schemas of reachable nonterminals. Thus, we obtain G' from $\bar{G}$ in linear time.

We summarize the result of this section as follows.

Theorem 1 *Let $G = (N, \Sigma, P, S)$ be an extended context-free grammar represented as an expression forest. Then, an equivalent, reduced extended context-free grammar $G' = (N', \Sigma', P', S)$ can be constructed from G in time $O(|G|)$ such that $|G'| \leq |G|$, $|N'| \leq |N|$, $|\Sigma'| \leq |\Sigma|$ and $|P'| \leq |P|$. Moreover, G' is also represented as an expression forest.*

3.2 Null-free form

Given a reduced extended context-free grammar $G = (N, \Sigma, P, S)$, we can determine the nullable nonterminals (the ones that derive the null string) using a similar algorithm to the one we used for usefulness in Section 3.1. This algorithm takes $O(|G|)$ time. Given the nullability information we can then make the given grammar null free in two steps.

First, we replace all appearances of each nullable symbol A with the regular expression $(A + \lambda)$. This step takes time proportional to the total number of appearances of nullable symbols in G—we use the symbol threads for fast access to them. Second, we transform each production schema $A \rightarrow E_A$, where A is nullable, into a null-free production schema $A \rightarrow E'_A$, where $\lambda \notin L(E'_A)$. Unfortunately, this step can take time $O(2^{|G|})$ in the worst case when we use the typical textbook algorithm and each production schema has nested dotted subexpressions in which each operand of the dot can produce the null string. We replace each dotted subexpression $F \cdot G$ with $F_{-} + G_{-} + (F_{-} \cdot G_{-})$, where F_{-} is the transformed version of F that does not produce the null string. Note that we at least double the length of the dotted subexpressions. Because similar doubling can occur in the subexpressions of F and G and of their subexpressions, we obtain the exponential worst-case bound. (Note that this is the same case that occurs with a standard context-free grammar in which every nonterminal is nullable.)

We want, however, to obtain at most a linear blowup in the size of the resulting grammar. Since nested dot expressions cause the nonlinearity, we modify the grammar to remove nested dot expressions. This approach was first suggested by Yehudai [13, 26] for standard context-free grammars—he converted a given grammar into Chomsky normal form to avoid the dot problem. We take a similar approach by removing nested dot, Kleene-plus and Kleene-star subexpressions from production schemas. The removal generates new nonterminals and their production schemas; however, the size of the resulting grammar is only linearly larger than the original grammar.

We replace each dot, Kleene-plus and Kleene-star node of an expression tree that has a dot, Kleene-plus or Kleene-star ancestor with a new nonterminal and add a new production schema to the grammar. We repeat this local modification until no such nested nodes exist. For example, given the production schema

$$A \rightarrow (((a + b + \lambda) \cdot (b + \lambda))^{+} \cdot (c + d + \lambda)) \cdot (e + \lambda));$$

we can replace it with the new production schemas:

$$A \rightarrow (B \cdot (c + d + \lambda)) \cdot (e + \lambda)$$

and

$$B \rightarrow ((a + b + \lambda) \cdot (b + \lambda))^{+}.$$

Repeating the transformation for B, we obtain

$$B \rightarrow (C)^{+}.$$

$$C \rightarrow (a + b + \lambda) \cdot (b + \lambda).$$

Repeating the transformation for A, we obtain

$$A \rightarrow D \cdot (e + \lambda)$$

and

$$D \rightarrow B \cdot (c + d + \lambda).$$

We say that the resulting grammar is in **dot normal form.** Its size is of the same order as the original size and the number of nonterminals is increased by at most the size of the original grammar.

Lemma 2 *Let $G = (N, \Sigma, P, S)$ be a reduced, extended context-free grammar represented as an expression forest. Then, an equivalent, reduced extended context-free grammar $G' = (N', \Sigma, P', S)$ in dot normal form can be constructed from G in time $O(|G|)$ such that $|G'|$ is $O(|G|)$, $|N'|$ is $O(|G|)$ and $|P'|$ is $O(|G|)$. Moreover, G' is also represented as an expression forest.*

We now apply the simple null-removal construction on a grammar G in dot normal form to produce a new grammar that has size $O(|G|)$. The algorithm runs in time $O(|G|)$.

Theorem 3 *Let $G = (N, \Sigma, P, S)$ be a reduced, extended context-free grammar in dot normal form represented as an expression forest. Then, an equivalent, reduced, null-free extended context-free grammar $G' = (N', \Sigma, P', S)$ in dot normal form can be constructed from G in time $O(|G|)$ such that $|G'|$ is $O(|G|)$, $|N'|$ is $O(|N|)$ and $|P'|$ is $O(|P|)$. Moreover, G' is also represented as an expression forest.*

3.3 Unit-free form

A **unit production** is a production of the form $A \rightarrow B$. We transform an extended context-free grammar into unit-free form in three steps. First, we identify all instances of unit productions. Second, we remove each unit-production instance from its schema. Third and last, for each modified schema, we add the unit-free schemas of the unit-production instances to the modified schema.

We now discuss these three steps in more detail. We assume that each reduced, null-free, extended context-free grammar G, is also in dot normal form. To identify instances of unit productions observe that, for each schema E_A, each root-to-frontier path contains at most one dot or Kleene-plus node, and no Kleene-star nodes. Now, assume that there is a unit-production instance of B in E_A (that is, $A \rightarrow B$ is in $A \rightarrow E_A$). Immediately, none of B's ancestors can be dot nodes; an ancestor can be a plus node and at most one ancestor can be a Kleene-plus node. To identify unit productions, we carry out a preorder traversal of E_A and identify root-to-frontier paths that satisfy the necessary conditions for unit-production instances and also have a nonterminal at their frontier nodes. This step takes $O(|E_A|)$ time. Whenever the traversal meets a Kleene-plus node or a plus node it continues the traversal in the corresponding subtrees. When it meets a dot node it terminates that part of the traversal. When eventually the traversal reaches a node labeled with a nonterminal B, then that occurrence of B corresponds to a unit production for A. The overall running time for the first step is $O(|G|)$.

Second, we remove the instances of unit productions from their schemas. That is, we transform each production schema $A \rightarrow E_A$ into a production schema $A \rightarrow E'_A$ such that $L(E'_A) = L(E_A) - N$. We define a new threading, which we refer to as the **unit thread** that connects all occurrences of nonterminals that correspond to unit productions in the schemas. The threading can be constructed during the identification step but it is used in the second step. Furthermore, while identifying unit productions, we determine, for each nonterminal A, the set U_A of nonterminals that are unit reachable from A. (Note that U_A may contain A.) We use these sets to modify the production schemas in the third step.

We traverse the expression trees from their frontiers to their roots and, in particular, we follow the paths that start from the nodes labeled with nonterminals that correspond to unit productions (we access them by following the unit threads). Assume that we are removing an instance of B. Then, its ancestors are only plus nodes with the possible exception that one ancestor is a Kleene-plus node.

To remove unit appearances from Kleene-plus subtrees, we globally transform all Kleene-plus subexpressions of the form F^+ in the expression forest into $(F + (F \cdot F^+))$. The idea behind this global transformation is that we have separated the unit appearances in F^+ from the non-unit appearances of the same symbols in F^+, since the unit appearances now occur only in the first F and the non-unit appearances of the same symbols appear in the subexpression $(F \cdot F^+)$. If node u is a Kleene-plus node in some expression tree, then we make two copies of u's only subtree R (we call them S and T) and ensure we maintain all threads in S and T except for the unit threads. We then remove the Kleene-plus node and reconnect R, S and T as $(R + (S \cdot (T^+)))$.

The removal of all unit appearances of each nonterminal B is now straightforward. We arrive at a node labeled B by following the unit thread and we replace B and B's parent with B's sibling and terminate the process for this occurrence of B. The only case we have not covered is when $A \rightarrow B$ is the only production for A. In this case, B has no parent; therefore, we are left, temporarily, with an empty expression tree for A. (Note that $B \neq A$ since A is useful.)

The time complexity of this second step is the same as that of the first step.

Third and last, we modify the production schemas such that, for each nonterminal A, if $B_1, \ldots, B_l$ are the nonterminal symbols that are unit reachable from A that do not include A, then the new production schema for A is

$$A \rightarrow E'_A + E'_{B_1} + \cdots + E'_{B_l}.$$

The resulting grammar has size $O(|G|^2)$, a quadratic blow up, since we must make copies of the E'_{B_i} subtrees to give an expression tree for A. The algorithm takes, therefore, $O(|G|^2)$ time.

Theorem 4 *Let $G = (N, \Sigma, P, S)$ be a reduced, null-free extended context-free grammar in dot normal form that is represented as an expression forest. Then, an equivalent, reduced, dot-normal-form, null-free, unit-free extended context-free grammar $G' = (N', \Sigma, P', S)$ can be constructed from G in time $O(|G|^2)$ such that $|G'|$ is $O(|G|^2)$, $|N'| \leq |N|$ and $|P'|$ is $O(|G|)$. Moreover, G' is also represented as an expression forest.*

Note that we can ensure that the blow up is linear, if we do not make multiple copies of the various subtrees, but we merely provide links to one copy of each distinct subtree. This approach takes $O(|N|^2)$ time and adds $O(|N|^2)$ extra space to the grammar G.

3.4 Greibach form

This normal form result for context-free grammars was established by Sheila Greibach [10] in the 1960's; it was a key result in the use of the multiple-path syntactic analyzer developed at Harvard University at that time. An extended context-free grammar is in *Greibach normal form* if its productions are of only the following form:

$$A \rightarrow a\alpha,$$

where a is a terminal symbol and α is a possibly empty string of nonterminal symbols. The transformation of an extended context-free grammar into Greibach normal form requires two giant steps: left-recursion removal and back left substitution. Recall that a grammar is left recursive if there is a nonterminal A such that $A \Rightarrow^+ A\alpha$ in the grammar, for some string α. We consider the second step first.

Assume that the given extended context-free grammar $G = (N, \Sigma, P, S)$ is reduced, factored, null free and unit free. (A grammar is **factored** if, for each nonterminal A, a string x in $L(E_A)$ is either completely nonterminal or it is a single terminal symbol. It is straightforward to factor a grammar and if we do it before we make the grammar null free, we avoid the possible introduction of unit productions.)

In addition, for the second step we also assume that the grammar is non-left recursive. Since the grammar is non-left recursive there is a partial order on the nonterminals, **left reachability,** that is defined by $A \leq B$ if there is a leftmost derivation $A \Rightarrow^* B\alpha$. As usual, we can consider the nonterminals to be enumerated as $A_1, \ldots, A_n$ such that whenever $A_i \leq A_j$, then $i \leq j$. Observe that A_n is already in Greibach normal form since it has only productions of the form $A_n \rightarrow a$, where $a \in \Sigma$. We now convert the nonterminals one at a time from A_{n-1} down to A_1. The conversion is conceptually simple, yet computational expensive. When converting A_i, we replace all nonterminals that can appear in the first positions in the strings in $L(E_{A_i})$ with their schemas. Thus, the resulting schema $A_i \rightarrow E'_{A_i}$ is now in Greibach normal form. To be able to carry out this substitution efficiently we first convert each schema E_{A_i} into **first normal form;** that is, we express each schema as the sum of regular expressions each of which begins with a unique symbol. More precisely, letting $\Sigma = \{a_{n+1}, \ldots, a_{n+m}\}$ and using the notation E_i instead of E_{A_i}, for simplicity, we replace E_i by $\bar{E}_i$ which is defined as follows:

$$\begin{aligned} \bar{E}_i \;=\; & A_{i+1} \cdot E_{i,i+1} + \cdots + A_n \cdot E_{i,n} \\ & + \delta^i_{n+1} \cdot a_{n+1} + \cdots + \delta^i_{n+m} \cdot a_{n+m}, \end{aligned}$$

where $\delta^i_{n+k} = \{\lambda\}$, if $a_{n+k} \in L(E_i)$, and $\delta^i_{n+k} = \emptyset$, if $a_{n+k} \notin L(E_i)$. The conversion must satisfy $L(\bar{E}_i) = L(E_i)$. Then, to convert $\bar{E}_i$ into an equivalent regular expression E'_i in Greibach normal form, we need only replace the first A_k of each term $A_k \cdot E_{i,k}$ with E'_k.

If the grammar is left recursive, we first need to make it non-left recursive. We use a technique introduced by Greibach [11], investigated in detail by Hotz and his co-workers [14, 21, 22] and rediscovered by others [7]. It involves producing, for each nonterminal, a distinct subgrammar of G that is left linear; hence, it can be converted into an equivalent right linear grammar. This conversion changes left recursion into right recursion and does not introduce any new left recursion. For more details, see Wood's text [25]. The important property of the left-linear subgrammars is that every sentential leftmost derivation sequence in G can be mimicked by a sequence of leftmost derivation sequences, each of which is a sentential leftmost derivation sequence in one of the left-linear subgrammars. Once we convert the left-linear grammars into right-linear grammars this property is weakened in that we mimic the original derivation sequence with a sequence of sentential rightmost derivation sequences in the right-linear grammars. The new grammar that is equivalent to G is the collection of the distinct right-linear grammars, one for each nonterminal in G.

As the modified grammar is no longer left recursive, we can now apply back left substitution to obtain a final grammar in Greibach normal form.

How well does this algorithm perform? Left recursion removal causes a blow up of $|N||G|$ at worst. Converting the production schemas into first normal form causes an additional blow up of $|N||G|$. We use the derivative $\frac{\mathrm{d}E}{\mathrm{d}X}$ of a regular expression E by a symbol X to give a new expression F such that $L(F) = \{y : Xy \in L(E)\}$ and $\{X\}L(F) \subseteq L(E)$. The derivative of a regular expression was introduced by Brzozowski [3] who defined it inductively. Now, given a schema E_A, we obtain its derivatives for each symbol $X \in N \cup \Sigma$. When we catenate X with its derivative we obtain one of the terms in the first normal form. Since G is null free and unit free, the only derivative that can cause exponential blow up is $\frac{\mathrm{d}F^+}{\mathrm{d}X} \cong \frac{\mathrm{d}F}{\mathrm{d}X} \cdot F^*$. We transform G such that no Kleene-plus subexpression is nested within any other Kleene-plus expression—a similar transformation to the one we used for conversion to dot normal form. This modification ensures that exponential blow up does not occur. The back left substitution causes, in the worst case, an additional blow up of $|N||G|$ in the size of the Greibach normal form grammar.

As all three operations take time proportional to the sizes of their output grammars essentially, the transformation to Greibach normal form takes $O(|N|^5|G|^2)$ time, in the worst case. The reason for the $|N|^5$ term is that we first remove left recursion which not only increases the size of the grammar but also squares the number of nonterminals from $|N|$ to $|N|^2$. The number of nonterminals is crucial in the size bound for the grammar obtained by first normal form conversion and by back left substitution.

We can however, reduce the worst-case time and space by using indirection as we did for unit-production removal. Rather than performing the back left substitution for a specific nonterminal, we use a reference to its schema. This technique gives a blowup of only $|G|+|N|^2$ at most; thus, it reduces the complete conversion time and size to $O(|N|^3|G|)$ in the worst case.

We may also apply the technique that Koch and Blum [18] suggested; namely, leave unit-production removal until after we have obtained a Greibach-like normal form. Moreover, transforming an extended context-free grammar into dot normal form appears to be a very useful technique to avoid undesirable blow up in grammar size. We are currently investigating this and other approaches.

4 Discussion

The results that we have presented are truly a generalization of similar results for context-free grammars. The time and space bounds are similar when relativized to the grammar sizes. The novelty of the algorithms is four-fold. First, we have introduced the regular expression forest with symbol threads as an efficient data representation for context-free grammars and extended context-free grammars. We believe that this representation is new. The only previously documented representations are those of Cohen and Gotlieb [5] and of Barnes [2] and they are more simplistic. Second, we have demonstrated how indirection using referencing can save time and space in the null-production removal and back left substitution algorithms. Although the use of the technique is novel in this context, it is well known technique in other applications. It is an application of lazy evaluation or evaluation on demand. Third, we have introduced dot normal form for extended context-free grammars that plays a role similar to Chomsky normal form for standard context-free grammars. Fourth, we have generalized the left-linear to right-linear grammatical conversion for extended grammars.

We are currently investigating whether we can obtain Greibach normal form more efficiently and whether we can improve the performance of unit-production removal.

We would like to mention some other applications of the regular expression forest with symbol threads. First, we can reduce usefulness determination to nullability determination. Given an extended context-free grammar $G = (N, \Sigma, P, S)$, we can replace every appearance of every terminal symbol with the null string to give $G' = (N, \emptyset, P', S)$. Now, a nonterminal A in G is useful if and only if it is nullable in G'. Second, we can use the same algorithm to determine the length of the shortest terminal strings generated by each nonterminal symbol. The idea is that we replace each appearance of a terminal symbol with the integer 1 and each appearance of the null string with 0. We then repeatedly replace: each node labeled "+" that has two integer children with the minimum of the two integers; each node labeled "·" that has two integer children with the sum of the two integers; and each node labeled "*" with 0. The root value is the required length. We can use the same "generic" algorithm to compute the smallest terminal alphabet for the terminal strings derived from a nonterminal, the $LL(1)$ first and follow sets, and so on.

Last, the careful reader will have observed that the space-efficient algorithms for unit freeness and Greibach normal form produce output grammars that are not represented as expression forests—they are represented as a set of expression dags (directed acyclic graphs). The dags have as many roots as there are nonterminals. Not surprisingly, each root-to-frontier traversal yields a tree since we have reduced space by sharing common subtrees among trees in the underlying expression forest. Clearly, we may also share common subtrees within the original trees in the expression forest, although we do not know of any "practical" computation that would benefit from such sharing. We are currently investigating the complexity of the transformations we have discussed when we are given a collection of expressions dags as the representation of an extended grammar. Although, a collection of dags is a dag, the dags we are dealing with have three properties. First, a traversal from any root node yields a tree that corresponds to a production schema; second, there are as many roots as there are nonterminals; and, third, the dags are threaded. For this reason, we call such a collection of expression dags, a **dagwood**[1].

References

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.

[2] K.R. Barnes. Exploratory steps towards a grammatical manipulation package (GRAMPA). Master's thesis, McMaster University, Hamilton, Ontario, Canada, 1972.

[3] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM*, 11:481–494, 1964.

[4] H.A. Cameron and D. Wood. Structural equivalence of extended context-free and extended E0L grammars. To be submitted, 2000.

[5] D.J. Cohen and C.C. Gotlieb. A list structure form of grammars for syntactic analysis. *Computing Surveys*, 2:65–82, 1970.

[6] D. Connolly. W3C web page on XML. http://www.w3.org/XML/, 1997.

[7] A. Ehrenfeucht and G. Rozenberg. An easy proof of Greibach normal form. *Information and Control*, 63:190–199, 1984.

[8] Timothy S. Freeman, Gregory M. Imirzian, Erich Kaltofen, and Lakshman Yagati. Dagwood: A system for manipulating polynomials given by straight-line programs. *ACM Transactions on Mathematical Software*, 14:218–240, 1988.

[9] D. Giammarresi and D. Wood. Transition diagram systems and normal form transformations. In *Proceedings of the Sixth Italian Conference on Theoretical Computer Science*, Singapore, 1998. World Scientific Publishing Co. Pte. Ltd.

[10] S.A. Greibach. A new normal form theorem for context-free phrase structure grammars. *Journal of the ACM*, 12:42–52, 1965.

[11] S.A. Greibach. A simple proof of the standard-form theorem for context-free grammars. Technical report, Harvard University, Cambridge, MA, 1967.

[12] M.A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1978.

[1] Dagwood and Blondie are two comic strip characters that have been around since the 1920's (the strip is called Blondie)—the combination of "dag" and "wood" was too tempting to resist. Interestingly, there is a symbolic computation system called Dagwood [8]; perhaps the authors were also using common subexpressions and expression dags.

[13] M.A. Harrison and A. Yehudai. Eliminating null rules in linear time. *Computer Journal*, 24:156–161, 1981.

[14] G. Hotz. Normal-form transformations of context-free grammars. *Acta Cybernetica*, 4:65–84, 1978.

[15] H.B. Hunt III, D.J. Rosenkrantz, and T.G. Szymanski. On the equivalence, containment and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12:222–268, 1976.

[16] ISO 8879: Information processing—Text and office systems—Standard Generalized Markup Language (SGML), October 1986. International Organization for Standardization.

[17] P. Kilpeläinen and D. Wood. SGML and exceptions. In C. Nicholas and D. Wood, editors, *Proceedings of the Third International Workshop on Principles of Document Processing (PODP 96)*, pages 39–49, Heidelberg, 1997. Springer-Verlag. Lecture Notes in Computer Science 1293.

[18] R. Koch and N. Blum. Greibach normal form transformation revisited. In Reischuk and Morvan, editors, *STACS'97 Proceedings*, pages 47–54, New York, NY, 1997. Springer-Verlag. Lecture Notes in Computer Science 1200.

[19] D. R. Raymond and D. Wood. Grail: Engineering automata in C++, version 2.5. Technical Report HKUST-CS96-24, Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong, 1996.

[20] D.R. Raymond and D. Wood. Grail: A C++ library for automata and expressions. *Journal of Symbolic Computation*, 17:341–350, 1994.

[21] R.J. Ross. *Grammar Transformations Based on Regular Decompositions of Context-Free Derivations*. PhD thesis, Department of Computer Science, Washington State University, Pullman, WA, USA, 1978.

[22] R.J. Ross, G. Hotz, and D.B. Benson. A general Greibach normal form transformation. Technical Report CS-78-048, Department of Computer Science, Washington State University, Pullman, WA, USA, 1978.

[23] A. Salomaa. *Formal Languages*. Academic Press, New York, NY, 1973.

[24] D. Wood. *Theory of Computation*. John Wiley & Sons, Inc., New York, NY, 1987.

[25] D. Wood. *Theory of Computation*. John Wiley & Sons, Inc., New York, NY, second edition, 1998. In preparation.

[26] A. Yehudai. *On the Complexity of Grammar and Language Problems*. PhD thesis, University of California, Berkeley, CA, 1977.