XIX. LINGUISTICS*

Academic and Research Staff

Prof. R. Jakobson
Prof. A. N. Chomsky
Prof. J. A. Fodor
Prof. M. Halle
Prof. J. J. Katz
Prof. R. P. V. Kiparsky
Prof. E. S. Klima
Prof. J. Kurlyowicz
Prof. G. H. Matthews
Prof. Krystyna Pomorska
Dr. S. Bromberger
Dr. M. F. Garrett
Dr. J. S. Gruber
Dr. S.-Y. Kuroda
Dr. A. Schwartz
Dr. D. E. Walker
G. B. Gragg
P. L. Peterson
J. J. Viertel

Graduate Students

G. R. Bedell IV
T. G. Bever
E. W. Browne
R. J. Carter
P. G. Chapin
Janet P. Dean
R. P. G. DeRijk
R. C. Dougherty
J. E. Emonds
J. L. Fidelholtz
M. L. Geis
R. Goldfield
J. W. Harris
T. R. Hofmann
R. S. Jackendoff
L. Jenkins
R. S. Kayne
J. P. Kimball
R. L. Mendelsohn
Amy E. Myers
A. J. Naro
D. M. Perlmutter
P. S. Peters, Jr.
C. B. Qualls
J. R. Ross
M. S. Snow
Carol A. Spielman
R. J. Stanley
Nancy H. Woo

A. A CHARACTERIZATION OF ESSENTIALLY CONTEXT-SENSITIVE LANGUAGES

The object of this report is to give a characterization of essentially context-sensitive languages (i.e., context-sensitive languages that are not context-free) and, for that matter, a characterization of context-free languages among context-sensitive languages.

We begin by introducing a restricted type of linear-bounded automata, linear-bounded automata with erasure:[1]

Definition. A lba M is said to be a lba with erasure if M contains in its alphabet a special symbol ϕ that does not affect computation in any way, i.e., if M scans a square with the symbol ϕ it will not replace it by another symbol, will not change its states, and will continue to move in the same direction as it has moved into the scanned square (namely, continue to move to the right (left) if it has moved to the right (left) or stay on the scanned square if it has stayed there). More exactly, the set of states of M is divided into three disjoint subsets, $\underline{S}_{+1}$, $\underline{S}_{-1}$, and $\underline{S}_0$ in such a way that M is in one of the states in $\underline{S}_{+1}$ ($\underline{S}_{-1}$, or $\underline{S}_0$) just after it has moved to the right (left or not moved), and further for every instruction of the form

$$(\phi, S) \rightarrow (a, T, D)$$

*This work was supported principally by the U.S. Air Force (Electronics Systems Division) under Contract AF 19(628)-2487; and in part by the Joint Services Electronics Programs (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DA 36-039-AMC-03200(E), the National Science Foundation (Grant GK-835), the National Institutes of Health (Grant 2 PO1 MH-04737-06), and the National Aeronautics and Space Administration (Grant NsG-496).

a is ϕ, T is S, and D is +1, -1, or 0 according as S is in $\underline{S}_{+1}$, $\underline{S}_{-1}$, or $\underline{S}_0$.

We define the notion of acceptance by a lba with erasure M as follows: a string x is accepted by M if there is a computation of M which accepts x in the usual sense of acceptance by a lba and if, moreover, at the end of the computation the tape contains only ϕ's. Acceptance by a lba with erasure in this sense may be referred to by acceptance with erasure. In spite of the apparent specialization of the notion of lba, it is easy to see that lba with erasure are as powerful as lba in the general sense; we state without proof the following proposition.

Proposition. If a language L is accepted by a lba, there is a lba with erasure that accepts L with erasure.

Now, let M be a lba with erasure and for each pair of states (S_i, S_j) of M let $[S_i, S_j]$ be the set of strings accepted by the lba with erasure $M_{i,j}$ which is the same as M except that its initial and final states are redefined as S_i and S_j, respectively.

Let x = yz be in $[S_i, S_j]$. We shall say that x is partitioned into y and z in $[S_i, S_j]$ if (i), given x as an input, there is a computation C of M with S_i and S_j as the initial and final states, respectively, which accepts x with erasure, and (ii) C is divided into two successive subcomputations C_1 and C_2 in such a way that, during C_1 and C_2, M scans, respectively, only the subtapes on which y and z are printed originally. In other words, at the beginning of the computation C, M is scanning the leftmost element of x (it is also the leftmost element of y) in the state S_i; at some moment M will scan the leftmost element of z for the first time, say, in the state S_k; this must be the end of the subcomputation C_1 and at the same time the beginning of the subcomputation C_2; at this moment y has all been replaced by ϕ's; after this M will never go to the left of the square that it is then scanning.

If x is partitioned into y and z in $[S_i, S_j]$, then y and z are elements of $[S_i, S_k]$ and $[S_k, S_j]$, respectively, for some S_k. Conversely, if x is in $[S_i, S_j]$, x = yz, and y and z are in $[S_i, S_k]$ and $[S_k, S_j]$ with some S_k, respectively, then x is partitioned into y and z in $[S_i, S_j]$.

Let x be again in $[S_i, S_j]$ and let x = vyw, where v, y, and w are non-null. We say that x is decomposed into an interior y and two boundaries v and w in $[S_i, S_j]$ if there is a computation C with the initial and final states S_i and S_j which accepts x with erasure and will be divided into three consecutive subcomputations C_1, C_2, and C_3 in the following way. At the beginning of the first subcomputation C_1 (i.e., actually at the beginning of the entire computation) M is scanning the leftmost element of v (i.e., the leftmost element of x). During the subcomputation C_1, M stays inside the subtape on which v is originally written. At the end of C_1, M will drop off the right edge of that subtape and scan the leftmost element of y in the state, say, S_k. This is at the same time the beginning of the subcomputation C_2. During C_2, M stays entirely inside the subtape on which y is originally written. At the end of C_2, this subtape contains only ϕ's

and M drops off the right edge of this subtape, scanning the leftmost element of w in the state, say, S_ℓ. Then the final subcomputation C_3 begins. During C_3, M may scan any square of the entire tape, but the subtape on which y is originally written does not affect the computation in any way, since it now contains only ϕ's. At the end of C_3 (i.e., at the end of the entire computation) the entire tape contains only ϕ's and M drops off the right edge of the entire tape in the state S_j.

In the sequel we are interested only in such cases for which the two boundaries v and w of a decomposition are elements (i.e., strings of length 1).

We have the following theorem.

Theorem. Let M be a lba with erasure. If for any pair of states (S_i, S_j) and for any string x of length greater than 1 in $[S_i, S_j]$, x can be either partitioned into two substrings or decomposed into an interior and two boundaries of length 1, then the language L(M) accepted by M is context-free.

Proof. For each pair of states (S_i, S_j), let us now consider $[S_i, S_j]$ as a nonterminal symbol for a grammar G, and let

$$[S_i, S_j] \rightarrow [S_i, S_k][S_k, S_j]$$

be a rule of G for any i, j, and k. Further, let

$$[S_i, S_j] \rightarrow a[S_k, S_\ell]b$$

be a rule if there exists x in the set $[S_i, S_j]$ which can be decomposed into an interior y and boundaries a and b in $[S_i, S_j]$, where y is in $[S_k, S_\ell]$. Further, if an element a is in $[S_i, S_j]$, let

$$[S_i, S_j] \rightarrow a$$

be a rule of G. Finally, let S be the initial symbol of G and for each final state S_f of M let

$$S \rightarrow [S_0, S_f]$$

be a rule of G, where S_0 is the initial state of M. Then, L(M) is just the language generated by G.

From the theorem we obtain a well-known result that was first due to Chomsky and Schützenberger:[2]

Corollary 1. If a language is accepted by a pushdown storage automaton, it is context-free.

Proof. A pushdown storage automaton can be simulated by a real-time pushdown storage automaton.[3] It is easy to see that a real-time pushdown storage automaton can be regarded as a lba with erasure with the property stated in the theorem.

Actually, Haines derived this result from the equivalence of pushdown storage

automata to real-time pushdown storage automata.[3] In his construction of a context-free grammar equivalent to a real-time pushdown storage automaton he refers to the instructions of the automaton rather than to its states as we did above in the proof of our theorem, and it would seem that the significance of the correspondence of the grammar and the automaton is not sufficiently clarified. Furthermore, he did not relate the notion of real-time pushdown storage automata to that of linear-bounded automata in an explicit way. Hence, although our theorem is a fairly straightforward generalization of his result, it may be of some interest, in particular, when we relate it to the fact that linear-bounded automata are the automata-theoretic counterpart of context-sensitive grammars, that is, that a language is context-sensitive if and only if there is a linear-bounded automaton that accepts it.[4] In fact, our theorem will give a certain characterization of essentially context-sensitive languages. Let us first put it in the following way.

Corollary 2. A context-sensitive language L is not context-free if and only if for any lba M with erasure that accepts it there exists a string in it which cannot be accepted by a completely localized computation of M.

The precise meaning of completely localized computation is as follows. Given a string x, consider a computation C starting at the left end of x with a state S_i and ending at the right end of x with a state S_j, and with the tape filled entirely with ϕ's. This computation is said to be localized if (i) x is partitioned into y and z in $[S_i, S_j]$, where y is in $[S_i, S_k]$ and z is in $[S_k, S_j]$ for some S_k, and the computation C consists of two successive subcomputations C_1 and C_2 which correspond to this partitioning of x into y and z, or (ii) x is decomposed into an interior y and two boundaries a and b and C consists of three successive subcomputations C_1, C_2, and C_3 which correspond to this decomposition of x into a, y, and b. A computation C of x is said to be completely localized if (i) x is of length 1, or (ii) if C is localized with respect to a partitioning of x into y and z and the corresponding subcomputations C_1 and C_2 are both completely localized, or (iii) C is localized with respect to a decomposition of x into a, y, and b, and the corresponding subcomputation of the interior y is completely localized. In brief, a computation is completely localized if x is processed by gradually partitioning and decomposing it until single elements are reached.

Let us reformulate Corollary 2 in terms of (general) lba. We define the notion of partition and decomposition with respect to a lba similarly to those notions with respect to a lba with erasure, except that in this case we do not require that after each subcomputation of a partition or after the subcomputation of the interior of a decomposition the corresponding subtape is filled with ϕ's, but we do require of a decomposition that whatever is left on the interior subtape after the second subcomputation of the decomposition does not affect the computation during the last subcomputation of the decomposition.[5] We

then define the notions of localized and completely localized computations with respect to a lba in the same way as with respect to a lba with erasure.

Now, given a lba M, we can construct a lba with erasure M' which is equivalent to M in such a way that if x is accepted by a localized computation of M it is accepted by a localized computation of M'. Indeed, M' can be constructed, in brief, by equipping M with a device that erases symbols (i.e., replaces them by ϕ) on the tape. Thus, we have the following corollary.

Corollary 3. A context-sensitive language L is not context-free if and only if for any lba M that accepts L there is a string in L which cannot be accepted by a completely localized computation of M.

L and M being as in Corollary 3, the set of all strings of L which cannot be accepted by a completely localized computation of M is obviously a subset of L which is responsible for the degree of complexity of L as a context-sensitive language and itself cannot be, for example, context-free.[6]

S.-Y. Kuroda

Footnotes and References

1. By linear-bounded automata (hereafter abbreviated as lba) we understand nondeterministic linear-bounded automata. For the definition of lba, see S.-Y. Kuroda, "Classes of Languages and Linear-bounded Automata," Information and Control 7, 207-223 (1964). Note, however, that boundary symbol # is dispensable in the definition of lba, contrary to the remark made in footnote 3 of this paper; see S. Ginsburg and G. F. Rose, "Preservation of Languages by Transducers" (to appear in Information and Control).

2. N. Chomsky, "Context-free Grammars and Pushdown storage," Quarterly Progress Report No. 65, Research Laboratory of Electronics, M.I.T., April 15, 1962, pp. 187-194; M. P. Schützenberger, "Context-free Languages and Pushdown Automata," Information and Control 6, 246-264 (1963).

3. L. Haines, "Generation and Recognition of Formal Languages," Ph.D. Thesis, M.I.T., 1965.

4. S.-Y. Kuroda, op. cit.

5. In this case the definitions of partition and decomposition are, in a sense, behavioristic. Thus, for example, we know that x is decomposed as ayb with respect to a computation only after having observed that the computation ended in the prescribed way, and not by means of a particular instantaneous description of computation that appears during the computation. Assume that we define $[S_i, S_j]$, analogously to the previous case of lba with erasure, as the set of strings that are accepted (this time in the general sense without erasure) with the initial and final states S_i and S_j, respectively. Then, on the one hand, it remains true that the product set $[S_i, S_k][S_k, S_j]$ is a subset of the set $[S_i, S_j]$. But, on the other hand, it does not hold that $a[S_k, S_\ell]b$ is a subset of $[S_i, S_j]$ if there exists a string x in $[S_i, S_j]$ which is decomposed as ayb with y in $[S_k, S_\ell]$. Thus, the proof of our theorem cannot be generalized directly to the general case without erasure.

6. Actually, the notions of lba and context-sensitive grammars are not essential restrictions in developing the ideas pursued in this report, and our theorem and corollaries can be generalized as statements in terms of Turing machines and unrestricted rewriting systems. On the one hand, this generalization is rather straightforward conceptually, but, on the other hand, its exact formulation would become greatly involved, since we must formulate a type of Turing machine that is allowed (not only to add at the edges but also) to insert a piece of tape between two squares. Thus we have restricted ourselves here to dealing with cases within the machinery that is readily available at present.