JOURNAL OF COMPUTER AND SYSTEM SCIENCES: 3, 37–56 (1969)

Syntax Directed Translations and the Pushdown Assembler*

A. V. AHO AND J. D. ULLMAN

Bell Telephone Laboratories, Incorporated
Murray Hill, New Jersey 07974

Received August 1, 1968; revised October 18, 1968

ABSTRACT

It is shown that there exists an infinite hierarchy of syntax-directed translations according to the number of nonterminals allowed on the right side of productions of the underlying context-free grammar. A device called the pushdown assembler is defined, and it is shown capable of performing exactly the syntax-directed translations.

I. INTRODUCTION

There has been considerable interest recently in the formal specification of translations. These specifications are expected to be of use in automated compiler writing.

To date, most of the methods used center around a context-free grammar. The input is parsed according to a given context-free grammar. Once the parse tree has been found, the order of the descendants of any node may be permutted according to fixed rules. There may also be rules that delete or introduce nodes with terminal labels.

Such a scheme is generally called a syntax-directed translation. Irons [*1*] was among the first to use this scheme. The syntax directed translation or a similar scheme have been studied in [*2-6*].

A subclass, called simple syntax-directed translations, allows no permutation of the order in which edges extend from nodes of the parse tree. These translations were shown equivalent to the translations definable by nondeterministic pushdown transducers [*3*]. In this paper, we intend to give a generalization of the pushdown automaton, called the pushdown assembler, which is capable of defining every syntax-directed translation.

We shall also study syntax-directed translations and show that there is an infinite hierarchy of them according to the number of nonterminals allowed on the right side of productions of the underlying context-free grammar. The existence of this hierarchy is relevant to the appropriateness of the definition of the pushdown assembler.

* Portions of this paper appeared in the Proceedings of the IEEE 9th Annual Symposium on Switching and Automata Theory.

II. Syntax Directed Translations

A syntax directed translation scheme (SDTS) is a system that generalizes the notion of a context-free grammar (CFG).[1] It will be denoted $G = (V, \Sigma, \Delta, R, S)$, where V, Σ and Δ are finite sets of *variables*, *input symbols*, and *output symbols*, respectively. V is disjoint from $\Sigma \cup \Delta$. S in V is the *start symbol*. R is a finite set of *rules*, a term we shall define more fully later. We first need an auxiliary definition.

A *form* of G is a triple (α, β, Π), where α is in $(V \cup \Sigma)^*$, β is in $(V \cup \Delta)^*$ and Π is a permutation.[2] The number of variables in the strings α and β must be equal, say k in each. Π must then be a permutation on k objects. For all i, $1 \leqslant i \leqslant k$, the ith variable of α (from the left) is the same as the $\Pi(i)$th variable of β. We say that the ith variable of α and the $\Pi(i)$th variable of β *correspond*.

Convention. If it is obvious which variables of α and β correspond (because no variable appears more than once in α or β), we shall often omit the permutation and write the form as (α, β).

A *rule* is an object $A \to (\alpha, \beta, \Pi)$, where A is a variable and (α, β, Π) a form.

Suppose $(\alpha_1, \beta_1, \Pi_1)$ is a form of G and A the ith variable of α_1, from the left. Also, suppose $A \to (\gamma, \delta, \Pi)$ is a rule. Then we can construct a form $(\alpha_2, \beta_2, \Pi_2)$ by replacing the ith variable of α_1 by γ and the $\Pi_1(i)$th variable of β_1 by δ. Π_2 is the permutation such that variables of α_1 other than the ith correspond to the same symbol in β_2 as in β_1 and each variable of γ corresponds to the variable of δ to which it corresponded according to Π.

Formally, let γ have m variables, $m \geqslant 0$. Π_2 is defined by:

(1) For all $j < i$, if $\Pi_1(j) < \Pi_1(i)$, then $\Pi_2(j) = \Pi_1(j)$, and if $\Pi_1(j) > \Pi_1(i)$, then $\Pi_2(j) = \Pi_1(j) + m - 1$.

(2) For all $j > i$, if $\Pi_1(j) < \Pi_1(i)$, then $\Pi_2(j + m - 1) = \Pi_1(j)$, and if $\Pi_1(j) > \Pi_1(i)$, then $\Pi_2(j + m - 1) = \Pi_1(j) + m - 1$.

(3) For all d, $1 \leqslant d \leqslant m$, $\Pi_2(i + d - 1) = \Pi_1(i) + \Pi(d) - 1$.

We say, if all of the above is true, that

$$(\alpha_1, \beta_1, \Pi_1) \underset{G}{\Rightarrow} (\alpha_2, \beta_2, \Pi_2).$$

[1] A CFG G is a system (V, Σ, P, S), where V and Σ are disjoint finite sets of *variables* and *terminals*, respectively. S in V is the *sentence symbol*. P is the finite set of *productions* of the form $A \to \alpha$, where A is in V and α in $(V \cup \Sigma)^*$. If $A \to \alpha$ is in P, β and γ in $(V \cup \Sigma)^*$, then we say $\beta A \gamma \underset{G}{\Rightarrow} \beta\alpha\gamma$. The relation $\overset{*}{\underset{G}{\Rightarrow}}$ is defined by $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$, and whenever $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ and $\beta \underset{G}{\Rightarrow} \gamma$, then $\alpha \overset{*}{\underset{G}{\Rightarrow}} \gamma$. The *language generated* by G, denoted $L(G)$, is $\{w \mid w \text{ is in } \Sigma^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w\}$. $L(G)$ is a *context-free language* (CFL).

[2] A permutation Π on k objects will be denoted $[i_1, i_2, \ldots, i_k]$, where i_j, $1 \leqslant j \leqslant k$, is an integer between 1 and k, and $i_m \neq i_n$ if $m \neq n$. $\Pi(j)$ is defined to be i_j; $\hat{\Pi}(j)$ is that k such that $\Pi(k) = j$. $\hat{\Pi}$ thus represents the "inverse" of Π in the group of permutations of k objects.

Convention. We will denote variables by capital letters at the beginning of the alphabet. Strings consisting of variables and input or output symbols will be denoted by small Greek letters. Small letters at the end of the alphabet will denote strings consisting only of input or output symbols, and small letters at the beginning of the alphabet denote input or output symbols. Capital Π's are permutations. We shall often omit comments on the nature of objects represented by symbols if they obey this convention.

Define the relation $\overset{*}{\underset{G}{\Rightarrow}}$ by $(\alpha, \beta, \Pi) \overset{*}{\underset{G}{\Rightarrow}} (\alpha, \beta, \Pi)$, and if $(\alpha_1, \beta_1, \Pi_1) \overset{*}{\underset{G}{\Rightarrow}} (\alpha_2, \beta_2, \Pi_2)$ and $(\alpha_2, \beta_2, \Pi_2) \underset{G}{\Rightarrow} (\alpha_3, \beta_3, \Pi_3)$, then $(\alpha_1, \beta_1, \Pi_1) \overset{*}{\underset{G}{\Rightarrow}} (\alpha_3, \beta_3, \Pi_3)$.

The *syntax-directed translation* (SDT) T defined by G, denoted $T(G)$, is

$$\{(x, y) \mid (S, S) \overset{*}{\underset{G}{\Rightarrow}} (x, y)\}.^{3}$$

III. A Hierarchy of Syntax-Directed Translations

$G = (V, \Sigma, \Delta, R, S)$ is said to be of *order* k if for all rules $A \to (\alpha, \beta, \Pi)$ in R, there are no more than k variables in α. An SDT is of order k if it is defined by some SDTS of order k. Let $\mathscr{T}_k$ be the set of SDT's of order k. Clearly $\mathscr{T}_k$ contains $\mathscr{T}_{k-1}$ for all $k \geqslant 2$. We shall show that, except for $k = 3$, $\mathscr{T}_k$ properly contains $\mathscr{T}_{k-1}$.[4] Also, $\mathscr{T}_2 = \mathscr{T}_3$.

It is easy to see that $\mathscr{T}_1 \neq \mathscr{T}_2$. Let $G = (V, \Sigma, \Delta, R, S)$ be an SDTS. Let G_1 be the CFG (V, Σ, P, S), where P consists of those productions $A \to \alpha$ such that $A \to (\alpha, \beta, \Pi)$ is a rule in R. Then $L(G_1)$ is the domain[5] of $T(G)$. If G is of order 1, G_1 is a linear grammar. Thus, the domain of an SDTS of order 1 is linear. However, any context-free language is generated by a CFG with at most two variables on the right of any production [7]. Hence, any context free language is the domain of an SDT of order 2. We thus have:

THEOREM 3.1. *$\mathscr{T}_1$ is properly contained in $\mathscr{T}_2$.*

Proof. Let L be a context-free language which is not linear. Let $T = \{(x, x) \mid x \text{ in } L\}$. Using the result that every CFL is generated by a Chomsky normal form grammar [7], one can show that T is in $\mathscr{T}_2$. But T is not in $\mathscr{T}_1$, since the domain of T is not linear.

We will have several uses for the following lemma.

[3] Note that permutations are omitted from the forms because S obviously corresponds to S in the first form and there are no variables in the second.

[4] The analous result for CFG'S is false. Chomsky [7] showed that every context-free language is generated by a grammar in which no production has more than two symbols on the right.

[5] The *domain* of T is the set $\{x \mid (x, y)$ is in T for some $y\}$. Also, the *range* of T is the set $\{y \mid (x, y)$ is in T for some $x\}$.

LEMMA 3.1. *If $T = T(G')$ for some* SDTS *G' of order $k \geqslant 2$ then $T = T(G)$, where $G = (V, \Sigma, \Delta, R, S)$ is a scheme of order $k \geqslant 2$ having the property that if $A \to (\alpha, \beta, \Pi)$ is a rule, then α and β are either both in V^* or α is in Σ^* and β in Δ^*.*

Proof. Let $G' = (V', \Sigma, \Delta, R', S)$. For each rule

$$A \to (w_1 A_1 w_2 A_2 \cdots A_m w_{m+1}, x_1 B_1 x_2 B_2 \cdots B_m x_{m+1}, \Pi)$$

in R', introduce new variables $C_1, C_2, \ldots, C_{m+1}$ and $D_1, D_2, \ldots, D_{m+1}$ to V. Then, replace this rule by:

$$\begin{aligned} A &\to (C_{m+1} D_{m+1}, C_{m+1} D_{m+1}), \\ D_{m+1} &\to (w_{m+1}, x_{m+1}), \\ C_{m+1} &\to (C_1 C_2 \cdots C_m, C_{\bar{\Pi}(1)} C_{\bar{\Pi}(2)} \cdots C_{\bar{\Pi}(m)}, \Pi). \\ C_j &\to (D_j A_j, D_j A_j), \qquad \text{for} \quad 1 \leqslant j \leqslant m. \\ D_j &\to (w_j, x_{\Pi(j)}), \qquad \text{for} \quad 1 \leqslant j \leqslant m. \end{aligned}$$

Since $C_1, C_2, \ldots, C_{m+1}$ and $D_1, D_2, \ldots, D_{m+1}$ are used in only this way, it should be clear that the new rules together can only produce the effect of the original rule. Therefore, a proof that $T(G) = T(G')$ is straightforward, and is omitted.

An SDT satisfying Lemma 3.1 is said to be in *normal form.*

THEOREM 3.2. $\mathscr{T}_2 = \mathscr{T}_3$.

Proof. Let T be in $\mathscr{T}_3$. T is defined by $G = (V, \Sigma, \Delta, R, S)$, an SDTS of order 3. We may assume, by Lemma 3.1, that G is in normal form. We will construct an equivalent SDTS G_1 of order 2. For each rule $A \to (\alpha, \beta, \Pi)$, where α consists of three variables, we introduce a new variable C and replace $A \to (\alpha, \beta, \Pi)$ by two other rules. Let $\alpha = B_1 B_2 B_3$ and $\beta = B_{\bar{\Pi}(1)} B_{\bar{\Pi}(2)} B_{\bar{\Pi}(3)}$. For each Π, the two rules replacing $A \to (\alpha, \beta, \Pi)$ are given in Table 3.1.

TABLE 3.1

RULES REPLACING $A \to (\alpha, \beta, \Pi)$

Π	Rules	
[1, 2, 3]	$A \to (B_1 C, B_1 C, [1, 2])$	$C \to (B_2 B_3, B_2 B_3, [1, 2])$
[1, 3, 2]	$A \to (B_1 C, B_1 C, [1, 2])$	$C \to (B_2 B_3, B_3 B_2, [2, 1])$
[2, 1, 3]	$A \to (C B_3, C B_3, [1, 2])$	$C \to (B_1 B_2, B_2 B_1, [2, 1])$
[2, 3, 1]	$A \to (C B_3, B_3 C, [2, 1])$	$C \to (B_1 B_2, B_1 B_2, [1, 2])$
[3, 1, 2]	$A \to (B_1 C, C B_1, [2, 1])$	$C \to (B_2 B_3, B_2 B_3, [1, 2])$
[3, 2, 1]	$A \to (C B_3, B_3 C, [2, 1])$	$C \to (B_1 B_2, B_2 B_1, [2, 1])$

Let $G_1 = (V_1, \Sigma, \Delta, R_1, S)$, where V_1 is V together with any variables introduced above. R_1 is the resulting set of rules.

An inspection of Table 3.1 shows that for each value of Π, the successive application of the two rules given yields the same form as an application of the rule $A \to (\alpha, \beta, \Pi)$. Also, suppose the first of the two rules replacing $A \to (\alpha, \beta, \Pi)$ is ever used in a derivation.[6] Since the new variable C is associated only with $A \to (\alpha, \beta, \Pi)$, there is only one rule with C on the left. This rule must eventually be used to replace C, if the derivation is to lead to a form with no variables. Thus, $T(G) = T(G_1)$.

We shall now offer an infinite sequence of SDT's which are generated only by SDTS's of increasingly higher order. To that effect, let i be an integer greater than 1. Define Π_{2i} to be the permutation $[i+1, 1, i+2, 2, \ldots, 2i, i]$. That is, $\Pi_4 = [3, 1, 4, 2]$, $\Pi_6 = [4, 1, 5, 2, 6, 3]$, etc. Define Π_{2i-1} to be the permutation $[i, 2i-1, 1, 2i-2, 2, \ldots, i+1, i-1]$. That is, $\Pi_5 = [3, 5, 1, 4, 2]$, $\Pi_7 = [4, 7, 1, 6, 2, 5, 3]$, etc. Let Σ_k, $k \geqslant 4$, be the alphabet $\{a_1, a_2, \ldots, a_k\}$, and define T_k to be the translation

$$\{(a_1^{i_1} a_2^{i_2} \cdots a_k^{i_k}, b_1^{j_1} b_2^{j_2} \cdots b_k^{j_k}) \mid \text{for } 1 \leqslant m \leqslant k, i_m \geqslant 1, b_{\Pi_k(m)} = a_m \text{ and } j_{\Pi_k(m)} = i_m\}.$$

T_k is in $\mathscr{T}_k$, since $T_k = T(G_k)$, where $G_k = (\{S, A_1, A_2, \ldots, A_k\}, \Sigma_k, \Sigma_k, R, S)$ and R consists of

$$S \to (A_1 A_2 \cdots A_k, A_{\bar{\Pi}_k(1)} A_{\bar{\Pi}_k(2)} \cdots A_{\bar{\Pi}_k(k)}, \Pi_k)$$

and $A_j \to (a_j A_j, a_j A_j)$ and $A_j \to (a_j, a_j)$ for each j, $1 \leqslant j \leqslant k$.

We claim that for all $k \geqslant 4$, T_k is not in $\mathscr{T}_{k-1}$. To prove this, we will make use of some constructions that are straightforward generalizations of analogous constructions in context-free language theory.

The argument is, essentially, that if $T_k = T(G)$, and G is of order $k-1$, then in almost every derivation of G, there is a variable which generates two different symbols say a_i and a_j. If G is to define T_k, then a_i and a_j must appear next to each other both in the domain and the range. This implies $|i - j| \leqslant 1$ and $|\Pi_k(i) - \Pi_k(j)| \leqslant 1$. For $k \geqslant 4$, an examination of Π_k reveals that the above inequalities cannot hold simultaneously.

LEMMA 3.2. *Every* SDT T *in* $\mathscr{T}_k$, $k \geqslant 2$, *is* $T(G)$ *for some normal form* SDTS $G = (V, \Sigma, \Delta, R, S)$ *of order* k, *such that:*

(1) *S does not appear in any α or β such that $A \to (\alpha, \beta, \Pi)$ is in R.*

(2) *If $A \neq S$, then $A \to (\epsilon, \epsilon)$ is not in R.*

(3) *For no A and B in V is $A \to (B, B)$ in R.*

[6] *A derivation* is a sequence of forms $F_1, F_2, \ldots, F_r$, such that $F_i \underset{G}{\Rightarrow} F_{i+1}$ for $1 \leqslant i < r$.

(4) *For every variable A, there exist y in Σ^* and z in Δ^* such that*

$$(A, A) \overset{*}{\underset{G}{\Rightarrow}} (y, z).$$

(5) *For every variable A, there exist α_1, α_2, β_1, β_2 and Π such that*

$$(S, S) \overset{*}{\underset{G}{\Rightarrow}} (\alpha_1 A \alpha_2, \beta_1 A \beta_2, \Pi).$$

Proof. For each of the five parts, we will assume we have a normal form SDTS $G' = (V', \Sigma, \Delta, R', S')$ satisfying any lower-numbered parts of the lemma. We shall give constructions that will introduce each property in turn, while preserving normal form, order and the previously introduced properties. The proofs that these constructions do not alter the translation defined are similar to those for analogous results concerning context free grammars. Each can be found in [8], and we will omit the details here.

(1) Let S be a new symbol. $V = V' \cup \{S\}$. $R = R' \cup \{S \to (\alpha, \beta, \Pi) \mid R'$ contains the rule $S' \to (\alpha, \beta, \Pi)\}$. Then $G = (V, \Sigma, \Delta, P, S)$ is an SDTS equivalent to G' satisfying (1).

(2) If $A \neq S$ and $(A, A) \overset{*}{\underset{G'}{\Rightarrow}} (\epsilon, \epsilon)$, say A is "type 1."[7] Otherwise, say A is "type 2." Form R from R' by removing all rules of the form $A \to (\epsilon, \epsilon)$ for $A \neq S$. Then, replace each rule

$$B \to (A_1 A_2 \cdots A_m, A_{\Pi(1)} A_{\Pi(2)} \cdots A_{\Pi(m)}, \Pi)$$

by the set of rules of the form

$$B \to (\alpha_1 \alpha_2 \cdots \alpha_m, \alpha_{\Pi(1)} \alpha_{\Pi(2)} \cdots \alpha_{\Pi(m)}, \Pi'),$$

where, for $1 \leqslant i \leqslant m$, if A_i is type 2, then $\alpha_i = A_i$. If A_i is type 1, α_i may be A_i or ϵ. However, if $B \neq S$, not all of $\alpha_1, \alpha_2, \ldots, \alpha_m$ may be ϵ. Π' is the permutation defined so that remaining variables correspond to the variables to which they corresponded in the original rule. That is, $\Pi'(i) = j$ if for some k, $\alpha_k \neq \epsilon$, exactly $i - 1$ of $\alpha_1, \alpha_2, \ldots, \alpha_{k-1}$ are not ϵ, $\Pi(k) = n$ and exactly $j - 1$ of $\alpha_{\Pi(1)}, \alpha_{\Pi(2)}, \ldots, \alpha_{\Pi(n-1)}$ are not ϵ.

(3) Form R from R' by removing all rules of the form $A \to (B, B)$ for each A in V'. However, if for some C, $(A, A) \overset{*}{\underset{G'}{\Rightarrow}} (C, C)$ and $C \to (\alpha, \beta, \Pi)$ is a rule in R with α and β not single variables, then introduce the rule $A \to (\alpha, \beta, \Pi)$ to R.

[7] This question, as well as those mentioned in the other parts of the lemma are decidable, but decidability is not required for proof.

(4) For each A in V', if such y and z do not exist, remove A from V' and all rules involving A from R'. V and R are the resulting variables and rules; $S = S'$.

(5) Again, if no such α_1, α_2, β_1 and β_2 exist, remove A from V' and rules involving A from R'.

We now proceed to the argument that T_k is not in $\mathscr{T}_{k-1}$, for $k \geqslant 4$. Let us fix our attention on a particular integer $k \geqslant 4$ and a hypothetical SDTS $G = (V, \Sigma_k, \Sigma_k, R, S)$, of order $k-1$, satisfying Lemma 3.2 and defining T_k. We shall show by a series of lemmas (3.3-3.6) that no such G exists.

Let a be in Σ_k and w in Σ_k^*. Define $\#_a(w)$ to be the number of instances of a in w. Let Σ be a subset of Σ_k, A in V and d an integer. We say that Σ is (A, d) *bounded in the domain* (alt. *range*) if whenever $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (w, x)$, there is some a in Σ such that $\#_a(w)$ (alt. $\#_a(x)$) is less than d. If for no integer d is $\Sigma(A, d)$ bounded in the domain (alt. range), then we say that A *covers* Σ in the domain (alt. range).

LEMMA 3.3. *A covers Σ in the domain if and only if A covers Σ in the range.*

Proof. Suppose A covers Σ in the domain, but Σ is (A, d) bounded in the range. By Lemma 3.2, we may write $(S, S) \overset{*}{\underset{G}{\Rightarrow}} (w_1 A w_2, w_3 A w_4)$ for some w_1, w_2, w_3 and w_4. Let $e = |w_3 w_4|$.[8] Since A covers Σ in the domain, we can find y and z in Σ_k^* such that $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (y, z)$, and for all a in Σ, $\#_a(y) \geqslant e + d$. However, since Σ is (A, d) bounded in the range, there is some b in Σ such that $\#_b(z) < d$. But $(S, S) \overset{*}{\underset{G}{\Rightarrow}} (w_1 y w_2, w_3 z w_4)$; $\#_b(w_1 y w_2) \geqslant e + d$ and $\#_b(w_3 z w_4) < e + d$. These relations imply that G does not define T_k, since the translation T_k preserves the number of occurrences of each symbol. A similar argument applies if A covers Σ in the range.

By Lemma 3.3, it is sufficient to say that "Σ is (A, d)-bounded" or "Σ is covered by A," without specifying the domain or range.

LEMMA 3.4. *If A covers Σ_k, then there exists a rule $A \to (A_1 A_2 \cdots A_m, \beta, \Pi)$ in R and subsets of Σ_k: $\Sigma^{(1)}, \Sigma^{(2)}, \ldots, \Sigma^{(m)}$, such that A_j covers $\Sigma^{(j)}$, for $1 \leqslant j \leqslant m$, and*

$$\bigcup_{1 \leqslant j \leqslant m} \Sigma^{(j)} = \Sigma_k .$$

Proof. For any $\Sigma \subseteq \Sigma_k$ and any B in V, either B covers Σ, or Σ is (B, d) bounded for some d. Let d_0 be the largest d such that some Σ is (B, d) bounded but not $(B, d-1)$ bounded. Let $p = d_0(k-1) + 1$. There must be some strings y and z such that $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (y, z)$,

$$y = a_1^{m_1} a_2^{m_2} \cdots a_k^{m_k} \qquad \text{and} \qquad z = a_{\Pi_k(1)}^{n_1} a_{\Pi_k(2)}^{n_2} \cdots a_{\Pi_k(k)}^{n_k} ,$$

[8] $|x|$ is the length (number of symbols) of string x.

and all of $m_1, m_2, \ldots, m_k, n_1, n_2, \ldots, n_k$ are at least p. (If not, then Σ_k would be (A, p)-bounded.) The first step of the derivation of (y, z) must be the application of some rule $A \to (A_1A_2 \cdots A_m, \beta, \Pi)$. Then there are strings $y_1, y_2, \ldots, y_m$ and $z_1, z_2, \ldots, z_m$, such that $(A_i, A_i) \overset{*}{\underset{G}{\Rightarrow}} (y_i, z_i)$, for $1 \leqslant i \leqslant m$,

$$y_1y_2 \cdots y_m = y \qquad \text{and} \qquad z_{\Pi(1)}z_{\Pi(2)} \cdots z_{\Pi(m)} = z.$$

Let $\Sigma^{(j)} = \{a \mid \#_a(y_j) > d_0\}$. Since $m \leqslant k - 1$, every a in Σ_k is in some $\Sigma^{(j)}$, else, $\#_a(y) \leqslant d_0m < p$.

Since $\#_a(y_j) > d_0$ for each a in $\Sigma^{(j)}$, A_j covers $\Sigma^{(j)}$. For if not, then $\Sigma^{(j)}$ is (A_j, d) bounded for some $d > d_0$, but not (A_j, d_0) bounded. This leads to a contradiction about our choice of d_0.

There are two orderings of the symbols of Σ_k that are associated with T_k. One is the ordering $a_1, a_2, \ldots, a_k$, in which the symbols appear in the domain. The other is the ordering $a_{\Pi_k(1)}, a_{\Pi_k(2)}, \ldots, a_{\Pi_k(k)}$, in which the symbols appear in the range. We say a_i is *between* a_m and a_n if either $m < i < n$ or $\Pi_k(m) < \Pi_k(i) < \Pi_k(n)$.

LEMMA 3.5. *Suppose A covers Σ_k and $A \to (A_1A_2 \cdots A_m, \beta, \Pi)$ is a rule satisfying Lemma 3.4. If A_j covers $\{a_r\}$ and $\{a_s\}$,[9] and a_i is between a_r and a_s, then A_j covers $\{a_i\}$ and for no $p \neq j$ does A_p cover $\{a_i\}$.*

Proof. We will treat the case where $r < i < s$. The case $\Pi_k(r) < \Pi_k(i) < \Pi_k(s)$ is handled in an analogous manner. Suppose that A_p covers $\{a_i\}$, $p \neq j$. Also, suppose $p > j$, i.e. A_p is to the right of A_j in the string $A_1A_2 \cdots A_m$. By allowing A_j to derive a string with a_s in it and A_p to derive a string with a_i in it, we have a situation $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (w, x)$, where w has an instance of a_s to the left of a_i. By Lemma 3.2 (5), there is a pair in $T(G)$ not in T_k. We can arrive at the same contradiction if $p < j$ by letting A_j generate a string with a_r in it.

Now, by Lemma 3.4, some one of $A_1, A_2, \ldots, A_m$ covers a set containing a_i. By the above, this one can only be A_j, so surely A_j covers $\{a_i\}$.

LEMMA 3.6. *If A covers Σ_k, then there is some A_j, $1 \leqslant j \leqslant m$, and a rule $A \to (A_1A_2 \cdots A_m, \beta, \Pi)$ such that A_j covers Σ_k.*

Proof. We may assume that $A \to (A_1A_2 \cdots A_m, \beta, \Pi)$ is a rule satisfying Lemma 3.4. That is, there exist sets $\Sigma^{(1)} \cup \ldots \cup \Sigma^{(m)} = \Sigma_k$ such that A_i covers $\Sigma^{(i)}$, for $1 \leqslant i \leqslant m$. Since $m < k$, there must be some j such that $\Sigma^{(j)}$ contains at least two elements, say a_r and a_s. We wish first to use Lemma 3.5 to show that A_j covers $\{a\}$ for all a in Σ_k.

[9] Note that this does not imply that A_j covers $\{a_r, a_s\}$. However, the following "law" applies to the covering relation. If A covers Σ, and $\Sigma' \subseteq \Sigma$, then A covers Σ'.

It is, by Lemma 3.5, sufficient to show that A_j covers $\{a_1\}$ and $\{a_k\}$. We will treat the case when k is even and leave the case of odd k, which is similar, to the reader.

Observe that for even k, $\Pi_k(1) = \frac{1}{2}k + 1$ and $\Pi_k(k) = \frac{1}{2}k$. Moreover $\Pi_k(i) \leqslant \frac{1}{2}k$ for even i, and $\Pi_k(i) \geqslant \frac{1}{2}k + 1$ for odd i.

Thus, if exactly one of r and s is even, A_j must cover $\{a_1\}$ and $\{a_k\}$, from which the result is immediate.

If r and s are both odd, there must be an even integer p such that a_p is between a_r and a_s. If r and s are both even, we can find an odd p such that a_p is between a_r and a_s. In either case, A_j covers $\{a_p\}$, from which it follows that A_j covers $\{a_1\}$ and $\{a_k\}$. This argument is complete for even k. A simple extension of these considerations proves the result for odd k.

It is not possible that A_p, $p \neq j$, covers any $\{b\}$, for b in Σ_k, since every b in Σ_k is between two other elements of Σ_k (either in the domain ordering or the range ordering). If A_p covered $\{b\}$, a violation of Lemma 3.5 would occur. Thus, by Lemma 3.4, A_j covers Σ_k.

THEOREM 3.3. *$\mathscr{T}_k$ properly contains $\mathscr{T}_{k-1}$, for $k \geqslant 4$.*

Proof. It suffices to show that T_k is not in $\mathscr{T}_{k-1}$, by showing that the grammar G, with which we have been dealing, does not exist. We have shown, in Lemma 3.6, that for each A which covers Σ_k, there is a rule $A \to (A_1A_2 \cdots A_m, \beta, \Pi)$, such that for some j between 1 and m, A_j covers Σ_k.

Surely, S covers Σ_k. Let V have t elements. By Lemma 3.6, we may construct a sequence of variables $B_0, B_1, \ldots, B_t$, such that $B_0 = S$, B_i covers Σ_k for all i, and for $0 \leqslant i < t$, there is a rule $B_i \to (\alpha_i B_{i+1}\beta_i, \gamma_i B_{i+1}\delta_i, \Pi_i)$. Not all of $B_0, B_1, \ldots, B_t$ are distinct. Let B appear twice in the sequence. Now, $|\alpha_i\beta_i| \neq 0$ or $|\gamma_i\delta_i| \neq 0$, by Lemma 3.2. Also, by Lemma 3.2, each variable involved derives some pair of input and output strings. Thus, we may construct derivations

$$(S, S) \overset{*}{\underset{G}{\Rightarrow}} (w_1Bw_2, w_3Bw_4) \overset{*}{\underset{G}{\Rightarrow}} (w_1x_1{}^pBx_2{}^pw_2, w_3x_3{}^pBx_4{}^pw_4)$$

for each $p \geqslant 1$, where either $|x_1x_2| > 0$ or $|x_3x_4| > 0$.

But for each a in Σ_k, $\#_a(x_1x_2) = \#_a(x_3x_4)$, else we can easily construct a pair (y, z) in $T(G)$ such that for some a in Σ_k, y and z do not have the same number of occurrences of a. Since B generates strings with occurrences of all symbols in Σ_k, we could easily construct a pair (y, z) in $T(G)$ but not in T_k unless x_1 consist only of a_1's and x_2 only of a_k's. But then x_3 and x_4 must consist only of a_1's and a_k's. It is then easy to find a pair (y, z) in $T(G)$ such that the symbols of z are out of the proper order. We conclude that G does not exist, and the theorem is proven.

IV. Pushdown Assemblers

One might well ask if there is a device which stands in relation to syntax directed translation schemata as pushdown automata do to context-free languages. In this section, we propose such a device.

A pushdown automaton (PDA) is a finite control with an input terminal at which input symbols appear when requested by the finite control. In addition, the PDA has a pushdown list. The finite control can read the top symbol of the list, and in any move can replace the top symbol by a finite length string of symbols, including the empty string. The device is nondeterministic, and may have any finite number of choices in each situation. A sequence of input symbols is accepted if some choice of moves of the PDA using that sequence of inputs causes the pushdown list to become empty. The language (set of inputs) accepted by the PDA is a context free language, and every context free language is accepted by a PDA.

We will add some features to the PDA, to enable the resulting device, called a pushdown assembler (PA), to perform any syntax directed translation. Associated with each symbol on the pushdown list will be k passive registers. Each register can be empty or hold a string of output symbols. Such a situation is shown in Fig. 4.1, where $k = 2$ and the pushdown list is CBA. Associated with A are two registers, the first empty (φ indicates an empty register) and the second holding string w. Both registers associated with B are empty, and the first register associated with C contains string x.

$$\begin{array}{ccc} C & \underline{x} & \underline{\varphi} \\ B & \underline{\varphi} & \underline{\varphi} \\ A & \underline{\varphi} & \underline{w} \end{array}$$

Fig. 4.1

Suppose C, at the top of the list, is replaced by string ED. Symbol D would replace C in Fig. 4.1, and E would appear above D with empty registers. The result appears in Fig. 4.2.

$$\begin{array}{ccc} E & \underline{\varphi} & \underline{\varphi} \\ D & \underline{x} & \underline{\varphi} \\ B & \underline{\varphi} & \underline{\varphi} \\ A & \underline{\varphi} & \underline{w} \end{array}$$

Fig. 4.2

The PA can write a finite string in any empty register at the highest level. For example, it could write y and z in the first and second registers of the top level, leaving the situation of Fig. 4.3.

$$\begin{array}{ccc} E & \underline{y} & \underline{z} \\ D & \underline{x} & \underline{\varphi} \\ B & \underline{\varphi} & \underline{\varphi} \\ A & \underline{\varphi} & \underline{w} \end{array}$$

FIG. 4.3

If the top symbol is erased, the contents of its registers are concatenated in order. If a register is empty, it is treated as though it contained ϵ. The resulting string then "waits" at the top of the list to be placed, on the next move, in an empty register. If E in Fig. 4.3 were erased, yz would be passed down to the level below and would appear temporarily to the left of D as in Fig. 4.4.

$$\begin{array}{cccc} yz & D & \underline{x} & \underline{\varphi} \\ & B & \underline{\varphi} & \underline{\varphi} \\ & A & \underline{\varphi} & \underline{w} \end{array}$$

FIG. 4.4

Next, yz is placed in an empty register at the top level. In this case, it must be the second register. If the PA tries to write into a register which is not empty, it "jams" and can make no further moves. Figure 4.4 thus should become Fig. 4.5.

$$\begin{array}{ccc} D & \underline{x} & \underline{yz} \\ B & \underline{\varphi} & \underline{\varphi} \\ A & \underline{\varphi} & \underline{w} \end{array}$$

FIG. 4.5

If next, D were erased and the string resulting from this concatenation of its registers were stored in the second register of the next lower level, the list would be (Fig. 4.6):

$$\begin{array}{ccc} B & \underline{\varphi} & \underline{xyz} \\ A & \underline{\varphi} & \underline{w} \end{array}$$

FIG. 4.6

Finally, suppose that on successive moves, B is erased, xyz stored in the first register of the next level, then A is erased. The resulting string, $xyzw$, would have no place to go, and will be deemed output of the PA; $xyzw$ would be considered a translation of whatever input string caused the sequence of moves, the terminal portion of which we have been describing.

We shall now give a formal notation incorporating the ideas we have been using. A *k-register pushdown assembler* is denoted $P = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0)$, where Q, Σ, Δ, and Γ are finite sets of *states*, *input symbols*, *output symbols* and *tape symbols*, respectively. q_0, in Q, is the *start state.* Z_0, in Γ, is the *start symbol.* λ, μ and ν are mappings which indicate the allowable moves of P. λ controls changes of the pushdown symbols (not register contents) on the pushdown list. μ controls the entry of finite length strings into registers. ν controls the insertion into registers of strings which have been displaced temporarily by the erasure of the top symbol on the pushdown list (as in Fig. 4.4).

λ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times \Gamma^*$.

μ is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the finite subsets of $Q \times \Delta^* \times \{1, 2,..., k\}$.

ν is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to the subsets of $Q \times \{1, 2,..., k\}$.

A *configuration* of P is denoted (q, α), where q is in Q, and α is a string either of the form $Z_1t_1Z_2t_2 \cdots Z_mt_m$ or $[w]\, Z_1t_1Z_2t_2 \cdots Z_mt_m$, where $Z_1, Z_2,..., Z_m$ are in Γ, $t_1, t_2,..., t_m$ are k-tuples of elements in $\Delta^* \cup \{\varphi\}$ and w is in Δ^*. t_i, $1 \leqslant i \leqslant m$, represents the contents of the k registers associated with Z_i. φ denotes an empty register. The string is prefixed by $[w]$ if the string w must be stored in some register, a condition akin to that of Fig. 4.4.

For any q in Q, a in $\Sigma \cup \{\epsilon\}$ and Z in Γ, suppose $\lambda(q, a, Z)$ contains $(p, Z_1Z_2 \cdots Z_m)$, $m \geqslant 1$. Then we write $a : (q, Z\alpha) \vdash_{\overline{P}} (p, Z_1tZ_2t \cdots Z_{m-1}tZ_m\alpha)$, where t is the k-tuple $(\varphi, \varphi,..., \varphi)$. If $\lambda(q, a, Z)$ contains (p, ϵ), then we write

$$a : (q, Z(w_1, w_2,..., w_k)\, \alpha) \vdash_{\overline{P}} (p, [w_1w_2 \cdots w_k]\, \alpha).$$

Here, $w_1w_2 \cdots w_k$ is the concatenation of $w_1, w_2,..., w_k$, with φ taken to be ϵ.

Suppose $\nu(q, a, Z)$ contains (p, i). Then we may write

$$a : (q, [w]\, Z(x_1, x_2,..., x_k)\, \alpha) \vdash_{\overline{P}} (p, Z(x_1,..., x_{i-1}, w, x_{i+1},..., w_k)\, \alpha),$$

provided $x_i = \varphi$. Finally, if $\mu(q, a, Z)$ contains (p, w, i), then we may write

$$a : (q, Z(x_1, x_2,..., x_k)\, \alpha) \vdash_{\overline{P}} (p, Z(x_1,..., x_{i-1}, w, x_{i+1},..., x_k)\, \alpha),$$

again provided $x_i = \varphi$.

The symbol $\overset{*}{\underset{P}{\vdash}}$ is defined by

(1) $\epsilon : Q \overset{*}{\underset{P}{\vdash}} Q$ for any configuration Q, and

(2) for w in Σ^* and a in $\Sigma \cup \{\epsilon\}$, if $w : Q_1 \overset{*}{\underset{P}{\vdash}} Q_2$ and $a : Q_2 \underset{P}{\vdash} Q_3$, then $wa : Q_1 \overset{*}{\underset{P}{\vdash}} Q_3$. The *translation defined by P*, denoted $\tau(P)$, is

$$\{(w, x) \mid w : (q_0, Z_0(\varphi, \varphi, \ldots, \varphi)) \overset{*}{\underset{P}{\vdash}} (q, [x]) \text{ for some } q \text{ in } Q\}.$$

EXAMPLE. Let us construct a pushdown assembler that will translate infix expressions, using $\langle$ and $\rangle$ for brackets and involving a (presumably nonassociative) operation $\#$ and variable a, into equivalent prefix expressions. For example, $\langle\langle a \# a\rangle \# a\rangle$ is translated to $\#\#aaa$, and $\langle a \#\langle a \# a\rangle\rangle$ is translated to $\#a\#aa$. An SDTS for this translation is $G = (\{S\}, \{\langle, \rangle, a, \#\}, \{a, \#\}, R, S\}$, where R consists of:

$$S \to (\langle S \# S\rangle, \#SS, [1, 2]),$$
$$S \to (a, a).$$

Let P be the three-register PA $(\{q_0, q_1, q_2\}, \{\langle, \rangle, a, \#\}, \{a, \#\}, \{B_1, B_2, B_3\}, \lambda, \mu, \nu, q_0, B_1)$, where λ, μ and ν are defined by:

$$\mu(q_0, a, B_1) = \{(q_1, a, 1)\}, \tag{1}$$
$$\lambda(q_1, \epsilon, B_1) = \{(q_0, \epsilon)\}. \tag{2}$$

When B_1 is at the top of the list, P is looking for a well formed infix expression on the input. The single symbol a is well formed and its translation is a. So P places a in the first register and B_1 is erased, passing the a to the level below.

$$\lambda(q_0, \langle, B_1) = \{(q_0, B_1B_2)\}, \tag{3}$$

If P is looking for a well formed expression and $\langle$ is the next input, P changes B_1 to B_2 (B_2 indicates that P must find the second half of a well-formed expression, i.e., the portion to the right of the center $\#$.) and grows a new level with symbol B_1.

$$\nu(q_0, \epsilon, B_2) = \{(q_0, 2)\}, \tag{4}$$
$$\mu(q_0, \#, B_2) = \{(q_2, \#, 1)\}, \tag{5}$$
$$\lambda(q_2, \epsilon, B_2) = \{(q_0, B_1B_3)\}. \tag{6}$$

If B_2 is the top symbol, P stores the result of the erasure of the level above, which will be a well formed expression, in the second register. Then (rule 5), P checks that the next input symbol is #, stores it in the first register, and (rule 6) changes B_2 to B_3, growing a new level with symbol B_1.

$$\nu(q_0, \epsilon, B_3) = \{q_0, 3)\}, \tag{7}$$

$$\lambda(q_0, \rangle, B_3) = \{(q_0, \epsilon)\}. \tag{8}$$

With B_3 on top, P stores the result of the level above in the third register and, if q is the next input, erases the level, having completed a well-formed expression.

Suppose the input to P is $\langle\langle a \# a\rangle \# a\rangle$. The sequence of configurations entered by P is shown in Fig. 4.7.

Configuration	Input Used	Rule No.
$(q_0, B_1(\varphi, \varphi, \varphi))$	start	
$(q_0, B_1(\varphi, \varphi, \varphi)\, B_2(\varphi, \varphi, \varphi))$	$\langle$	3
$(q_0, B_1(\varphi, \varphi, \varphi)\, B_2(\varphi, \varphi, \varphi)\, B_2(\varphi, \varphi, \varphi))$	$\langle$	3
$(q_1, B_1(a, \varphi, \varphi)\, B_2(\varphi, \varphi, \varphi)\, B_2(\varphi, \varphi, \varphi))$	a	1
$(q_0, [a]\, B_2(\varphi, \varphi, \varphi)\, B_2(\varphi, \varphi, \varphi))$	ϵ	2
$(q_0, B_2(\varphi, a, \varphi)\, B_2(\varphi, \varphi, \varphi))$	ϵ	4
$(q_2, B_2(\#, a, \varphi)\, B_2(\varphi, \varphi, \varphi))$	#	5
$(q_0, B_1(\varphi, \varphi, \varphi)\, B_3(\#, a, \varphi)\, B_2(\varphi, \varphi, \varphi))$	ϵ	6
$(q_1, B_1(a, \varphi, \varphi)\, B_3(\#, a, \varphi)\, B_2(\varphi, \varphi, \varphi))$	a	1
$(q_0, [a]\, B_3(\#, a, \varphi)\, B_2(\varphi, \varphi, \varphi))$	ϵ	2
$(q_0, B_3(\#, a, a)\, B_2(\varphi, \varphi, \varphi))$	ϵ	7
$(q_0, [\#aa]\, B_2(\varphi, \varphi, \varphi))$	$\rangle$	8
$(q_0, B_2(\varphi, \#aa, \varphi))$	ϵ	4
$(q_2, B_2(\#, \#aa, \varphi))$	#	5
$(q_0, B_1(\varphi, \varphi, \varphi)\, B_3(\#, \#aa, \varphi))$	ϵ	6
$(q_1, B_1(a, \varphi, \varphi)\, B_3(\#, \#aa, \varphi))$	a	1
$(q_0, [a]\, B_3(\#, \#aa, \varphi))$	ϵ	2
$(q_0, B_3(\#, \#aa, a))$	ϵ	7
$(q_0, [\#\#aaa])$	$\rangle$	8

FIG. 4.7

We would like to show that for $k \geqslant 2$, a translation is defined by a k register pushdown assembler if and only if it is an SDT of order k.

THEOREM 4.1. *If a translation T is an* SDT *of order $k \geqslant 2$, then $T = \tau(P)$ for some k register pushdown assembler P.*

Proof. Let $T = T(G)$ where $G = (V, \Sigma, \Delta, R, S)$ is an SDTS of order k. Assume G is in normal form. [If $A \to (\alpha, \beta, \Pi)$ is in R, then α and β are all variables or have no variables.] We will construct the k register PA, $P = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, S)$. Q consists of the symbol q_0 and the finite set of symbols of the form q_w, where w is in Σ^* and there is a rule in R of the form $A \to (w, x)$. Γ consists of:

(a) symbols in V,

(b) symbols $[\alpha, \Pi]$, where α is in V^*, there is a rule $A \to (\gamma, \beta, \Pi)$ in R and α is suffix of γ.

(c) symbols $[w]$, where w is in Σ^* and the length of w does not exceed the longest string in Σ^* found among the rules of R.

We define λ, μ and ν by:

1. Suppose $A \to (\alpha, \beta, \Pi)$ is a rule. If α is in $V^* - \{\epsilon\}$, then

   (i) $\lambda(q_0, \epsilon, A)$ contains $(q_0, [\alpha, \Pi])$.

If α is in Σ^*, then

   (ii) $\mu(q_0, \epsilon, A)$ contains $(q_\alpha, \beta, 1)$ and

   (iii) $\lambda(q_\alpha, \epsilon, A)$ contains $(q_0, [\alpha])$.

With variable A at the top of the list, P guesses a rule using A. If that rule replaces A by variables, the list of variables and their permutation replaces A at the top of the list. If A is replaced by input and output symbols, the output symbols are placed in the first register, then the input symbols replace A on the pushdown list.

2. (i) $\lambda(q_0, a, [aw]) = \{(q_0, [w])\}$, for all a in Σ and w in Σ^*, such that $[aw]$ is in Γ.

   (ii) $\lambda(q_0, \epsilon, [\epsilon]) = \{(q_0, \epsilon)\}$.

If a symbol representing a string of input symbols is at the top of the list, they are compared with the next input symbols. If all compare, the top symbol of the pushdown list is eventually erased.

3. (i) $\lambda(q_0, \epsilon, [A\alpha, \Pi]) = \{(q_0, A[\alpha, \Pi])\}$.

   (ii) $\nu(q_0, \epsilon, [\alpha, \Pi]) = \{(q_0, i)\}$. i is equal to $\Pi(m - |\alpha|)$ if Π is a permutation on m objects.

   (iii) $\lambda(q_0, \epsilon, [\epsilon, \Pi]) = \{(q_0, \epsilon)\}$.

If the top symbol is a pair of a variable string and a permutation, the first variable on the list is placed on the next higher level. When the output of the higher level is passed down, it is stored in the register dictated by the permutation.

We shall now prove that $w : (q_0, A(\varphi, \varphi, ..., \varphi)) \overset{*}{\underset{P}{\vdash}} (q_0, [x])$ if and only if $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (w, x)$, for any variable A. From this result, it immediately follows that $\tau(P) = T(G)$.

If: We will prove the result by induction on the number of steps used in the derivation of (w, x) from (A, A). The one step case follows directly from an application of rules 1(ii), 1(iii), 2(i) as many times as needed, and 2(ii).

Assume the result for less than k steps, $k \geqslant 2$. Then the first step in a k step derivation must be of the form

$$(A, A) \underset{G}{\Rightarrow} (A_1 A_2 \cdots A_m, A_{\hat{\Pi}(1)} A_{\hat{\Pi}(2)} \cdots A_{\hat{\Pi}(m)}, \Pi).$$

By rule 1(i), we know that $\epsilon : (q_0, A(\varphi, \varphi, ..., \varphi)) \vdash_{\overline{P}} (q_0, [A_1 \cdots A_m, \Pi](\varphi, ..., \varphi))$. We can write $w = w_1 w_2 \cdots w_m$ and $x = x_1 x_2 \cdots x_m$, such that $(A_i, A_i) \overset{*}{\underset{G}{\Rightarrow}} (w_i, x_{\Pi(i)})$ for each i. Thus, by the inductive hypothesis,

$$(\#)\ w_i : (q_0, A_i(\varphi, \varphi, ..., \varphi)) \overset{*}{\vdash_{\overline{P}}} (q_0, [x_{\Pi(i)}]).$$

Using rule 3(i), relation (#) and rule 3(ii), m times, then rule 3(iii), one can easily put together a sequence of moves of P which demonstrate that

$$w : (q_0, A(\varphi, \varphi, ..., \varphi)) \overset{*}{\vdash_{\overline{P}}} (q_0, [x]).$$

Only if: We will prove by induction on the number of moves made by P that if $w : (q_0, A(\varphi, \varphi, ..., \varphi) \overset{*}{\vdash_{\overline{P}}} (q_0, [x])$, then $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (w, x)$. The result is true vacuously for fewer than three moves. Suppose it is true for less than k moves. For a sequence of k moves, the first move must be due either to rule 1(i) or 1(ii). In the latter case, the subsequent moves of P are completely determined by rules 1(iii), 2(i) and 2(ii). Moreover, by 1(ii), there is a rule $A \rightarrow (w, x)$ in R. The result we desire follows in this case without reference to the inductive hypothesis. If the first move is due to 1(i), we can express the subsequent operation of P as

$$\epsilon : (q_0, [A_1 A_2 \cdots A_m, \Pi]\, t_0) \vdash_{\overline{P}} (q_0, A_1 t_0 [A_2 \cdots A_m, \Pi]\, t_0)$$

$$w_1 : (q_0, A_1 t_0 [A_2 \cdots A_m, \Pi]\, t_0) \overset{*}{\vdash_{\overline{P}}} (q_0, [x_1]\, [A_2 \cdots A_m, \Pi]\, t_0)$$

$$\epsilon : (q_0, [x_1]\, [A_2 \cdots A_m, \Pi]\, t_0) \vdash_{\overline{P}} (q_0, [A_2 \cdots A_m, \Pi]\, t_1)$$

$$\epsilon : (q_0, [A_2 \cdots A_m, \Pi]\, t_1) \vdash_{\overline{P}} (q_0, A_2 t_0 [A_3 \cdots A_m, \Pi]\, t_1)$$

$$\vdots$$

$$w_m : (q_0, A_m t_0 [\epsilon, \Pi]\, t_{m-1}) \overset{*}{\vdash_{\overline{P}}} (q_0, [x_m]\, [\epsilon, \Pi]\, t_{m-1})$$

$$\epsilon : (q_0, [x_m]\, [\epsilon, \Pi]\, t_{m-1}) \vdash_{\overline{P}} (q\ , [\epsilon, \Pi]\, t_m)$$

$$\epsilon : (q_0, [\epsilon, \Pi]\, t_m) \vdash_{\overline{P}} (q_0, [x]).$$

Here $t_0 = (\varphi, \varphi, ..., \varphi)$ and t_i is t_{i-1} with x_i replacing φ in the $\Pi(i)$th position.

The initial move of P implies that R has a rule

$$A \to (A_1A_2 \cdots A_m, A_{\hat{\Pi}(1)}A_{\hat{\Pi}(2)} \cdots A_{\hat{\Pi}(m)}, \Pi).$$

By the inductive hypothesis, the above sequence of moves of P implies that $(A_i, A_i) \overset{*}{\underset{G}{\Rightarrow}} (w_i, x_i)$ for all i. Putting the above together, we can easily show that $(A, A) \overset{*}{\underset{G}{\Rightarrow}} (w, x)$.

To prove the converse of Theorem 4.1 we will need two auxiliary lemmas.

LEMMA 4.1. *Every translation defined by some k register* PA *P' is defined by a k register* PA, *$P = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0)$, for which, if $\lambda(q, a, Z)$ contains (p, α), then $|\alpha| \leqslant 2$.*

Proof. This is a generalization of a simple construction on pushdown automata. Let $P' = (Q', \Sigma, \Delta, \Gamma, \lambda', \mu, \nu, q_0, Z_0)$. For each $(p, Z_1Z_2 \cdots Z_m)$ in $\lambda'(q, a, Z)$ with $m > 2$, introduce new states $q_1, q_2, \ldots, q_{m-2}$ to Q. Remove $(p, Z_1Z_2 \cdots Z_m)$ from $\lambda'(q, a, Z)$ and replace it by $(q_1, Z_{m-1}Z_m)$. Define

$$\lambda(q_i, \epsilon, Z_{m-i}) = \{(q_{i+1}, Z_{m-i-1}Z_{m-i})\} \qquad \text{for} \qquad i = 1, 2, \ldots, m-3$$

and

$$\lambda(q_{m-2}, \epsilon, Z_2) = \{(p, Z_1Z_2)\}.$$

After making all such replacements, Q is the resulting set of states and λ the result of making these alterations in λ'.

LEMMA 4.2. *Every translation defined by a k register* PA *P' is defined by a k register* PA *P, which satisfies Lemma 4.1 and has the additional property that if it erases an entry on its pushdown list, then that entry has no empty registers.*

Proof. Let $P' = (Q', \Sigma, \Delta, \Gamma', \lambda', \mu', \nu', q_0, Z_0)$, and assume P' satisfies Lemma 4.1. P will simulate P', but in addition, in the control symbol of each entry on the pushdown list, P will keep track of which registers of that entry are empty. If P' erases an entry, P first stores ϵ in each empty register. We will give a construction which incorporates these ideas.

Formally, let $P = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, [Z_0, \varphi])$. Let $K = \{1, 2, \ldots, k\}$ and $\Gamma = \{[X, S] \mid X \text{ is in } \Gamma' \text{ and } S \subseteq K\}$. Q contains the states of Q' and some new states which are introduced through the definitions of λ, μ and ν, below. For all q and p in Q', a in $\Sigma \cup \{\epsilon\}$, X, Y and Z in Γ' and $S \subseteq K$:

1. If $\lambda'(q, a, Z)$ contains (p, Y) then $\lambda(q, a, [Z, S])$ contains $(p, [Y, S])$. If $\lambda'(q, a, Z)$ contains (p, XY) then $\lambda(q, a, [Z, S])$ contains $(p, [X, \varphi]\,[Y, S])$. ($P$ keeps track of full registers at each level when manipulating the pushdown list.)

2. If $\lambda'(q, a, Z)$ contains (p, ϵ) and $S \neq K$, let $\{i_1, i_2, \ldots, i_m\}$ be $K - S$. For this S, introduce to Q new states $q_1, q_2, \ldots, q_m$. Let $\mu(q, a, [Z, S])$ contain (q_1, ϵ, i_1), $\mu(q_1, \epsilon, [Z, S])$ contain $(q_2, \epsilon, i_2), \ldots, \mu(q_{m-1}, \epsilon, [Z, S])$ contain (q_m, ϵ, i_m) and $\lambda(q_m, \epsilon, [Z, S])$ contain (p, ϵ). If $S = K$, let $\lambda(q, a, [Z, S])$ contain (p, ϵ). (If P' would erase an entry that has empty registers, these registers are filled with ϵ before erasing.)

3. If i is not in S and $\mu'(q, a, Z)$ contains (p, w, i), introduce to Q a new state q_1. Let $\lambda(q, a, [Z, S])$ contain $(q_1, [Z, S \cup \{i\}])$ and $\mu(q_1, \epsilon, [Z, S \cup \{i\}])$ contain (p, w, i). (If P' stores an output string, P updates the set of full registers.)

4. If i is not in S and $\nu'(q, a, Z)$ contains (p, i), introduce to Q a new state q_1. Let $\nu(q, a, [Z, S])$ contain (q_1, i) and $\lambda(q_1, \epsilon, [Z, S])$ contain $(p, [Z, S \cup \{i\}])$. (When P' stores the result of the erasure of the entry above, P' also updates the set of full registers.)

THEOREM 4.2. *If* $T = \tau(P)$ *for a* k *register* PA $P = (Q, \Sigma, \Delta, \Gamma, \lambda, \mu, \nu, q_0, Z_0)$, *then* $T = T(G)$ *for an* SDTS $G = (V, \Sigma, \Delta, R, S)$ *of order* k.

Proof. Assume P satisfies Lemmas 4.1 and 4.2. V will consist of:

1. The symbol S,
2. a symbol $[q, Z, p]$ for each q and p in Q and Z in Γ, and
3. a symbol $\langle q, Y, p, Z, i \rangle$ for each q and p in Q, Y and Z in Γ, and integer $i \leqslant k$.

Intuitively, we wish the symbol $[q, Z, p]$ to generate (x, y)

$$[\text{i.e., } ([q, Z, p], [q, Z, p]) \overset{*}{\underset{G}{\Rightarrow}} (x, y).]$$

exactly when $x : (q, Z(\varphi, \varphi, \ldots, \varphi)) \overset{*}{\underset{P}{\vdash}} (p, [y])$. We also want $\langle q, Y, p, Z, i \rangle$ to generate (x, y) when $x : (q, Yt) \overset{*}{\underset{P}{\vdash}} (p, Zt')$, where t' is t with φ replaced by y in the ith component.

The symbols $[q, Z, p]$ and $\langle q, Y, p, Z, i \rangle$ can be thought of as representing "events" in the history of computations of P. We wish to specify the rules of G in such a manner that the relation between the events faithfully describes the actions allowed to P. These rules are:

1. $S \to ([q_0, Z_0, p], [q_0, Z_0, p])$, for each p in Q. G is to generate those (x, y) such that

$$x : (q_0, Z_0(\varphi, \varphi, \ldots, \varphi)) \overset{*}{\underset{P}{\vdash}} (p, [y]) \qquad \text{for some} \quad p \ \text{ in } \ Q.$$

2. Suppose Π is a permutation on k objects. If for $q_1, q_2, \ldots, q_{k+1}, p$ in Q, $Z_1, Z_2, \ldots, Z_{k+1}$ in Γ, and a in $\Sigma \cup \{\epsilon\}$, $\lambda(q_{k+1}, a, Z_{k+1})$ contains (p, ϵ), then

$$[q_1, Z_1, p] \rightarrow (A_1 A_2 \cdots A_k a, A_{\Pi(1)} A_{\Pi(2)} \cdots A_{\Pi(k)}, \Pi),$$

where A_i, $1 \leqslant i \leqslant k$, is the symbol $\langle q_i, Z_i, q_{i+1}, Z_{i+1}, \Pi(i)\rangle$. These rules express the ways in which an entry can be erased from the pushdown list after its registers are filled in some order.

3. If $\lambda(q, a, Z)$ contains (r, Y), then R contains

$$\langle q, Z, p, X, i\rangle \rightarrow (a\langle r, Y, p, X, i\rangle, \langle r, Y, p, X, i\rangle)$$

and

$$\langle p, X, r, Y, i\rangle \rightarrow (\langle p, X, q, Z, i\rangle\, a, \langle p, X, q, Z, i\rangle),$$

for each p, q, r in Q, X, Y, Z in Γ, a in $\Sigma \cup \{\epsilon\}$ and integer i. These rules represent moves which do not involve registers or alteration of the length of the pushdown list.

4. If $\lambda(q, a, Z)$ contains (p, XY) and $\nu(r, b, Y)$ contains (s, i), then R contains $\langle q, Z, s, Y, i\rangle \rightarrow (a[p, X, r]\, b, [p, X, r])$, for all p, q, r, s in Q, X, Y, Z in Γ, a and b in $\Sigma \cup \{\epsilon\}$ and integer i. These rules represent the situation in which P increases the length of its pushdown list and, when the new level is erased, stores the result in the ith register.

5. If $\mu(q, a, Z)$ contains (p, w, i), then R contains $\langle q, Z, p, Z, i\rangle \rightarrow (a, w)$ for any p and q in Q, a in $\Sigma \cup \{\epsilon\}$, w in Δ^* and integer i.

We can prove that:

a. $([q, Z, p], [q, Z, p]) \overset{*}{\underset{G}{\Rightarrow}} (x, y)$ if and only if $x : (q, Z(\varphi, \varphi, \ldots, \varphi)) \overset{*}{\underset{P}{\vdash}} (p, [y])$

b. $(\langle q, Z, p, Y, i\rangle, \langle q, Z, p, Y, i\rangle) \overset{*}{\Rightarrow} (x, y)$ if and only if $x : (q, Zt) \overset{*}{\underset{P}{\vdash}} (p, Yt')$, where the ith component of t is φ, and t' is t with y in the ith register.

The proof proceeds by induction on the number of moves of P (in the "if" direction) or steps in a derivation in G (in the "only if" direction). It is a straightforward application of the definition of G and will be omitted. From rule (1) and statement (a) above, it follows that $\tau(P) = T(G)$.

V. Conclusions

We have investigated the class of syntax-directed translations and shown the existence of an infinite hierarchy of these in terms of the number of variables allowed in the right side of a rule. This number is called the order of the translation. While three variables are no better than two, any other increase in the order results in an increase in the set of translations definable.

We have also defined a device called the pushdown assembler. This device is essentially a pushdown automaton with storage registers associated with each entry on the pushdown list. When an entry is erased from the top of the list, the contents of its registers are passed to the entry below. A pushdown assembler with k registers, $k \geqslant 2$, at each level was shown to define exactly the syntax directed translations of order k.

References

1. E. Irons. A syntax directed compiler for ALGOL-60, *CACM*, **4**, 51–55 (1961).
2. K. Culik. Well-translatable languages and ALGOL-like languages, *in "Formal Language Description Languages,"* (T. Steele, Ed.) pp. 75–85, North Holland, Amsterdam, 1966.
3. P. M. Lewis II and R. E. Stearns. Syntax directed transduction, *JACM* **15**, 465–488 (1968).
4. M. Paull. Bilateral descriptions of syntactic mappings. *In* "First Annal Princeton Conference on Information Sciences and Systems," 76–81. Princeton University Press, Princeton, New Jersey, 1967.
5. D. Younger, Context free language processing in time n^3. *IEEE Conf. Record 7th Annl. Symp. Switching and Automata Theory*, 7–20 (1966). Also, Recognition and parsing of context free languages in time n^3. *Inform. Control*, **10**, 189–208 (1967).
6. A. V. Aho and J. D. Ullman. Properties of syntax directed translations (to appear in *J. Comp. System Science*).
7. N. Chomsky. On certain formal properties of grammars. *Inform. Control*, **2**, 137–167 (1959).
8. J. E. Hopcroft and J. D. Ullman. "Formal Languages and Their Relation to Automata." Addison Wesley, Reading Massachusetts (1969).