North East Linguistics Society

Volume 8 *NELS 8* Article 4

1978

# An Elementary Proof of the Peters-Ritchie Theorem

Emmon Bach
*University of Massachusetts/Amherst*

William Marsh
*Hampshire College*

# An Elementary Proof of the Peters-Ritchie Theorem

Emmon Bach
University of Massachusetts/Amherst

William Marsh
Hampshire College

0. Introduction. The mathematical results about various classes of transformational grammars continue to play a role in linguistic discussions. Peters and Ritchie (1973a) proved that transformational grammars of the "standard" sort with a context-sensitive base were equivalent to unrestricted rewriting systems (equivalently, Turing machines) in their weak generative capacity, that is, that there was such a grammar for every recursively enumerable language. The proof can be presented informally and is easy to grasp (see Bach, 1974, for an informal presentation of the proof).

A further and stronger result was proved in Peters and Ritchie (1971): Limiting the base to a context-free, finite state or even to a fixed base makes no difference. A transformational grammar still has the unrestricted power of the most powerful systems studied in recursive function theory (e.g. again Turing machines). This result is even more interesting for linguistic theory than the previous one. In the first place, if you think that the excessive power of transformational grammars is something to worry about, then this result pinpoints just where the trouble lies. It is in the transformational component of a standard grammar, since restricting the base does not decrease the power of the system. Second, the result is relevant to the idea that there is a universal set of base rules. This idea was advanced by a number of linguists in the sixties and is still assumed in some of the mathematical and empirical studies of learnability (Hamburger and Wexler, Culicover and Wexler). The result shows that it follows from the formal properties of the theory that there is such a base (in fact, infinitely many) and that the hypothesis thus has no empirical content within a formalization of the standard theory.

In contrast to the proof of the first theorem, the proof of the fixed-base theorem (as we may call it) is exceedingly complex and it is likely that many linguists simply have to take it on faith. The proof proceeds by constructing a complex transformation which, given an input containing all the elements of the basic vocabulary (plus some other junk), will imitate computations of a given

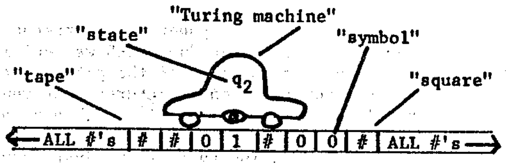
Turing machine. The purpose of this paper is to present an alternative proof for the same result, but one which we feel is somewhat easier to grasp. The central idea of the proof is to set up a base component which generates deep structures that represent all possible computations by a Turing machine with a given vocabulary and then use transformations which are constructed to match the instructions of a given machine to cull out just those final strings which would be computed by the machine.

The theorem in question states that for any alphabet A there is a regular base B such that for any recursively enumerable set R in A* there is a transformational grammar which weakly generates R from B. For concreteness we will fix A = $\{0,1,\#\}$. We hope that our proof is easily accessible to anyone who knows what a transformation and a Turing machine are. Since we expect that most of our readers are linguists, we will provide some background on Turing machines, though for reasons of clarity and economy of presentation we will present this more as a review than an introduction to the material.

In Section 1 we recall, or rather, avoid, some standard definitions concerning Turing machines and give an example which will be carried through the paper. In Section 2 we present a grammar for our fixed regular base B. In Section 3 we define the transformational grammar which mimics our example Turing machine and in Section 4 prove the Peters-Ritchie theorem. We close with a few comments in Section 5.

## 1. Turing Machines.

One of several provably equivalent definitions of the notion of algorithm was given by A.M. Turing in terms of very simple idealized computers, several years before the actual building of the first electronic computers. These "Turing machines" are myopic and methodical creatures which operate in a series of steps on a "tape" of "squares," each one of which contains one symbol from an alphabet or terminal vocabulary A; a single step is either a rewriting of a single symbol or a move to an adjacent square. Our machines can look at only one square at a time, and the step taken by the machine depends only on the symbol on that square and the "state" the machine is in; at the end of each step the machine moves into a (possibly) new state. Figure (1) illustrates a typical moment in the life of a Turing machine.



,

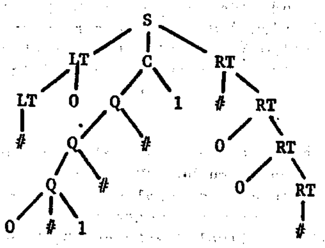
Figure (1)

Note that we include a # as a "blank" symbol, and picture the tape as being infinite in each direction, but all blanks after a certain point.

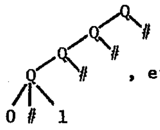
It is convenient to insert the state into the string being operated on just to the left of the symbol being considered and to describe the whole situation by the "configuration" (which we treat as a sentence):

# 0 $q_2$ 1 # 0 0 #.

We will modify this unspeakable sentence slightly and parse it



Note that $q_3$, for example, would be encoded by



, etc.

and that C includes the state the machine is in and the symbol being looked at, while LT and RT encode the left and right sides of the tape respectively.

To recapitulate, a Turing machine's action at a given moment depend only on the state it's in and the symbol on the one square it can see; actions are limited to exactly one of

(1) rewriting the symbol it sees,
(2) moving one square to the left,
(3) moving one square to the right,
and (4) stopping forever.

Thus we can describe a Turing machine as a set of quadruples, and we hereby introduce Charlie, our example Turing machine, and label each quadruple for future reference.

"Charlie" (a typical Turing machine)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (1) | $q_0$ | # | 0 | $q_1$ | (3) | $q_0$ | 1 | R | $q_0$ |
| (2) | $q_0$ | 0 | L | $q_0$ | (4) | $q_1$ | 0 | R | $q_1$ |
| | | | | | (5) | $q_1$ | 1 | # | $q_2$ |

The first symbol is the state the machine is in before the action, the second the symbol it is looking at, the third encodes the action taken, and the fourth, the state the machine goes into. We will illustrate by giving three typical biographies or "computations" performed by Charlie, which vary depending on the world (i.e., tape) that he is thrust into.

Our three example worlds are #1#, #10#, and #010#. A Turing machine always starts life in state $q_0$ on the left-most non-# on a tape.

The first biography goes

$$\#1\# \longrightarrow \#q_0 1\# \longrightarrow \#1q_0\#\# \longrightarrow \#1q_1 0\# \longrightarrow 10q_1\#\# \longrightarrow \#10\#$$

The second arrow shows an application of quadruple (3) (another # is added to the right end of the tape for convenience). The third arrow shows an application of (1) and the fourth, of (4). Since no quadruple begins with $q_1$#, Charlie stops, and the last string shows how he has left the world.

While the life just portrayed is quite conservative, our next biography is of a typical liberal: Charlie lives forever, moving back and forth from left to right, but never accomplishes anything

$$\#10\# \longrightarrow q_0 10\# \longrightarrow \#1q_0 0\# \longrightarrow \#q_0 10\# \longrightarrow \ldots.$$

Our third biography turns out as a tragedy: $\#010\# \longrightarrow \#q_0 010\# \longrightarrow$

$$\#q_0\#010\# \longrightarrow \#q_1 0010\# \longrightarrow \#0q_1 010\# \longrightarrow \#00q_1 10\# \longrightarrow \#00q_2\#0\# \longrightarrow \#00\#0\#$$

The sadness lies in the fact that although Charlie stops, we have a convention that if a Turing machine leaves an "internal" blank on the tape, the string is "filtered."

We thus say that Charlie (1) = 10, read "Charlie of 1 is 0," but that Charlie (0) and Charlie (010) are undefined. We hope that the notions illustrated above are clear; we refer readers wanting formal definitions to Davis (1968).

Finally a set R of strings over A is called "recursively enumerable" if there is a Turing machine M such that R is the set of (unfiltered) outputs of computations by M beginning on strings of A* which aren't totally blank.

2. In this section we give a grammar for our fixed base language B, which the reader can check is the regular set

0#1 (#A*0#1(#)*A*#)*

where the Kleene X* indicates the set of all strings - even the empty one - formed from the strings or symbols in X. The base structures encode all possible sequences of Turing machine configurations, among which are included all computations done by any Turing machine on any legal starting tape. The transformational grammar for a given machine M among other actions selects out from B only the computations M performs, making sure everything else in B ends up getting filtered. Our grammar, which uses the non-terminals S, LT, C. RT, Q, and BEG, contains the following sixteen rules:

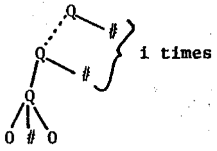
$$\begin{array}{lll} S \longrightarrow S\ \ LT\ C\ RT & LT \longrightarrow \# & Q \longrightarrow Q\ \# \\ S \longrightarrow BEG\ LT\ C\ RT & RT \longrightarrow \# & Q \longrightarrow 0\ \#\ 1 \\ BEG \longrightarrow 0\ \#\ 1 & & \end{array}$$

$$\left.\begin{array}{l} LT \longrightarrow LT\ a \\ RT \longrightarrow a\ RT \\ C \longrightarrow Q\ a \end{array}\right\} \left\{\begin{array}{l} \text{for all} \\ a \in V_T \\ \end{array}\right.$$

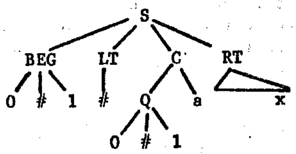
3. The transformational grammar which mimics a given Turing machine will have transformations of four classes: Beginning, Rewriting, Movement, and Ending, which appear in this order in the cycle. Beginning and Ending transformations apply only to sentences containing no embedded sentences, while the others apply only to sentences containing exactly one embedded sentence.

Proposition 1. It follows that if any cycle ends with an embedded sentence, no transformations will apply at higher cycles.

Before defining our transformations let us agree to use $Q_i$ to denote

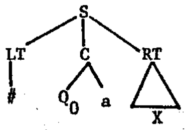


Our two Beginning Transformations will delete a BEG from deep structures that encode computations that start in $q_0$ at the left-most non-blank symbol:

| | CONDITION | ACTION |
|---|---|---|
| $T_1$ with a = 0<br>$T_2$ with a = 1 |  | Delete the BEG. |

Since none of the other transformations to be introduced will apply to any sentence containing a BEG, it will follow that

Proposition 2. Deep structures whose deepest S do not encode proper Turing machine beginnings are filtered and in the others, the deepest S is left in the form

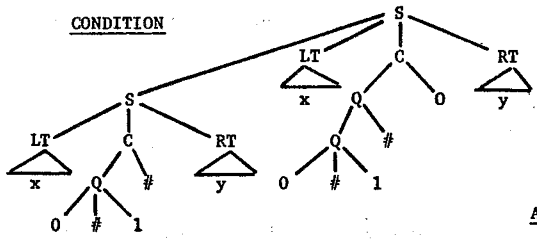


Note also that since there is exactly one BEG per deep structure, it follows that

Proposition 3. There is at most a single application of a Beginning Transformation to a deep structure.

There will be one Rewriting Transformation for each rewriting quadruple in the Turing machine; for concreteness we will give that for $q_0\#0q_1$.

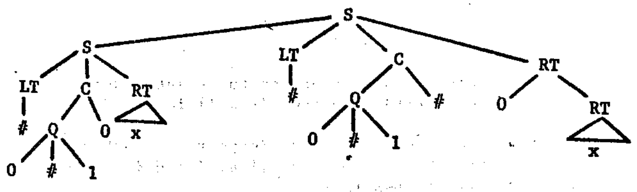
CONDITION



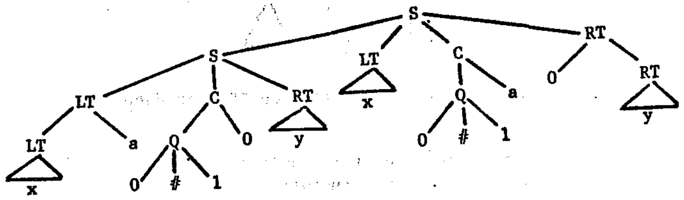
ACTION

Delete the embedded S.

For each movement quadruple our grammar will contain four Movement Transformations, one for the case when you have to add a # to the end of the tape and three for cases when you don't. We illustrate the conditions of these transformations for the quadruple $q_0 0Lq_0$; the actions are always deleting the embedded S.



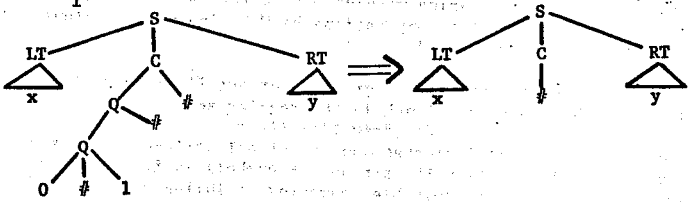
And for each $a \in A$,



We wish to note the obvious

Proposition 4. (i) Rewriting and Movement Transformations apply only to sentences with exactly one embedded S; (ii) At most one of them can apply to any such sentence; (iii) If one does apply, it encodes exactly the correct action of the Turing machine being mimicked and leaves a sentence which encodes the result of that action (and contains, therefore, no embedded S).

Finally, there will be one <u>Ending Transformation</u> for each pair $q_i a$ where $a \in A$ and $Q_i$ occurs in a quadruple in our machine but the machine contains no quadruple beginning with $q_i a$. For Charlie, $q_1\#$ is a case in point and we put in the transformation



Note that the top S in a deep structure will have all its Q's removed only if an ending transformation applies at the end of its cycle, so we have

Proposition 5. A deep structure is unfiltered only if in its last cycle an ending transformation applies.

We summarize the construction for the general case as follows: Given M construct the set of transformations G as follows (in each case the action is to delete the underlined part).

I. For each $a \in V_T - \#$:

$$\underline{BEG} \quad \#Q_0 aRT$$

II. For each quadruple $q_i abq_j$ in M:

$$\underline{LTQ_i aRT} \; LTQ_j bRT$$

III. For each quadruple $q_i aLq_j$ in M:

for each $b \in V_T$ (including #!)

$$\underline{[Xb]_{LT} \; Q_i \; RT} \quad [X]_{LT} \; Q_j \; b \; a \; RT$$

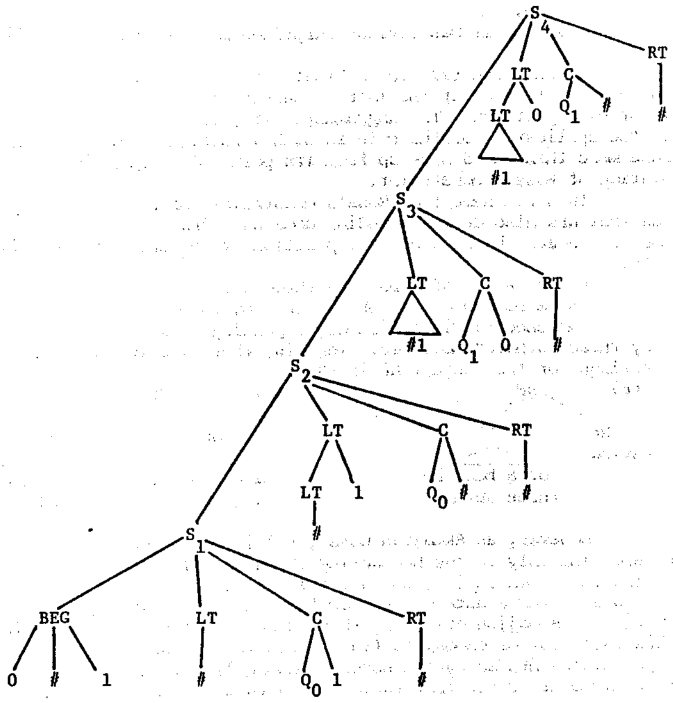
IV. For each quadruple $q_i aRq_j$ in M:

for each $b \in V_T$

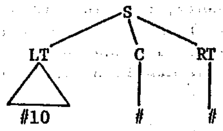
$\underline{[X]\ Q_i\ a\ [bY]}$ $[Xa]\ Q_j\ b\ [Y]$
$\underline{LT \qquad RT}$ $LT \qquad RT$

V. For each quadruple $q_i a$ such that $q_i \in$ the set of states for M, $a \in V_T$, but there is no quadruple beginning $q_i a$ in M:
LT $\underline{Q_i}$ a RT

4. Consider the deep structure



where we subscript S's for easy reference. Observe that $T_2$ and no other transformation applies to $S_1$; that the movement transformation that goes with $q_0 1Rq_0$ applies at $S_2$, deleting what was left of $S_1$; that the rewrite transformation we illustrated applies to $S_3$; and that a movement and then an ending transformation applies to $S_4$, leaving



The extra "external" # doesn't hurt anything and the sentence is the output 10 = Charlie (1). Thus the tranformational grammar exactly mimics the first example computation in Section 1.

To "see" that the G constructed for a machine M correctly models M, consider an arbitrary computation C by M on a string $\underline{x}$:

$$\#\underline{x}\# = \alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \cdots\cdots \longrightarrow \alpha_n = \underline{y}.$$

Clearly there is a deep structure D which encodes C just as that beginning Section 4 encoded our first example computation. Just as clearly, the appropriate transformations will apply one at a time to leave just y with possible extra #'s at each end of the cycle on the top-most S. On the other hand, by Propositions 2 and 5, only deep structures that begin and end correctly will be unfiltered by G; by Proposition 4 only Rewrite and Movement transformations that correctly encode M ever apply; and by Proposition 1 if on any cycle no transformation applies the resulting sentence is filtered.

5. We close with a few comments on properties of the grammars that result from our construction and on the relation between the grammars and various constraints that have been suggested in the literature.

The grammar obeys the property of subjacency which has figured both in Chomsky's discussions of conditions on rules (e.g., 1973) and in the work of Wexler, Culicover, and Hamburger on learnability. Indeed, using a grammar which does obey this property simplifies the construction considerably. The result shows that subjacency has no effect on the weak generative capacity of the class of transformational grammars.

Peters and Ritchie (1973b) have investigated a class of grammars that they call "local-filtering" transformational grammars. Such grammars meet the restriction that all internal blocking symbols (##) must be removed from a structure at the end of the cycle on that structure. They show that this condition has an effect on the generative power of the system since the grammars will not now represent every recursively enumerable language (although they will represent some non-recursive languages). Our grammars are not local-filtering, obviously, but since they obey subjacency are "local + 1" filtering. This shows that relaxing local filtering to filtering with some fixed bound on the depth of the filtering give you all the power of non-local-filtering grammars.

## References

Bach, Emmon (1974). Syntactic theory. New York.

Chomsky, Noam (1973). Conditions on transformations. In Stephen R. Anderson and Paul Kiparsky, eds., A festschrift for Morris Halle.

Culicover, Peter and Kenneth Wexler (1977). Some syntactic implications of a theory of language learnability. In P.W. Culicover, T. Wasow, and A. Akmajian, eds., Formal syntax. New York.

Davis, Martin (1968). Computability and unsolvability. New York.

Hamburger, Henry and Kenneth N. Wexler (1973). Identifiability of a class of transformational grammars. In K.J.J. Hintikka, J.M.E. Moravcsik, and P. Suppes, eds., Approaches to natural language. Dordrecht.

Peters, P.S., Jr. and R.W. Ritchie (1971). On restricting the base component of transformational grammars. Information and control, 18: 483-501.

Peters, Stanley and R.W. Ritchie (1973a). On the generative power of transformational grammars. Information sciences, 6: 49-83.

Peters, P. Stanley, Jr., and R.W. Ritchie (1973b). Nonfiltering and local-filtering transformational grammars. In K.J. Hintikka, J.M.E. Moravcsik, and P. Suppes, eds., Approaches to natural language. Dordrecht.