A non-regular leftist language.

Suman Bandyopadhyay* Meena Mahajan†
K Narayan Kumar*

April 23, 2005

Keywords: formal languages, grammars, automata

1 Introduction

A leftist grammar is a rewrite system in which the only kind of production rules allowed are "create left-context" via $a \longrightarrow ba$, and "erase left-context" via $ba \longrightarrow a$. There is a special start symbol, S, which cannot be created or erased, but which can create context. A string w is said to be in $L(G)$ if wS can be transformed through a sequence of rewrite rules to S.

To the best of our knowledge, these leftist grammars have appeared in just one place in the literature, namely, in the work of [8]. There, the authors present schemes for determining accessibility or safety in certain protection systems, which provide the formal basis for trust management. Their model, first proposed in [9, 2] in the context of Java virtual worlds, is a capability-based system. It strictly generalises the grammatical protection systems of [1, 6] (where creation of new objects is disallowed and the take-grant model of [7] (where there is only a restricted set of rights), and is a special case of the general access-matrix model [3] which is undecidable.

The authors of [8] show that accessibility in their model is decidable by mapping it to a membership query in a leftist grammar. They explicitly raise the question of placing these grammars within the Chomsky hierarchy (for basics about formal language theory, see an standard textbook, such as [4, 5]). While all the examples they constructed were context-free, they were unable to show that all leftist languages are context-free, or even context-sensitive.

*Chennai Mathematical Institute, 92 G. N. Chetty Road, T. Nagar, Chennai 600 017, India. {suman,kumar}@cmi.ac.in

†The Institute of Mathematical Sciences, CIT Campus, Taramani, Chennai 600 113, India. meena@imsc.res.in

In this work, we make some progress towards answering this question. We show that leftist languages without any "create-context" rules can be non-regular, by constructing a leftist grammar and proving that the language it generates is non-regular. We use a novel technique of constructing a deterministic infinite-state automaton accepting the language, and then proving that it is minimal. Since the leftist rewrite system has no rules for creating context, it is trivially context-free. Thus we show that leftist systems with no create-context rules are incomparable with regular languages and properly contained in context-free languages.

We also show that over binary alphabets, all leftist languages (even those generated by grammars with create-context rule) are regular, though the converse is not true.

2 Definition and some properties of leftist languages

Definition 1 *A leftist grammar is a rewrite system $G = (V, S, P)$ where $S \in V$ and every rule $\alpha \longrightarrow \beta \in P$ satisfies the following conditions:*

- $1 \leq |\alpha|, |\beta| \leq 2$.
- *If $|\alpha| = 2$, then $\alpha = ab$ where $a \in V \setminus \{S\}$, $b \in V$, and $\beta = b$.*
- *If $\alpha = b \in V$, then $\beta = ab$ for some $a \in V \setminus \{S\}$.*

The grammar G generates the langauge $L(G) = \{w \in V^* \mid S \Longrightarrow_G^* wS\}$.

A leftist language is a langauge generated by some leftist grammar.

By reversing the derivation process, we can see that $w \in L(G)$ iff $wS \Longrightarrow_{G'}^* S$, where G' has rules $\beta \longrightarrow \alpha$ for each $\alpha \longrightarrow \beta \in P$. Indeed, it is in this form that these languages are introduced in [8], as strings that can be completely erased by rules of the prescribed form.

Due to the very limited nature of the rules, we will show that leftist languages have to be highly structured. Thus several regular languages are not leftist.

Definition 2 *For any non-empty word $w \in V^*$, the sets suffix(w), prune(w) and stutter(w) are defined as follows:*

suffix(w) $= \{x \in V^* \mid \exists y, w = yx\}$

stutter(w) $= \{x \in V^* \mid w = a_1 a_2 \dots a_n$ *and* $x \in a_1^+ a_2^+ \dots a_n^+\}$

prune(w) $= \{x \in V^* \mid w \in stutter(x)\}$

We extend these definitions to sets of words in the natural way: $\text{suffix}(L) = \bigcup_{w \in L} \text{suffix}(w)$, $\text{prune}(L) = \bigcup_{w \in L} \text{prune}(w)$ and $\text{stutter}(L) = \bigcup_{w \in L} \text{stutter}(w)$. The proof of the following lemma is quite direct.

Lemma 3 *If L is a leftist language, then suffix(L), prune(L) and stutter(L) all equal L.*

Corollary 4 *The regular language $L = \{a^{2n} \mid n \in \mathbb{N}\}$ is not leftist.*

Proof: This follows from lemma 3 and the fact that $a \notin L$, but $a \in \text{suffix}(L) \cap \text{prune}(L)$. ■

Another useful property of leftist languages is composition:

Lemma 5 *If L is a leftist language, and $w, x \in L$, then $wx \in L$.*

Proof: By assumption, there are derivations $S \Longrightarrow^* wS$ and $S \Longrightarrow^* xS$. Just compose them: $S \Longrightarrow^* wS \Longrightarrow^* wxS$. ■

We now show that any leftist language over a binary alphabet is necessarily regular. Thus to exhibit a non-regular language, we need at least three different terminals, and in the next two sections we establish that three terminals suffice.

Theorem 6 *Let $L \subset V^*$ be a leftist language, where $|V| \leq 2$. Then L is regular.*

Proof: If L is unary, say a subset of a^*, then either L has no non-empty word, or L equals $\text{stutter}(\text{prune}(L))$ which contains $\text{stutter}(a)$ which equals a^+. Either way, L is regular.

So now assume that $V = \{a, b\}$. If both $S \longrightarrow aS$ and $S \longrightarrow bS$ are in P, then it is easy to see that $L(G) = V^*$. If neither rule is there in P, then $L(G) = \emptyset$. Let G have only one "create" rule from S, say $S \longrightarrow aS$. If $a \longrightarrow ba$ is not in P, then b can never be created and the language is unary. If $a \longrightarrow ba$ is in P, then there are two cases: if S can erase a (i.e. rule $aS \longrightarrow S$ is in P), then effectively S can create b and so $L(G) = V^*$. If S cannot erase a, then the rightmost letter of any $w \in L$ must be a. But any such string can be generated, since $S \Longrightarrow aS \Longrightarrow^* b^*aS$ and Lemma 5 applies. Thus $L(G) = V^*a$. Thus in all cases, L is regular. ■

3 The non-regular leftist language

The language we consider is that defined by the following leftist grammar:

Definition 7 *$G = (\{a, b, c, S\}, S, P)$ where P has the following rules:*

$$S \longrightarrow aS \qquad a \longrightarrow ba \qquad b \longrightarrow cb \qquad c \longrightarrow ac$$

The langauge L generated by the above grammar is trivially context-free: it is generated by the grammar $G' = (\{a, b, c\}, \{S, A, B, C\}, S, P')$ where P' has the rules:

$$S \longrightarrow AS \quad A \longrightarrow BA \quad B \longrightarrow CB \quad C \longrightarrow AC \quad A \longrightarrow a \quad B \longrightarrow b \quad C \longrightarrow c$$

We now prove our main result, namely, that L is not regular. One approach for doing this is to show that $L \cap R$ is non-regular for some regular set R. In particular, let $R = \{(bca)^*(cba)^*\}$ and $A = \{(bca)^m(cba)^n \mid m \leq n\}$. Showing that $L \cap R = A$ would suffice. It is easy to see that $A \subseteq L \cap R$; see the Appendix. However, to see that $L \cap R \subseteq A$, we must show that if $m > n$ then $(bca)^m(cba)^n \notin L$. We do not see any easy way of proving this. In particular, one could use the Cocke-Younge-Kasami parser for context-free languages (see eg [4]) to decide membership of these strings. But since this has to be established for all m, n, it leads to an extremely tedious proof. Instead, we present here an automata-theoretic proof which, in our view, is more elegant.

We first construct a deterministic finite-state automaton M with countably infinite states. We establish that $L(G) = L(M)$. Then we argue that the automaton M is minimal, hence concluding that $L(G)$ is non-regular. (Since we already know that $A \subseteq L \cap R$, it would suffice to show that $L \subseteq L(M)$ and $L(M) \cap R \subseteq A$. However, we find presenting the entire proof that $L = L(M)$ more satisfying. Further, we believe that the automaton-based approach would be applicable while considering general leftist languages as well.)

The automaton M is as shown in Figure 1. Formally, it is defined below.

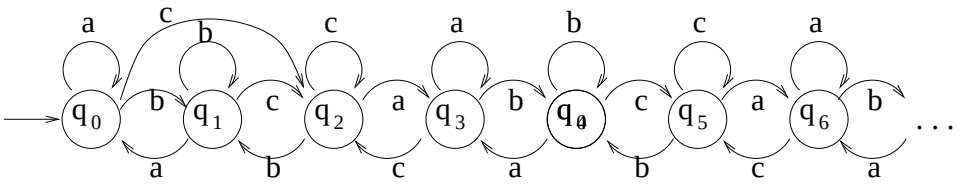

Figure 1: The automaton M

Definition 8 *The automaton M is given by $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{q_0, q_1, q_2, \ldots\} = \{q_i \mid i \geq 0\}$, $\Sigma = \{a, b, c\}$, $F = \{q_0\}$, and δ is defined as follows:*

$i \pmod 3$	0	1	2
$Back_i$	c	a	b
$Front_i$	b	c	a
$Loop_i$	a	b	c

$$\begin{aligned}
\delta(q_i, Back_i) &= q_{i-1} \text{ for } i > 0 \\
\delta(q_0, Back_0) &= q_2 \\
\delta(q_i, Loop_i) &= q_i \text{ for } i \geq 0 \\
\delta(q_i, Front_i) &= q_{i+1} \text{ for } i \geq 0
\end{aligned}$$

The transition function $\delta : Q \times \Sigma \longrightarrow Q$ is extended to $\widehat{\delta} : Q \times \Sigma^* \longrightarrow Q$ in the natural way.

The following are easy observations.

Proposition 9 *For each $i > 0$, $Back_{i+1} = Loop_i = Front_{i-1}$. Also, $Back_1 = Loop_0 = Front_2$. Furthermore, for each $i \geq 0$, the following rules are productions of G, and these are the only productions not involving S.*

$$\begin{aligned}
Loop_i &\longrightarrow Front_i \;\; Loop_i \\
Back_i &\longrightarrow Loop_i \;\; Back_i \\
Front_i &\longrightarrow Back_i \;\; Front_i
\end{aligned}$$

Lemma 10 *For every $w \in \Sigma^*$, and for every state q_i,*

1. *If $\widehat{\delta}(q_i, w) = q_j$ and the $q_i \rightsquigarrow q_j$ path on w does not use the edge $\delta(q_0, c) = q_2$, then $\widehat{\delta}(q_{i+3}, w) = q_{j+3}$.*

2. *If $\widehat{\delta}(q_i, w) = q_j$ and the $q_i \rightsquigarrow q_j$ path on w uses the edge $\delta(q_0, c) = q_2$ at least once, then $\widehat{\delta}(q_{i+3}, w) = q_j$.*

Proof: We induct on $|w|$. For $|w| = 0$, the claim is obvious. Now let $w = w'd$. For any state q_i, consider the path from q_i on w'. If this path uses the anomalous edge $\delta(q_0, c) = q_2$, then by the induction hypothesis, $\widehat{\delta}(q_{i+3}, w') = \widehat{\delta}(q_i, w')$ and so $\widehat{\delta}(q_{i+3}, w) = \widehat{\delta}(q_i, w)$ as well. So now assume that until d is seen, the anomalous edge is not used. Hence by the induction hypothesis, if $\widehat{\delta}(q_i, w') = q_j$, then $\widehat{\delta}(q_{i+3}, w') = q_{j+3}$. We consider two cases:

Case 1: On input d, the anomalous edge is used. This means that $d = c$ and $j = 0$, giving $\widehat{\delta}(q_i, wd) = q_2$. So $\widehat{\delta}(q_{i+3}, w') = q_3$, giving $\widehat{\delta}(q_{i+3}, w'd) = q_2$ because $\delta(q_3, c) = q_2$.

Case 2: On input d, the anomalous edge is not used. But all other transitions from q_j and q_{j+3} are *isomorphic*, for every j. So if $\delta(q_j, d) = q_l$, then $\delta(q_{j+3}, d) = q_{l+3}$.

■

Lemma 11 *If $d \longrightarrow ed$ is a rule in G where $d \neq S$, and $\widehat{\delta}(q_0, wd) = q_j$, then $\widehat{\delta}(q_0, wed) \in \{q_j, q_{j-3}\}$. (In particular, if $j < 3$, then $\widehat{\delta}(q_0, wed) = q_j$.)*

Proof: We induct on $|w|$. Let $\widehat{\delta}(q_0, w) = q_k$ and $\delta(q_k, d) = q_j$. It is easy to verify explicitly that if $j < 3$ (and hence $k \leq 3$), then $\widehat{\delta}(q_0, wed) = q_j$. So we assume now that $j \geq 3$. We consider three cases:

$d = \mathbf{Loop}_k$**:** So $k = j$. By Proposition 9, $\text{Loop}_k \longrightarrow \text{Front}_k\text{Loop}_k$, so $e = \text{Front}_k$. Also, $d = \text{Back}_{k+1}$. So $\widehat{\delta}(q_0, wed) = \widehat{\delta}(q_k, ed) = \delta(q_{k+1}, d) = q_k$.

$d = \mathbf{Back}_k$**:** So $j = k - 1$. By Proposition 9, $\text{Back}_k \longrightarrow \text{Loop}_k\text{Back}_k$, so $e = \text{Loop}_k$. So $\widehat{\delta}(q_0, wed) = \widehat{\delta}(q_k, ed) = \delta(q_k, d) = q_{k-1} = q_j$.

$d = \mathbf{Front}_k$**:** So $j = k + 1$. By Proposition 9, $\text{Front}_k \longrightarrow \text{Back}_k\text{Front}_k$, so $e = \text{Back}_k$. So $\widehat{\delta}(q_0, wed) = \widehat{\delta}(q_k, ed) = \delta(q_{k-1}, d)$. Now $d = \text{Front}_k = \text{Back}_{k+2} = \text{Back}_{k-1}$, so $\delta(q_{k-1}, d) = q_{k-2} = q_{j-3}$.

■

With these two lemmas, we can now establish that $L(G) \subseteq L(M)$.

Lemma 12 *$L(G) \subseteq L(M)$. That is, For all $w \in \Sigma^*$, if $S \Longrightarrow_G^* wS$ then $\widehat{\delta}(q_0, w) = q_0$.*

Proof: The proof procceds by induction on $|w|$. We want to show that for every $w \in \Sigma^*$, if $S \Longrightarrow_G^* wS$ then $\widehat{\delta}(q_0, w) = q_0$. We induct on $|w|$. It is straightforward to verify this for $|w| < 2$. Assume that it is true for all strings of length upto k, and now consider a string w of length $k + 1$.

We zero in on the last step in a derivation of w. If this step uses the rule $S \longrightarrow aS$, then the derivation must be $S \Longrightarrow_G^* w_1 S \Longrightarrow_G w_1 aS = wS$. But by induction, $\widehat{\delta}(q_0, w_1) = q_0$, and so $\widehat{\delta}(q_0, w_1 a) = \delta(q_0, a) = q_0$.

So now assume that the last step in the derivation uses a rule $d \longrightarrow ed$ for $d \in \{a, b, c\}$. So the derivation has the form $S \Longrightarrow_G^* w_1 dw_2 S \Longrightarrow_G w_1 edw_2 S = wS$. Let $\widehat{\delta}(q_0, w_1 d) = q_j$ for some j. By the induction hypothesis, we know that $\widehat{\delta}(q_j, w_2) = q_0$. By Lemma 11 $\widehat{\delta}(q_0, w_1 ed) = q_k \in \{q_j, q_{j-3}\}$.

If $k = j$, then $\widehat{\delta}(q_0, w_1 edw_2) = \widehat{\delta}(q_0, w_1 dw_2) = q_0$, and we are done.

If $k = j - 3$, let us assume that $\widehat{\delta}(q_k, w_2) = q_l$ for some $l > 0$. Then by Lemma 10 above, $\widehat{\delta}(q_j, w_2)$ equals $\widehat{\delta}(q_{k+3}, w_2)$ and is either q_l or q_{l+3}. Either way, it is not q_0. So $\widehat{\delta}(q_0, w_1 dw_2)$ is not q_0, contradicting the induction hypothesis. Hence our assumption must be wrong, and in fact $\widehat{\delta}(q_k, w_2) = q_0$. It follows that $\widehat{\delta}(q_0, w_1 edw_2) = q_0$. ■

It is easy to see that for each state $q_i \in Q$, the shortest string taking M from q_i to q_0 is unique and is of length i. Let r_i be this string; it is nothing but $\text{Back}_i\text{Back}_{i-1}\ldots\text{Back}_1$. More formally, $r_0 = \epsilon$, and for $j \geq 1$, $r_j = \text{Back}_j r_{j-1}$.

For each $w \in \Sigma^*$, define $f(w) = i$ if $\widehat{\delta}(q_0, w) = q_i$.

Lemma 13 *For all $w \in \Sigma^*$, $S \Longrightarrow_G^* wr_{f(w)}S$.*

Proof: The proof proceeds by induction on $|w|$. For the base case, $|w| = 0$, so $w = \epsilon$. Then $f(w) = 0$, $r_{f(w)} = r_0 = \epsilon$, and so the statement is trivially true.

Assume now that the statement is true for all strings of length less than k. Consider a string w of length exactly k. Let $w = xd$ where $d \in \Sigma$, $x \in \Sigma^*$, $|x| = k - 1$. Let $\widehat{\delta}(q_0, x) = q_j$ and $\delta(q_j, d) = q_i$ so that $\widehat{\delta}(q_0, w) = q_i$. By the induction hypothesis, we know that $S \Longrightarrow_G^* xr_jS$. We need to establish that $S \Longrightarrow_G^* xdr_iS$. We consider two cases.

$j = 0$**:** Then $S \Longrightarrow_G^* xS$. Observe that $S \Longrightarrow_G aS \Longrightarrow_G baS \Longrightarrow_G cbaS$.

- $d = a$**:** Then we have $i = 0$, and $r_0 = \epsilon$, so $wr_{f(w)} = w = xa$. From the induction hypothesis and the observation above, we have $S \Longrightarrow_G^* xS \Longrightarrow_G xaS$, proving the claim.
- $d = b$**:** Then we have $i = 1$, and $r_1 = a$, so $wr_{f(w)} = wa = xba$. From the induction hypothesis and the observation above, we have $S \Longrightarrow_G^* xS \Longrightarrow_G^* xbaS$, proving the claim.
- $d = c$**:** Then we have $i = 2$, and $r_2 = ba$, so $wr_{f(w)} = wba = xcba$. From the induction hypothesis and the observation above, we have $S \Longrightarrow_G^* xS \Longrightarrow_G^* xcbaS$, proving the claim.

$j \geq 0$**:** Then $S \Longrightarrow_G^* xr_jS$. Again, there are three cases.

- $d =$ **Back**$_j$**:** In this case, we have $i = j - 1$, and $dr_i = r_j$. The result now follows from the induction hypothesis.
- $d =$ **Loop**$_j$**:** In this case, we have $i = j$ and $r_i = r_j$. From the induction hypothesis and Proposition 9, we have $S \Longrightarrow_G^* xr_jS = x\text{Back}_j r_{j-1}S \Longrightarrow_G x\text{Loop}_j\text{Back}_j r_{j-1}S = xdr_jS$, proving the claim.
- $d =$ **Front**$_j$**:** In this case, $i = j+1$ and $d = \text{Loop}_i$. Now $xdr_i = x\text{Front}_j Back_{j+1}r_j = x\text{Front}_j Loop_j r_j$. From the induction hypothesis and Proposition 9, we have $S \Longrightarrow_G^* xr_jS = x\text{Back}_j r_{j-1} \Longrightarrow_G x\text{Loop}_j\text{Back}_j r_{j-1}S \Longrightarrow_G x\text{Front}_j\text{Loop}_j\text{Back}_j r_{j-1}S = x\text{Front}_j\text{Loop}_j r_jS$, proving the claim.

■

Since $r_0 = \epsilon$, and since $f(w) = 0$ for every $w \in L(M)$, the following corollary is immediate.

Corollary 14 *$L(M) \subseteq L(G)$.*

Lemma 15 *The automaton M is minimal; no two states are equivalent.*

Proof: Simply observe that r_i is of length i for each i. Since $\widehat{\delta}(q_i, r_i) \in F$, while $\widehat{\delta}(q_j, r_i) \notin F$ for any $j > i$, it follows that no two states q_i and q_j are equivalent. ■

Theorem 16 *The leftist language L generated by the grammar defined in Definition 7 is not regular.*

Proof: From Lemma 12 and Corollary 14, it follows that $L(M) = L(G)$. Lemma 15 then shows that M is minimal. And M has infinite states. So no finite-state automaton can accept L. ■

From Corollary 4 and Theorem 16 our main result follows:

Theorem 17 *Leftist languages with only erase rules are incomparable with regular languages.*

Acknowledgments

The second author thanks Chandra Chekuri and Suresh Venkatasubramaniam for bringing this problem to her notice. The first author thanks Ujjyaini Mitra, and the second and third authors thank Rani Siromoney, for useful and enjoyable discussions on the topic.

References

[1] T. Budd. Safety in grammatical protection systems. *International Journal of Computer and Information Sciences*, 12:413–430, 1983.

[2] O. Cheiner and V. Saraswat. Security analysis of matrix. Technical report, AT&T Shannon Laboratory, 1999.

[3] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19:461–470, 1976.

[4] A. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 2001.

[5] D. Kozen. *Automata and Computability.* Springer-Verlag, 1997.

[6] R. Lipton and T. Budd. *On classes of protection systems*, pages 281–296. Academic Press, 1978.

[7] R. Lipton and L. Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24:455–464, 1977.

[8] R. Motwani, R. Panigrahy, V. Saraswat, and S. Ventkatasubramanian. On the decidability of accessibility problems (extended abstract). In *Proceedings of 32nd ACM Symposium on Theory of Computing (STOC)*, pages 306–325, 2000.

[9] V. Saraswat. The matrix design. Technical report, AT&T Laboratory, 1997.

Appendix

Lemma 18 *For every pair of numbers $0 \leq m \leq n$, the string $(bca)^m(cba)^n$ is in $L(G)$.*

Proof: We will show that $\{(cba)^l \mid l \geq 0\} \subseteq L(G)$ and $\{(bca)^m(cba)^m \mid m \geq 0\} \subseteq L(G)$. The result then follows from Lemma 5.

To see the first claim, note that $\underline{S} \Longrightarrow \underline{a}S \Longrightarrow \underline{b}aS \Longrightarrow cbaS$, and $\underline{c} \Longrightarrow \underline{a}c \Longrightarrow \underline{b}ac \Longrightarrow cbac$; thus we have $S \Longrightarrow^3 cbaS$ and $S \Longrightarrow^{3l} (cba)^l S$. (In the derivation, at each step the letter creating an additional letter to its left is underlined.)

We prove the second claim by induction on m. The base case, when $m = 0$, is obvious. Now assume that we have a derivation $S \Longrightarrow^* (bca)^{m-1}(cba)^{m-1}S$. We append to this the following derivation, starting from the first c in the $(cba)^{m-1}$ part: $\underline{c} \Longrightarrow \underline{a}c \Longrightarrow b\underline{a}c \Longrightarrow b\underline{b}ac \Longrightarrow bc\underline{b}ac \Longrightarrow bc\underline{c}bac \Longrightarrow bcacbac$. These six steps insert $(bca)(cba)$ in the middle of the string generated inductively, creating $(bca)^m(cba)^m$ as desired. ■