RAIRO. INFORMATIQUE THÉORIQUE

ROCKFORD ROSS
KARL WINKLMANN

Repetitive strings are not context-free

RAIRO. Informatique théorique, tome 16, n° 3 (1982), p. 191-199
<http://www.numdam.org/item?id=ITA_1982__16_3_191_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 16, n° 3, 1982, p. 191 à 199)

REPETITIVE STRINGS ARE NOT CONTEXT-FREE (*)

by Rockford Ross and Karl WINKLMANN (1) (2)

Communicated by M. NIVAT

Abstract. – *Let* Σ *be an alphabet. A string of the form* $xyyz$ *with* $x, z \in \Sigma^*$ *and* $y \in \Sigma^+$ *is called* repetitive. *In this paper we prove that the set of repetitive strings over an alphabet of three or more letters is not context-free, settling a conjecture from* [1].

Résumé. – *Soit* Σ *un alphabet. Un mot* contient un carré *s'il est de la forme* $xyyz$ *avec* $x, z \in \Sigma^*$ *et* $y \in \Sigma^+$. *Dans cet article, nous prouvons que l'ensemble des mots contenant un carré sur un alphabet à trois lettres n'est pas algébrique, résolvant une conjecture de* [1].

Let Σ be an alphabet. A string of the form yy with $y \in \Sigma^+$ is called a *repetition.* A string which contains a repetition as a substring (i. e. a string of the form $xyyz$ with $x, z \in \Sigma^*$ and $y \in \Sigma^+$) is called a *repetitive string.* Interest in repetitive and nonrepetitive strings dates back at least as far as Thue's 1906 paper [13]. One question which has defied an answer for some time is whether or not the set of repetitive strings over an alphabet of three or more characters is context-free. (For binary alphabets the question is trivial.) This question is settled in this paper, confirming a conjecture from [1].

THEOREM. – *The set of repetitive strings over an alphabet of three or more characters is not context-free.*

Classical techniques for showing languages to be not context-free appear to be of no use in proving this result, *cf.* [1, pp. 374-375]. At the same time, intuition very strongly suggests that repetitive strings are not context-free for the same reason that repetitions (i. e. strings of the form ww with $w \in \Sigma^+$) are not context-free: the first-in-last-out nature of a pushdown store does not provide the means

(*) Received in October 1980, revised in March 1981.
(1) Computer Science Department, Washington State University, Pullman, Washington 99164 U.S.A.
(2) Supported in part by N.S.F. Grant No. MCS-80004128.

R.A.I.R.O. Informatique théorique/Theoretical Informatics, 0399-0540/1982/ 191/$ 5.00

to remember one substring and then check it for equality with another substring – the information needed first for such a check is bound to reside at the bottom of the store.

Thus, the situation is one where strong intuition does not readily translate into a proof. We regard this as a deficiency in the theory of context-free languages, which we expect to repair by first dealing with a special instance (the non-context-freeness of repetitive strings) and then, in a subsequent paper, generalizing our proof technique into a new necessary condition for context-freeness.

The remainder of this paper consists of a proof of the above theorem and some concluding remarks. Ends of proofs are marked with the symbol □.

We first prove that repetitive strings over a six-letter alphabet are not context-free. The extension to three-letter alphabets will follow quite easily, using a result from [2].

Let R be the set of repetitive strings over some fixed alphabet containing at least the six symbols a, b, c, S, 0, and 1. For the sake of deriving a contradiction assume that R is context-free. Let M be a nondeterministic pushdown automaton with $L(M)=R$. Without loss of generality we may assume that M has the following properties:

- it accepts by empty store,
- it has only one internal state,
- it changes its stack height by at most 1 in any single step, and
- it reads one input symbol in every step.

These properties of M are the result of assuming that the grammar for R is given in "2-Greibach-Normal-Form" [5, 9, 11, 12], where all productions are of the form $A \to aBC$, $A \to aB$, or $A \to a$ with A, B, C being syntactic variables and a a terminal symbol. The standard construction of a nondeterministic pushdown automaton from a context-free grammar (*see* e. g. [8], pp. 115-116) then yields a machine with the above properties.

The basic idea behind our proof is to analyze how the *pda* M can store information about its input. Specifically, we are going to exploit the fact that information received (on the input tape) during an early stage of the computation is bound to reside near the bottom of the stack. Information on the stack simply cannot be arbitrarily juggled around. Some of the technical details of such an analysis are simplified by the assumption that the height of the stack changes by at most 1 in any single move. But while convenient, this assumption is not essential. At the cost of adding some technical detail to the proof we could adopt the weaker assumption that there is some constant k, not necessarily 1, such that

M changes its stack height by at most k in any single step. This weaker assumption corresponds to assuming that the grammar for R is given in Greibach-Normal-Form but not necessarily in 2-Greibach-Normal-Form.

Let s denote the size of the stack alphabet of M. Choose two natural numbers, p and q, such that they satisfy:

$$2^p > \sum_{h=0}^{4q} s^h, \qquad (\star)$$

and:

$$2^q > (2p)^2 s \qquad (\star\star)$$

$\sum_{h=0}^{4q} s^h$ is a bound on the number of different stack configurations possible in M up to height $4q$; $(2p)^2 s$ is the number of different triples consisting of one stack symbol and two positions in a string of length $2p$ (i. e. two numbers between 1 and $2p$); and 2^p and 2^q are, of course, the numbers of different binary strings of lengths p and q, respectively. Why we want inequalities ($\star$) and ($\star\star$) to hold will become clear shortly. At the moment just note that numbers p and q satisfying ($\star$) and ($\star\star$) do indeed exist. To see this we can combine ($\star$) and ($\star\star$) into:

$$(2^q/s)^{1/2}/2 > p > \log_2\left(\sum_{h=0}^{4q} s^h\right),$$

and observe that the value of the expression on the left grows exponentially with q whereas the value of the expression on the right grows only linearly with q. Therefore by choosing q large enough we can indeed find an integer p satisfying ($\star$) and ($\star\star$).

Using these chosen numbers we will now construct a set A of strings whose "repetitive properties" are easily understood. We will then show that even on this restricted set A the *pda* M will not be able to distinguish some repetitive strings from "similar" nonrepetitive ones.

The set A is constructed as follows. Let w be a fixed string of length $n=p+q$ over $\{a, b, c, S\}$ with the property that ww contains only one repetition (ww itself). One way to get such a string w is to choose a nonrepetitive string w' of length $n-1$ over $\{a, b, c\}$ and let w be Sw'. (The fact that arbitrarily long nonrepetitive strings over a three-letter alphabet do exist was first shown in [13]. Proofs can also be found in [4, 7, 10, 14] and [6, pp. 36-40]. *See* also [3].) The elements of A, then, will be all strings obtained from ww by inserting a 0 or 1 after each character of ww.

Formally, define for every string $u \in \{0, 1\}^{2n}$ the *interleaving*, $I(u)$, *of* u *with the fixed string* ww to be:

$$I(u) = w_1 u_1 \ldots w_n u_n w_1 u_{n+1} \ldots w_n u_{2n}.$$

(Here $w = w_1 \ldots w_n$ with $|w_i| = 1$ for $1 \leqq i \leqq n$, and $u = u_1 \ldots u_{2n}$ with $|u_j| = 1$ for $1 \leqq j \leqq 2n$. Recall that $w \in \{a, b, c, S\}^*$ and $u \in \{0, 1\}^*$.) Then:

$$A = \{I(u) \mid u \in \{0, 1^{2n}\}.$$

The cardinality of A is 2^{2n}, the number of different binary strings of length $2n$. Furthermore, our choice of w ensures that $I(u)$ is repetitive if and only if u is a repetition, i. e. if and only if $u = xx$ for some $x \in \{0, 1\}^n$. This is stated as Lemma 1.

LEMMA 1: *For all* $x, x' \in \{0, 1\}^n$, $I(xx') \in R$ *if and only if* $x = x'$.

Proof: Straightforward. □

In describing the actions of the *pda* M as it examines an interleaved string $I(u)$, it will be necessary to refer to certain portions of the input $I(u)$. Therefore, we will consistently write u as $yzy'z'$ with $y, y' \in \{0, 1\}^p$ and $z, z' \in \{0, 1\}^q$; thus,

$$I(u) = I(yzy'z')$$
$$= w_1 y_1 \ldots w_p y_p w_{p+1} z_1 \ldots w_{p+q} z_q w_1 y'_1 \ldots w_p y'_p w_{p+1} z'_1 \ldots w_{p+q} z'_q.$$

In $I(yzy'z')$ we will refer to:

$w_1 y_1 \ldots w_p y_p$ as the "first y-portion",
$w_{p+1} z_1 \ldots w_{p+q} z_q$ as the "first z-portion",
$w_1 y'_1 \ldots w_p y'_p$ as the "second y-portion", and
$w_{p+1} z'_1 \ldots w_{p+q} z'_q$ as the "second z-portion".

Informally speaking, any *pda* which can distinguish between repetitive and nonrepetitive strings from A must have the first p 0's and 1's of the input stored on its stack at the moment when the last character of the first y-portion has been read. Otherwise there could not be a comparison with the corresponding characters in the second half of the input. In other words, we expect a typical accepting computation of M (on inputs from A) to use enough storage space to hold at least p bits of information just after the first y-portion of the input string has been read. The following definition makes this notion of "typical" computations precise. Lemma 2 then says, roughly, that those computations indeed deserve to be called "typical".

For every $y \in \{0, 1)^p$, $z \in \{0, 1\}^q$ and computation C on input $I(yzyz)$ define $\text{StackHeight}_C(yzyz)$ to be the height of the stack in computation C immediately after reading last character of the first y-portion of the input $I(yzyz)$, and call C "typical" if $\text{Stack Height}_C(yzyz) > 4q$. (Recall that by $(\star)$ a stack height of $4q$ is not enough to store p bits of information.)

LEMMA 2 : *There is a* $y \in \{0\ 1\}^p$ *such that for each* $z \in \{0, 1\}^q$ *all* accepting *computations* C *on input* $I(yzyz)$ *are typical.*

Proof: Assume to the contrary that for all $y \in \{0, 1\}^p$ there is a $z \{0, 1\}^q$ and there is an accepting computation C on input $I(yzyz)$ with $\text{StackHeight}_C(yzyz) \leqq 4q$. By $(\star)$ there are more strings in $\{0, 1\}^p$, viz. 2^p, than there are stack configurations up to height $4q$, viz. $\sum_{h=0}^{4q} s^h$. Hence there are two strings y and y' in $\{0, 1\}^p$, $y \neq y'$, such that for some z, $z' \in \{0, 1\}^q$ there are accepting computations C and C' (on inputs $I(yzyz)$ and $I(y'z'y'z')$ respectively) where the stack configurations just after reading the first y-portion of the input are the same in C and C'. (Recall that M has only one internal state, hence the stack contents alone determine the future behavior of M on a given input.) Therefore M will also accept $I(yz'y'z')$ and $I(y'zyz)$, neither of which, by Lemma 1, is repetitive. □

To repeat, the intuition behind this Lemma 2 is quite simple: if M is to accept all repetitive strings but no others, then for it to accept a string of the form $I(yzyz)$ it must "remember" the first y-portion to make sure it matches the second one.

Again speaking informally, what is left to show is that if our pushdown automaton M behaves in this "typical" way, which it must by Lemma 2, then something else is bound to go wrong. Specifically, what we will show is that M will be unable to check z against z' in inputs of the form $I(yzyz')$. Consequently, some nonrepetitive strings will be accepted.

In every typical accepting computation C on input $I(yzyz)$, the following "events" happen:

Event	Definition of Event
$\text{Exit}_C(yzyz)\ldots$	For the last time *during* the reading of the first y-portion of the input $I(yzyz)$, the stack height changes from $2q$ to $2q+1$
$\text{Entry}_C(yzyz)\ldots$	For the first time *after* the reading of the first y-portion of the input $I(yzyz)$, the stack height changes from $2q+1$ to $2q$.

By definition, the event $\text{Exit}_C(yzyz)$ will occur when reading a symbol in the first y-portion of the input $I(yzyz)$ [always presuming that C is a typical computation

on input $I(yzyz)$]. Since M accepts by empty store, $\text{Entry}_C(yzyz)$ must happen at some point during the computation C and, in fact, it must happen when reading a character in the second y-portion of $I(yzyz)$; it cannot happen during the first z-portion, because at the beginning of the first z-portion the stack height is more than $4q$ and the $2q$ symbols of the z-portion do not give M enough time to decrease the stack height by more than $2q$. Similarly, it cannot happen during the second z-portion because then M would not have time to empty the stack in order to accept the input. Figure 1 illustrates this.

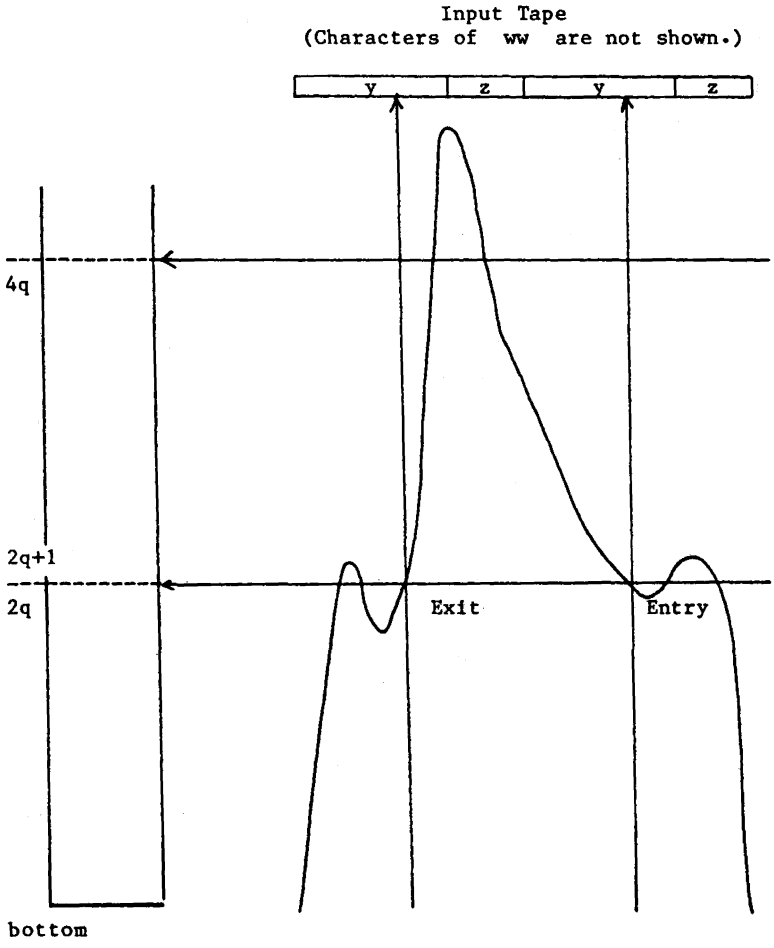

Figure 1. – The movements of the stack head plotted against the movement of the input head in a typical accepting computation.

For typical accepting computations C, let Exit-Time$_C$ $(yzyz)$ be the position of the input character being read in $I(yzyz)$ when Exit$_C$ $(yzyz)$ occurs in the computation C. Similarly, let Entry-Time$_C$ $(yzyz)$ be the position of the input character read when Entry$_C$ $(yzyz)$ occurs in the computation C. And finally let Exit-Symbol$_C$ $(yzyz)$ be the symbol put into stack location $2q+1$ when Exit$_C$ $(yzyz)$ occurs. From the remarks in the preceding paragraph we know that Exit-Time$_C$ $(yzyz)$, taken over all typical accepting computations and all inputs $I(yzyz)$, ranges over no more than $2p$ values, and that the same is true for Entry-Time$_C$ $(yzyz)$. The range of Exit-Symbol$_C$ $(yzyz)$ obviously is no more than the s symbols in the stack alphabet of M. This will be used in the proof of Lemma 4 later on.

LEMMA 3: *If*

$$\textit{Exit-Time}_C\ (yzyz) = \textit{Exit-Time}_{C'}\ (yz'\ yz'),$$

$$\textit{Exit-Symbol}_C\ (yzyz) = \textit{Exit-Symbol}_{C'}\ (yz'\ yz') \quad \textit{and}$$

$$\textit{Entry-Time}_C\ (yzyz) = \textit{Entry-Time}_{C'}\ (yz'\ yz'),$$

for some $y \in \{0, 1\}^p$, $z, z' \in \{0, 1\}^q$ *and typical accepting computations* C *on input* $I(yzyz)$ *and* C' *on input* $I(yz'\ yz')$, *then* M *accepts inputs* $I(yz'\ yz)$ *and* $I(yzyz')$.

Proof: The following is an accepting computation of M on input $I(yz'\ yz)$: M runs as in C until the event Exit$_C$ $(yzyz)$ occurs, then runs as in C' until Entry$_{C'}$ $(yz'\ yz')$ occurs, and then finishes as in C. Similarly for input $I(yzyz')$. □

LEMMA 4: *There is a string* $y \in \{0, 1\}^p$ *and there are strings* $z, z' \in \{0, 1\}^q$, $z \neq z'$, *and typical computations* C *on input* $I(yzyz)$ *and* C' *on input* $I(yz'\ yz')$ *which satisfy the conditions of Lemma* 3.

Proof: Choose y as shown to exist in Lemma 2. Then by ★'s there are more different strings z in $\{0, 1\}^q$ than there are different "Exit-Time", Exit-Symbol, Entry-Time" triples. □

Lemma 4 shows that M will indeed accept strings which are nonrepetitive. Therefore, there does not exist a *pda* accepting R, and, hence, R is not context-free. This finishes the proof of the Theorem for six-letter alphabets.

This result can be extended to alphabets with fewer than six characters by modifications to the proof. However, a more elegant way to show that the Theorem holds for alphabets with only three letters is provided by the following result from [2]:

There is an ε-free homomorphism h from a six-letter alphabet to a three-letter alphabet which preserves nonrepetitiveness, i. e. if w is nonrepetitive then so is $h(w)$.

For such a homomorphism h, the set of repetitive strings over a six-letter alphabet is the image, under h^{-1}, of the set of repetitive strings over a three-letter alphabet. Since context-free languages are closed under inverse homomorphism, and since we have shown that the set of repetitive strings over a six-letter alphabet is not context-free, it follows that the set of repetitive strings over a three-letter alphabet is not context-free either.

A GENERALIZATION

Our proof easily generalizes to all sets R_k, $k \geqq 2$, defined as:

$$R_k = \{ xy^k z \mid x, z \in \Sigma^* \text{ and } y \in \Sigma^+ \}.$$

SUMMARY, CONCLUSIONS, AND OPEN PROBLEMS

We have shown that the set of repetitive strings over a three - letter alphabet is not context-free. The proof consists of a careful analysis of the distribution, in the pushdown store of a *pda*, of information about the input to the *pda*.

Perhaps more important than the immediate result is the fact that the proof technique employed here seems, at least in this one instance, to be more powerful than known classical techniques. Whether or not it generalizes to a new and useful necessary condition for context-freeness remains to be seen. We expect to follow up on this issue in a subsequent paper.

ACKNOWLEDGMENTS

We wish to acknowledge the help of David Benson, who first brought to our attention the problem of showing that repetitive strings are not context-free, and the help of Richard Lorentz and Michael Main in discussing details and providing insights into the problem.

We are grateful to the referee who pointed out the result from [2] which provided an elegant way to extend our result to three-letter alphabets.

REFERENCES

1. J. M. AUTEBERT, J. BEAUQUIER, L. BOASSON and M. NIVAT, *Quelques problèmes ouverts en théorie des langages algébriques*, R.A.I.R.O. Informatique Théorique, Vol. 13, (4), 1979, pp. 363-378.

2. D. R. BEAN, A. EHRENFEUCHT and G. F. MCNULTY, *Avoidable Patterns in Strings of Symbols*, Pacific Journal of Mathematics, Vol. 85, (2), 1979, pp. 261-294.
3. J. BERSTEL, *Sur les mots sans carré définis par un morphisme*, A. MAURER, Ed., Automata, Languages, and Programming, Lecture Notes in Computer Science, Vol. 71, Springer-Verlag, 1979, pp. 16-25.
4. C. H. BRAUNHOLTZ, *Solution to Problem* 5030, American Math. Monthly, Vol. 70, 1963, pp. 675-676.
5. S. A. GREIBACH, *Eraseable Context-Free Languages*, Information and Control, Vol. 4, 1975, pp. 301-306.
6. M. HARRISON, *Introduction to Formal Language Theory*, Addison-Wesley, 1978.
7. G. A. HEDLUND, *Remarks on the Work of Axel Thue on Sequences*, Nordisk Matematisk Tidskrift, Vol. 15, 1967, pp. 148-150.
8. J. E. HOPCROFT and J. D. ULLMAN, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.
9. G. HOTZ and R. ROSS, *LL (k)-and LR (k)-Invarianz von Kontextfreien Grammatiken unter einer Transformation auf Greibach-Normalform*, Elektronische Informationsverarbeitung und Kybernetik, Vol. 15, (1/2), 1979, pp. 73-86.
10. P. A. B. PLEASANTS, *Nonrepetitive Sequences*, Proc. Cambridge Philosophical Society, Vol. 68, 1970, pp. 267-274.
11. R. ROSS, *Grammar Transformations Based on Regular Decompositions of Context-Free Derivations*, Dissertation, Computer Science Department, Washington State University, Pullman, WA. 1978.
12. R. ROSS, G. HOTZ and D. BENSON, *A General Greibach-Normal-Form transformation*, Technical Report CS-78-048, Computer Science Department, Washington State University, Pullman, WA. 1978.
13. A. THUE, *Über Unendliche Zeichenreihen*, Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl. (*Kristiania*), Nr. 7, 1906, S. 1-22.
14. A. THUE, *Über die Gegenseitige Lage Gleicher Teile Gewisser Zeichenreihen*, Norske Videnskabers Selskabs Skrifter Mat.-Nat. Kl. (*Kristiania*), Nr. 1, 1912, S. 1-67.