13 POST'S NORMAL-FORM THEOREM

13.0 INTRODUCTION

The theorem proved in this chapter is the normal-form theorem of Post's [1943] paper. I feel that it is one of the most beautiful theorems in mathematics: Any formal system can be reduced to a Post canonical system with a single axiom and only productions of the simple form

$$g\$ \rightarrow \$h \qquad \text{("Normal" production)}$$

Post's proof of the theorem is quite lengthy. Our proof seems to us considerably simpler, because it is less concerned with the order in which things happen. Some clarity is gained because of fewer auxiliary symbols; some is perhaps lost because of the simultaneous operation of many rules. To make up for this, we illustrate the proof with a detailed example that should help the reader see intuitively why the theorem is true.

We will state and prove the theorem in a series of forms of increasing strength, then give some new results using the same general methods used in the proofs. In the subsequent chapter we will examine the relation between these and the results of earlier chapters.

13.1 THE NORMAL-FORM THEOREM FOR SINGLE-ANTECEDENT PRODUCTIONS

THEOREM 13.1

Given a Post canonical system P with alphabet A and productions of the form

$$g_0\$_1 g_1 \$_2 \ldots \$_n g_n \rightarrow h_0 \$_1' h_1 \$_2' \ldots \$_m' h_m \qquad (\pi)$$

we can construct a new "normal" canonical system P whose productions all have the simple form*

$$g\$ \rightarrow \$h$$

and which is a canonical extension of P over A. That is, those theorems of P^* which contain only letters from the original alphabet A of P will be precisely all the theorems of P.

Recalling the meaning of the production (π) we can foresee the following difficulties. We will have to overcome the apparent limitation of normal productions to "look" only at the initial letters of a theorem. They can tell when a string begins with g_0, but how can they check whether a string contains, for example, a copy of g_2 preceded by a copy of g_1? More precisely, in spite of this limitation, we have to (1) determine when a string has the antecedent form of (π)—that is, when a string contains non-overlapping occurrences of $g_0, g_1, \ldots, g_n$ in that order, and (2) separate out the \$ strings that come between the discovered g_i's and insert copies of them into the proper positions of the consequent form—that is, between copies of the constant strings $h_0, h_1, \ldots, h_m$. Furthermore, we must (3) provide that if the antecedent form is satisfied in several different ways, then all corresponding consequent forms for these are generated.

The construction turns around a technique of "rotating" strings so that each part of the string eventually comes around to the front.

In all constructions we will use upper-case letters for the auxiliary symbols introduced for the new system P^*. We will begin by supposing that the letter T (for "Theorem of P") is available and that theorems of P are represented in P^* by their form in P preceded by the single letter T. We proceed first by illustrating the construction of P^* for a particular example.

13.1.1 Antecedent form recognition

Suppose (for example) that our production is actually

$$Tab\$_1 cb\$_2 b\$_3 \rightarrow Taa\$_3 bb\$_1 aa\$_3 c \qquad (\pi\text{-example})$$

so that we have

$g_0 = Tab$	$h_0 = Taa$
$g_1 = cb$	$h_1 = bb$
$g_2 = b$	$h_2 = aa$
$g_3 = \text{null}$	$h_3 = c$

and we want to apply this to the string S

$$Tabefcbgbdc \qquad (S)$$

(which does fit the antecedent form, with $\$_1 = ef$, $\$_2 = g$, and $\$_3 = dc$). The result should be

$$T\underline{aa}dc\underline{bb}ef\underline{aa}dc\underline{c}.$$

We begin by providing normal productions to check whether a string has this antecedent form. We will use the following system:

Alphabet: letters of A, and $T, T_1, T_2, \ldots, T_n$

Productions:

$$Tab\$ \rightarrow \$T_1$$
$$T_1cb\$ \rightarrow \$T_2$$
$$T_2b\$ \rightarrow \$T_3$$
$$x\$ \rightarrow \$x \qquad \text{(for all } x \text{ in } A\text{)}$$
$$T_ix\$ \rightarrow \$T_i \qquad \text{(for all } x \text{ in } A\text{, and all } i = 1, \ldots, n\text{)}$$

This system of productions has the recognition property: *If a string has the form* $Tab\ \$_1cb\$_2b\$_3$, *then, and only then, this system will produce the string* T_3. To see why this is so, let us apply these productions to the string. The following strings are produced in order:

$$\boxed{T\ a\ b}\ e\ f\ c\ b\ g\ b\ d\ c \rightarrow$$
$$e\ \boxed{f}\ c\ b\ g\ b\ d\ c\ T_1 \rightarrow$$
$$e\ f\ \boxed{c}\ b\ g\ b\ d\ c\ T_1\ e \rightarrow$$
$$e\ f\ c\ \boxed{b}\ g\ b\ d\ c\ T_1\ e\ f \rightarrow$$
$$e\ f\ c\ b\ \boxed{g}\ b\ d\ c\ T_1\ e\ f\ c \rightarrow$$
$$e\ f\ c\ b\ g\ \boxed{b}\ d\ c\ T_1\ e\ f\ c\ b \rightarrow$$
$$e\ f\ c\ b\ g\ b\ \boxed{d}\ c\ T_1\ e\ f\ c\ b\ g \rightarrow$$
$$e\ f\ c\ b\ g\ b\ d\ \boxed{c}\ T_1\ e\ f\ c\ b\ g\ b \rightarrow$$
$$e\ f\ c\ b\ g\ b\ d\ c\ \boxed{T_1}\ e\ f\ c\ b\ g\ b\ d \rightarrow$$
$$e\ f\ c\ b\ g\ b\ d\ c\ \boxed{T_1\ e}\ f\ c\ b\ g\ b\ d\ c$$

so that the T symbol has rotated around to the front of the string. Now (only) the production $T_1e\$ \rightarrow \$T_1$ applies, yielding

$$\boxed{T_1\ e}\ f\ c\ b\ g\ b\ d\ c \rightarrow$$
$$f\ \boxed{c}\ b\ g\ b\ d\ c\ T_1 \rightarrow$$
$$f\ c\ \boxed{b}\ g\ b\ d\ c\ T_1\ f \rightarrow$$
$$\ldots \quad \boxed{T_1\ f}\ c\ b\ g\ b\ d\ c \rightarrow$$
$$c\ \boxed{b}\ g\ b\ d\ c\ T_1 \rightarrow$$
$$\ldots \quad \boxed{c}\ T_1\ c\ b\ g\ b\ d \rightarrow$$
$$T_1\ c\ b\ g\ b\ d\ c$$

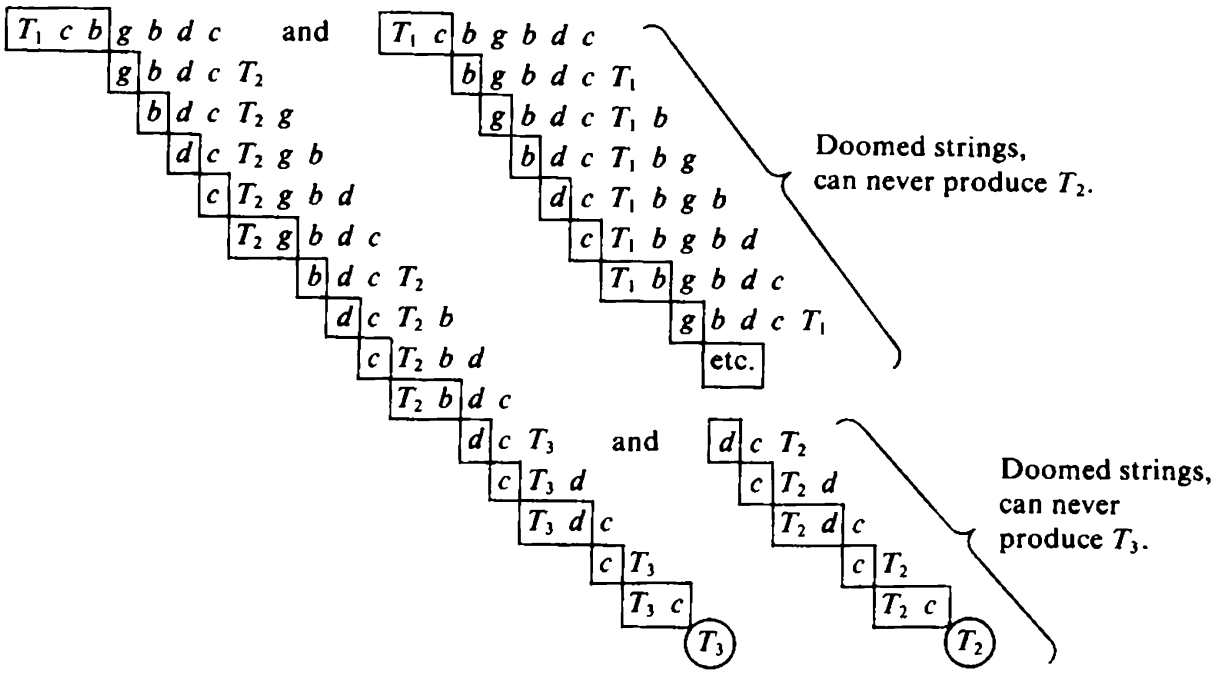
At this point, for the first time, two different productions can apply, generating two families of strings: that is, both

$$T_1 cb\$ \rightarrow \$T_2$$

and

$$T_1 c\$ \rightarrow \$T_1$$

apply, yielding



So the system produced the string T_3. Observe that, to produce T_3 the system *must* produce T_1 and T_2 along the way, in order, and that each advance—in the index of the T symbol—requires that the T symbol encounter g_i, the required substring of the antecedent form. Thus, for instance, the string to the upper right, $bgbdcT_1$ is "doomed" in that its T_1 will never encounter the substring cb it needs to become upgraded to T_2. Clearly, then, if the string has the antecedent form, there is a route which produces T_3, and if the string has not the required form, production of T_3 is impossible. The example thus illustrates a perfectly general method:

The system of productions:

$$\begin{cases} x\$ \rightarrow \$x & (\text{all } x \text{ in } A) \\ T_j x\$ \rightarrow \$T_j & (\text{all } x \text{ in } A, \text{ all } j) \\ T_j g_j\$ \rightarrow \$T_{j+1} & (\text{all } j) \end{cases}$$

produces the string T_{n+1} from a string S if and only if the string S contains the non-overlapping substrings $g_0, g_1, \ldots, g_n$ in that order.

We have now solved most of our problem: we know how to get around

the apparent limitations of the normal-form production (namely, by the rotation trick), and we know how to recognize that a string has a given antecedent form. We still have to show how to generate the appropriate consequent form.

REMARK

If the input string has the antecedent form in different ways, then our system will produce T_3 by different routes. For example

$$Tabcbcbba$$

will have three interpretations:

$$\underline{Tab}cb\underline{cb}\underline{ba}, \quad \underline{Tab}\underline{cb}cb\underline{ba}, \quad \text{and} \quad \underline{Tab}\underline{cb}\underline{cb}ba$$
$$\$_1 \quad \$_3 \qquad \$_2 \quad \$_3 \qquad \$_2 \quad \$_3$$

These should eventually result in three different consequent strings.

The only trouble with the present system is that, while it recognizes antecedents, it destroys the information about what the \$'s are, and this information is needed to construct the consequent. To save this information, we will use the more complicated system of productions below. We introduce an array of new upper-case letters:

$$A^j_x \qquad \begin{matrix}(j = 1, \ldots, n)\\ (\text{all } x \text{ in } A)\end{matrix}$$

and a new (and final) system of productions:

$$\left.\begin{array}{rl} x\$ & \to \$x \\ A^j_x\$ & \to \$A^j_x \\ T_j x\$ & \to \$A^j_x T_j \\ T_j g_j\$ & \to \$T_{j+1} \\ T_n A^j_x\$ & \to \$QA^j_x \end{array}\right\} \begin{matrix}(\text{all } x \text{ in } A)\\ (j = 1, \ldots, n)\end{matrix} \qquad (P^*)$$

This system is like the previous system except that it preserves the \$ information instead of deleting it. For example, on the string

$$\underline{Tab}\underline{bc}\underline{ba}\underline{ab}$$

we obtain, following only the main path and ignoring doomed strings,

$$\boxed{T\ a\ b}\ b\ c\ b\ a\ a\ b$$
$$b\ c\ b\ a\ a\ b\ T_1$$
$$\cdots$$
$$\boxed{T_1\ b}\ c\ b\ a\ a\ b$$
$$c\ b\ a\ a\ b\ A_b^1\ T_1$$
$$\cdots$$
$$A_b^1\ T_1\ c\ b\ a\ a\ b$$
$$\boxed{T_1\ c\ b}\ a\ a\ b\ A_b^1$$
$$a\ a\ b\ A_b^1\ T_2$$
$$\cdots$$
$$\boxed{T_2\ a}\ a\ b\ A_b^1$$
$$a\ b\ A_b^1\ A_a^2\ T_2$$
$$\cdots$$
$$b\ A_b^1\ A_a^2\ A_a^2\ T_2$$
$$\cdots$$
$$\boxed{T_2\ b}\ A_b^1\ A_a^2\ A_a^2$$
$$A_b^1\ A_a^2\ A_a^2\ T_3$$
$$\cdots$$
$$\boxed{T_3\ A_b^1}\ A_a^2\ A_a^2$$
$$A_a^2\ A_a^2\ Q\ A_b^1$$
$$\cdots$$
$$Q\ A_b^1\ A_a^2\ A_a^2$$

asserting that $\$_1 = b$ and $\$_2 = aa$. Similarly the string

$$\underline{Tabefcbgbdc}$$

yields

$$QA_e^1A_f^1A_g^2A_d^3A_c^3 \qquad (\underline{S'})$$

while the string

$$\underline{Tabcbcbba}$$

yields

$$QA_c^1A_b^1A_a^3,\quad QA_c^2A_b^2A_a^3,\quad \text{and}\quad QA_c^2A_b^3A_a^3$$

13.1.2 Consequent form construction

To construct the consequent, we have to make copies of the $ strings into the proper positions in the consequent form. That form itself will be

set up by the production

$$\boxed{Q\$ \rightarrow \$h_0 V_{i_1} h_1 V_{i_2} \ldots V_{i_m} h_m Z Y} \qquad (P^*\text{—continued})$$

in which the h's are the constant strings of the consequent and the V's are new letters indicating where the corresponding \$'s are to go—that is, a copy of $\$_i$ is to be inserted at each occurrence of its V_i. The new letters Z and Y will be used later. When this production is added to the system of the last section and applied to the string *Tabefcbgbdc* we obtain (using the h's of our original example)

$$A_e^1 A_f^1 A_g^2 A_d^3 A_c^3 aa V_3 bb V_1 aa V_3 c Z Y \qquad (S'')$$

Our trick will be to cause the A's to "trickle" across the string, without changing their order. Whenever an A_x^i passes a V_i with the same index (i), it will leave behind a copy of its subscript letter x. Thus when all the A's have passed all the V's there will be a copy of $\$_i$ next to each V_i!

The productions to do the trickling are:

$$\boxed{\begin{array}{l} A_x^i y\$ \rightarrow \$ y A_x^i \qquad (\text{all } x \text{ in } A, y \text{ in } A) \\ \qquad\qquad\qquad\qquad\quad (\text{all } i = 1, \ldots, n) \end{array}}$$

which lets the A's pass over lower-case letters,

$$\boxed{A_x^i V_j\$ \rightarrow \$ V_j A_x^i \qquad (\text{if } i \neq j)}$$

so that the A's can skip over non-matching V's, and $(P^*\text{—continued})$

$$\boxed{A_x^i V_i\$ \rightarrow \$ V_i x A_x^i \qquad (\text{all } x \text{ in } A, i = 1, \ldots, n)}$$

which leaves a copy of x to the right of the V_i. We also will need

$$\boxed{\begin{array}{l} Y\$ \rightarrow \$Y \\ Z\$ \rightarrow \$Z \end{array}}$$

to allow strings to rotate around.

Applying these productions to our example string S'' yields a great many routes and paths of generated strings, but there is only one final result. Typical strings in the process, as the A's migrate to the right, are:

$$Y A_e^1 A_f^1 A_g^2 A_d^3 A_c^3 aa V_3 bb V_1 aa V_3 c Z$$
$$Y A_e^1 A_f^1 A_g^2 a A_d^3 a V_3 c A_c^3 bb V_1 aa V_3 c Z$$
$$Y A_e^1 A_f^1 aa V_3 dcb A_g^2 b V_1 aa V_3 dcc A_d^3 A_c^3 Z$$
$$Y aa V_3 dcbb A_e^1 V_1 faa A_f^1 A_g^2 V_3 dcc A_d^3 A_c^3 Z$$

and eventually,

$$Yaa V_3 \boxed{dc} bb V_1 \boxed{ef} aa V_3 \boxed{dc} cA_e^1 A_f^1 A_g^2 A_d^3 A_c^3 Z$$

where we have marked where the \$ strings have been copied in. Now our work is done, except for removing the scaffolding—the A's and V's and Y and Z used in the construction. We use a trick based on the following observations: We can eliminate an A when it reaches Z, for then its work is done. We can eliminate a V once all the A's have passed it, for then *its* work is done. We will use the Y to inform the V's that all the A's have passed through. We will not permit Y to pass an A, but once all the A's have passed a V, Y can come up and eliminate the V. Thus the productions

$Yx\$ \rightarrow \$xY$	(x in A)	
$YV_i\$ \rightarrow \$Y$	(all $i = 1, \ldots, n$)	(P^*-continued)
$A_x^i Z\$ \rightarrow \$Z$	x in A, $i + 1, \ldots, n$	

will destroy the A's on contact with Z and the V's on contact with Y. Only when all the A's and V's are gone can Y come to stand just to the left of Z, and we celebrate this completion of the whole process with the final production of our system:

$$\boxed{YZ\$ \rightarrow \$T} \qquad (P^*\text{-completed})$$

Applying all this to our example, string S results finally in

$$Taadcbbefaadcc$$

PROBLEM 13.1-1. Reconstruct this proof using productions $g\$ \rightarrow \$h$ in which neither g nor h have more than two letters.

PROBLEM 13.1-2. Show that only two auxiliary letters are needed in proving theorem 13.1-1. In fact, only one is needed(!) but this is very much harder to prove and requires a different method for proving the theorem.

13.1.3 Completing the proof

There remain a few loose ends in the proof. First, note that we haven't really produced a legitimate extension P^* of P because P^*, so far, does not produce any strings with only lower-case letters. (Proof of this: The axioms of P contain the upper-case letter T. Every production with an upper-case letter in its antecedent also has one in its consequent. So, by induction, every produced string has an upper-case letter.) Now the strings we want to "detach" are those that begin with T, because any

string $T\$$ is an assertion that $\$$ is a theorem of the system P. It would be tempting to introduce the production $T\$ \rightarrow \$$ to simply remove an initial T, but this won't work. The reason is that this would lead to spurious lower-case theorems, namely, rotated versions of legitimate theorems, because the present system already contains the productions $x\$ \rightarrow \$x$. In fact, we have to conclude that there is *no* way, in the present system, to avoid this. The cure: begin all over again with a new alphabet, supposing that $B = b_1, b_2, \ldots, b_r$ is the real alphabet of P and that $A = a_1, a_2, \ldots, a_r$ were really new letters of P^*. Now we can detach the theorems of P by the productions

$$\boxed{\begin{array}{c} Ta_j\$ \rightarrow \$Rb_j \\ a_j\$ \rightarrow \$b_j \\ R\$ \rightarrow \$ \end{array}}$$

which do not allow the b's to rotate. Of course, in this system, a's may be converted to b's prematurely, but this leads only to "doomed strings" and not to spurious pure b strings.

PROBLEM. Prove this.

What if the original system P has more than one production? We simply carry out the whole construction again for each P-production, using entirely new sets of auxiliary letters. Only the key symbol T is common. Then the P-productions operate independently, linked only by the common T that allows any P-production to operate on the *final* results of the operation of other P-productions.

We need one more theorem to complete the proof that the canonical systems of Post, in their full generality, can be replaced by normal extensions. We have to account for those more powerful productions whose antecedent concerns several strings rather than just one. For example, in logic one often has rules of inference like: "From a theorem R and another theorem of the form R *implies* Q we can deduce Q." This says, in the language of productions, that something like

$$R \text{ AND } (R \Rightarrow Q) \rightarrow Q$$

should be a production of the system, where AND is not a string but a way of saying that the production has two separate antecedents.† In the next section we will prove a more general theorem about such systems. The

†There is a serious difficulty in representing logical systems directly as systems of productions, unless one introduces auxiliary symbols for making sure that parenthesis sets are not erroneously broken.[1]

method of proof there is of interest in itself, as we shall see later, because it shows another connection between logistic systems and machines.

13.2 THE NORMAL FORM THEOREM FOR MULTIPLE-ANTECEDENT PRODUCTIONS. REDUCTION TO SINGLE-AXIOM SYSTEM

We now consider a more general form of a Post system, in which a production can combine several strings (theorems) to form a new theorem. In this most general system a typical production has the form

$$\left.\begin{array}{ll} g_{10}\$_{11}g_{11}\$_{12}\cdots\$_{1n_1}g_{1n_1} & \textbf{and} \\ g_{20}\$_{21}g_{21}\$_{22}\cdots\$_{2n_2}g_{2n_2} & \textbf{and} \\ \cdots\cdots\cdots\cdots\cdots\cdots & \\ g_{p0}\$_{p1}g_{p1}\$_{p2}\cdots\$_{pn_p}g_{pn_p} & \end{array}\right\} \rightarrow h_0\$'_1h_1\$'_2\cdots\$'_mh_m \qquad (\pi)$$

where all the g's and h's are given constant strings and all the \$'s are variables and where each $\$'_k$ is some one of the $\$_{ij}$'s. The meaning is that, if there are already p theorems that satisfy the p antecedent conditions, then a new theorem is produced according to the consequent form, which may use parts from all the antecedent strings. Post argued that any operation performed by a mathematician could be accounted for by some such production. We want to show that any system of this form can be replaced by a normal canonical extension. We will show, incidentally, that this can be done in such a way that only one axiom is required. As you will observe, however, this refinement is pretty much forced upon us.

THEOREM 13.2-1

Given a Post canonical system P of the general multi-antecedent kind, we can construct a new canonical system P that has only single-antecedent productions, has a single axiom, and is a canonical extension of P.* (Then by the Theorem of 13.1 we can further construct a *normal* extension P^{**} of P, still with *a single axiom.*)

Let A be the alphabet of P (assumed as before to be lower-case) and let $\Phi_1, \ldots, \Phi_r$ be the axioms of P. The one axiom of the new system P^* will be

$$BB\Phi_1BB\Phi_2BB\ldots BB\Phi_rBB$$

There would be little point in having several axioms now, anyway, since with single-antecedent productions there could be no possibility of their interacting to produce theorems. Corresponding to each production of

the form π belonging to the original system P, we will introduce the following monstrously complicated production:

$$B\$_0^*B\ g_{10}\$_{11}\ldots\$_{1n_1}g_{1n_1}\ B\$_1^*B\ \ldots\quad\ldots B\$_{p-1}^*Bg_{p0}\$_{p1}\ldots\$_{pn_p}g_{pn_p}B\$_p^*B$$

$$\longrightarrow$$

$$\boxed{B\$_0^*B\ldots\text{(exactly the same as antecedent)}\ldots B\$_p^*B}\ h_0\$_1'h_1\$_2'h_2\ldots$$

$$\ldots\$_m'h_mX\$_{11}\$_{12}\ldots\$_{1n_1}\$_{21}\$_{22}\ldots\$_{2n_2}\ldots\quad\ldots\$_{p1}\$_{p2}\ldots\$_{pn_p}X$$

What does this do? The antecedent attempts to analyze a string to see if it contains, sandwiched between B's, substrings that fit the antecedent forms

$$g_{10}\$_{11}\ldots\$_{1n_1}g_{1n_1}$$

$$\ldots$$

$$g_{p0}\$_{p1}\ldots\$_{pn_p}g_{pn_p},$$

that is, to see if the proper ingredients for the production are to be found somewhere among a string of theorems of the form

$$BB\$BB\$BB\ldots BB\$BB\$BB$$

If this happens, the proper consequent will be assembled, eventually, and adjoined to the string:

$$BB\$BB\$BB\ldots BB\$BB\$BB\$_CBB$$

so that in the future the new theorem $\$_C$ will have the same status as an axiom; that, after all, is what a theorem is.

That is the plan, anyway. Two difficulties arise. The first is that our monstrous production requires the antecedent strings to occur in a given order—an undesirable restriction we will lift shortly. The other difficulty is more serious. We have to check that the analyses of the antecedents are *correct* in that the strings assigned to the $\$_{ij}$'s are *proper* parts of old theorems and axioms. The danger is that a $\$_{ij}$ may contain too much—it may run from the beginning of one P string, through some B's, to the end of a different P string. Now this will be the case if, and only if, a $\$_{ij}$ contains one or more B's in its interior. Our monstrous production is designed to make it easy to test for this contingency. That is why the production appends *the concatenation of all the variable strings,* in the form

$$X\$_{11}\ldots\$_{ij}\ldots\$_{pn_p}X$$

(Observe also that the consequent does not end in a B; so the monstrous production cannot operate on it again, immediately.) We can now test to

see that this appendage contains no B's by using the productions

$$\$_1 X x \$_2 X \rightarrow \$_1 X \$_2 X \qquad \text{(all } x \text{ in } A\text{)}$$
$$\$XX \rightarrow \$BB$$

The result is that the inner X moves to the right, erasing lower-case letters. If it ever encounters a B, the string is doomed. If the inner X gets across to the final X, there were no B's in the appendage, and the system reverts to the axiom form, but with the new theorem appended to the theorem list!

We must now lift the restriction that the antecedent components occur in some fixed order. One method might be to introduce a distinct production system for each permutation of the antecedents—this would mean $p!$ forms for each original production. A more elegant solution is simply to adjoin the single production

$$B\$_1 BB\$_2 BB\$_3 B \rightarrow B\$_1 BB\$_2 BB\$_3 BB\$_2 B$$

PROBLEM. Why does this eliminate the need for permuted productions?

As usual, we must finish by providing a mechanism by which P strings can be released without any auxiliary letters (namely, B's and X's). To do this, we use the same trick we used in checking for B's, by adding the productions

$$B\$_1 B\$_2 B\$_3 B \rightarrow Y\$_2 Y$$
$$\$_1 Y x \$_2 Y \rightarrow \$_1 x Y \$_2 Y \qquad \text{(all } x \text{ in } A\text{)}$$
$$\$YY \rightarrow \$$$

which releases any lower-case string that is enclosed by B's.

13.3 A UNIVERSAL CANONICAL SYSTEM

It is possible to construct a canonical system that is a sort of analogue to a universal Turing machine. We will do this by using the methods developed in the previous section. The innovation is that instead of applying the technique to the *theorems* of another system we will apply it to the *productions* of the other system, regarding the productions themselves as strings of symbols. A "universal" system is one which works with the *description* of another system to compute what that other system does. Here, as we shall see, it is so easy to work with descriptions of Post systems that no arithmetic tricks, or the equivalent, are needed.

THEOREM 13.3-1

There exists a certain system U of Post productions with the property: Given any other canonical system Q, then we can construct a single axiom A_Q for U such that the system U with the axiom A_Q is a canonical extension of Q in the following sense. Let $(a_1, \ldots, a_r)$ be the alphabet of Q. Let **a** and **b** be new letters, and encode $(a_1, \ldots, a_r)$ into the alphabet **(a, b)** by representing a_j by j **a**'s followed by a **b**. Then the strings of U, with axiom A_Q, that have only letters **a** and **b** will be exactly this encoding of the theorems of Q.

To prove this, we have to exhibit the productions of U and show how to construct the axiom A_Q.

By the theorems proved earlier in the chapter, the arbitrary system Q has a normal extension P. Suppose that this system P has axioms $\Phi_1, \ldots, \Phi_s$ and productions $g_i\$ \rightarrow \$h_i$. Let $(a_1, \ldots, a_r)$ be the alphabet of P and let A, C, S, and T be new letters. Our system U will be supplied with the axiom

$$A_P = \boxed{ACAg_1Ch_1Ag_2Ch_2A \ldots Ag_nCh_nAS\Phi_1S\Phi_2S \ldots S\Phi_sSTTT}$$

Observe that this axiom is a complete description of P since from it one can reconstruct all the axioms and productions of P.

Next, consider the production

$$\boxed{\begin{array}{l}\$_1A\$_AC\$_CA\$_2S\$_B\$S\$_3TTT \\ \qquad\qquad\qquad\qquad \rightarrow \\ \$_1A\$_AC\$_CA\$_2S\$_B\$S\$_3\$\$_CST\$_A\$\$_CT\$_B\$\$_CT\$\$_C\end{array}} \qquad (\pi_U)$$

This production looks (in the first, or production, part of a theorem) for a "production" of the form $A\$_AC\$_CA$ that matches a "theorem" of the form $S\$_B\$S$ and attempts to adjoin to the theorem list the string $\$\$_C$ in accord with the "production" $\$_A\$ \rightarrow \$\$_C$. This would be a legitimate thing to do only (1) if it is the case that the strings $\$_B$ and $\$_A$ are *identical* and (2) if none of the strings concerned (namely, \$, $\$_B$, $\$_A$, and $\$_C$) contain any upper-case letters. For in that case we can be sure that $\$\$_C$ is a theorem of the system P and that it is appropriate to add it to the theorem- (or axiom-) list represented by the strings between the S's. The following productions check both these conditions:

$$\boxed{\$_1Tx\$_2Tx\$_3T\$_4 \rightarrow \$_1T\$_2T\$_3T\$_4 \qquad (x \text{ in } (a_1, \ldots, a_r))} \qquad (\pi_i)$$

The trick is this: These productions remove, one at a time, *identical lower-case* letters from $\$_A\$\$_C$ and from $\$_B\$\$_C$. If, and only if, both

strings are identical and entirely lower-case, both strings will vanish, leaving a string of the form $\$_1 TTT\$_2$. So we adjoin also the production

$$\boxed{\$_1 TTT\$_2 \rightarrow \$_1 TTT} \qquad (\pi_T)$$

which restores the string to its original form, except with the new theorem $\$\$_C$ inserted in the theorem list. To release the new theorem into the pure lower-case alphabet, we appended an extra copy of it at the end of the intermediate string, and it can be released by the production

$$\boxed{\$_1 TTT\$_2 \rightarrow \$_2} \qquad (\pi_R)$$

We put the initial ACA into the main production just so that the axioms themselves could be produced as theorems of the system, just in case the production $\$ \rightarrow \$$ was not included among the productions of P.

The system has the defect that the productions used above depend on the alphabet of P. If we use the binary encoding mentioned in the statement of the theorems, then the system of test productions is replaced by two fixed productions:

$$\$_1 T\mathbf{a}\$_2 T\mathbf{a}\$_3 T\$_4 \rightarrow \$_1 T\$_2 T\$_3 T\$_4 \qquad (\pi_{\mathbf{a}})$$

$$\$_1 T\mathbf{b}\$_2 T\mathbf{b}\$_3 T\$_4 \rightarrow \$_1 T\$_2 T\$_3 T\$_4 \qquad (\pi_{\mathbf{b}})$$

and our whole system U has only five productions: π_U, π_T, π_R, $\pi_{\mathbf{a}}$, and $\pi_{\mathbf{b}}$!

PROBLEM 13.3-1. The production π_T

$$\$_1 TTT\$_2 \rightarrow \mathsf{S}_1 TTT$$

can be eliminated by making a trivial addition to the antecedent of the main production π_U. What is this change? This reduces the universal system to just four productions. I can see no way to reduce the number of productions further because it would seem that we need one to "do the work," two to check the binary alphabet conditions, and one more to release the pure strings. Such reasoning, however, often turns out to be unsound.

PROBLEM 13.3-2. Construct a version of U that has only one upper-case letter, and only the lower case letters **a** and **b**. Can you do it with still only four productions? The resulting system is probably minimal, in some sense, with three symbols and four productions.

PROBLEM 13.3-3. Construct a version of U with only normal productions. Do not try to minimize the numbers of letters or productions.

PROBLEM 13.3-4. Using the analogy with a universal Turing machine, construct an unsolvable decision problem about the strings produced by U

as a function of the given axiom. Consider the prospect of developing a theory of computability on this basis as compared with the Turing or recursive-function basis.

PROBLEM 13.3-5. We have not quite proved everything claimed in theorem 13.3-1. We have proved it for any normal system P. Now prove it for the original, general system Q. The present system, as described, will release the theorems of P (which include those of Q but also some others). To complete the proof requires a slightly more complicated release system.

PROBLEM 13.3-6. We have never allowed an antecedent with a double occurrence of a $\$_i$. As noted in the remarks of section 12.5, Post allowed this, but we feel that it is not entirely in the spirit of the finite-state approach. In any case, show that for any system with this more general production permitted, there are canonical extensions of our more conservative kind. In particular, begin by showing that we can simulate the effect of the production

$$\$_1 0 \$_1 1 1 \$ \rightarrow \$_1 0 \$$$

by an extension that does not use double $\$_i$'s in its antecedent. Use the T method, as in section 13.1.[2]

NOTES

1. For example, there is no canonical system for the theorems of the propositional calculus, in its conventional form, that does not use at least one extension letter. Or so I believe, but I have not been able to reconstruct what I think was a proof of this.
2. See note 1 of chapter 12.