1975

Optimal Compression of Parsing Tables in a Parsergenerating System

V. B. Schneider

M. D. Mickunas

Report Number:
75-150

OPTIMAL COMPRESSION OF PARSING TABLES IN A PARSER-GENERATING SYSTEM

V. B. Schneider

and

M. D. Mickunas

Computer Sciences Department
Purdue University
West Lafayette, Indiana 47907

CSD-TR 150

OPTIMAL COMPRESSION OF PARSING TABLES IN A PARSER-GENERATING SYSTEM

V. B. Schneider
M. D. Mickunas

Computer Sciences Department
Purdue University
West Lafayette, Indiana
47907

Key Words and Phrases: parser generators, compression algorithm, syntactic analysis, normal-form grammars, bounded-context acceptors, table-driven parsers

CR Categories: 4.12, 5.22, 5.23

This research was supported in part by NSF Grants GJ 851 and GJ 31572.

Address for Correspondence: V. B. Schneider
Computer Sciences Department
Purdue University
West Lafayette, Indiana
47907

ABSTRACT

This paper describes the optimal table-compression algorithm of a parser-generating system for implementing compilers currently in use at Purdue University. A proof is given of the minimality of the parsing tables generated by the algorithm, and figures are presented that demonstrate the superiority of our system in terms of the small overall size of parsers generated by it. In addition, our comparisons show that compiling speeds obtainable with our system compare favorably with the fastest "one-pass" compilers available for the CDC-6500 computer on which the system is implemented.

INTRODUCTION

An earlier paper[15] described a parser generating system (PGS) that generates compact, efficient programming language parsers. The two inadequacies of this PGS, namely, that it could only handle (1,2)bounded right context grammars and that the parser compression algorithm was not optimal, have since been overcome. The new version of the PGS now accepts LR(k) grammars and transforms then internally to (1,1) bounded right context grammars that "cover" the original LR(k) grammar [5] (i.e., parses of the transformed grammar cause the same sequence of code generator calls as those using the original LR(k) grammar). This transformation to (1,1)BRC grammars from LR(k) has been described elsewhere [13,14], and some of the results reported in these papers will be used in what follows. The main thrust of this paper is the description of the improved compression algorithm, together with a proof of its optimality under certain constraints.

Before undertaking a description of the parser compression algorithm, some informal explanation of how the PGS parsers operate is first in order. PGS parsers parse in a single pass, moving from left to right across the source program under analysis. During a given step in a parse, either one source program symbol is consumed or one symbol from the top of the parsing stack. Each step in the parse essentially corresponds to a syntactic reduction in which the right-hand side of a "normal-form" rule is replaced by the corresponding left-hand side nonterminal. In this sense, the parsing algorithm resembles the bounded right context parsers of Samelson and Bauer [16] rather than the precedence parsers of Wirth and others[5,18] .

In a typical PGS parser, the input LR(k) grammar has been transformed into a (1,1)BRC normal-form grammar (1NFG) having rules of the following four types:

(1) $A_i \rightarrow a_i$ (3) $A_k \rightarrow A_{k1}a_{k2}$

(2) $A_j \rightarrow A_{j1}$ (4) $A_m \rightarrow A_{m1}A_{m2}$

where capital letters are nonterminal and small letters are terminals. With such a normal-form grammar, each entry in the

parser's table consists of a single rule that specifies a reduction of either a source program or a parsing stack symbol. These 1NFG rules have some interesting properties:

(1) As is easily deduced, type (1) rules correspond to the leftmost reduction of a parse subtree. These rules are all grouped together into an "initial state" of the parser. This initial state is where the parse begins and where the parse continues after completing a subtree of its parse.

(2) The nonterminals A_{k1} in type (3) rules and A_{m2} in type (4) rules are nonterminals that merge immediately with an adjacent symbol in some reduction in a left-to-right parse These nonterminals are members of the "state set" of the parser. That is, each A_{k1} in the state set corresponds to a place in the parser's tables that directs a syntactic reduction with a_{k2} as the currently scanned source program symbol. Each A_{m2} in the state set likewise corresponds to a place in the parser's tables that directs a syntactic reduction with A_{m1} as the symbol on top of the parsing stack.

(3) The type (2) rules (which we also call "renaming rules") exist to reflect the presence of such renaming rules in the operator precedence grammars for languages like Algol 60, Algol W, etc. There has been considerable partisan debate on the necessity of renaming rules, and it is our experience that elimination of renaming rules from programming language grammars only serves to enlarge the grammars that result from their absence, while eliminating a natural, intuitive method of describing the structure of expressions.

(4). The A_{m1} nonterminals in type (4) rules are the roots of subtrees in a parse that are ultimately joined to the A_{m2} subtrees in reductions involving type (4) rules. These A_{m1} nonterminals thus comprise the "stack-symbol" set of the parser.

During a parse, when a 1NFG reduction results in a symbol of the stack set, that symbol is stored on top of the parse stack, and the parse continues in the initial state. When the reduction results in a symbol of the state set, the parse continues at that point in the parsing table corresponding to the reductions possible

for the rules in that state. Parsing ends on reduction of the program to the initial symbol of the grammar.

From this description, we can see that the parsing table entries specify both the next symbol to be scanned and the syntactic reduction to be made. Thus, our PGS parsers require only one parsing table, by contrast to the two tables used in more traditional precedence-matrix parsers [5,18]. Also, it can be easily seen that the lower bound on parsing table size for the PGS is simply the number of normal-form rules in the corresponding grammar. Moreover, each such parsing table can be implemented with 24-bit entries, yielding typical Algol parsing tables of under 1200 bytes linked to driver programs expressed in 264 bytes of machine code instructions [15]. Unfortunately, the minimum table size is unattainable except for simple grammars. A more attainable table size, such as for the Algol parser given above, is closer to 50% above the minimum. The focus of our compression algorithm, then, will be the state-by-state minimization in the number of table entries necessary to implement the parser, subject to certain constraints to avoid combinatoric problems.

As a simple example of the compression process, consider the initial state of some parser, consisting of all the type (1) rules in the grammar. If each terminal seen by the initial state corresponds to exactly one rule, then the compressed initial state consists of one entry for each rule. Now, suppose there exists a subset of rules in the initial state, all having the same right-hand side. I.e., rules

$A_{ij} \rightarrow a'$ such that $|\{A_{ij} \rightarrow a'\}| > 1$.

We can transform the grammar to one in which

$A_{ij} \rightarrow [a']$ with $[a']$ an invented nonterminal

and $[a'] \rightarrow a'$ a rule of the initial state. Now, the compressed initial state has exactly one entry for each terminal symbol scanned, and the resolution of which rule to apply in a reduction of a' has been transferred to state $[a']$ involving renaming rules.

As a second example, consider a state A_k composed exclusively of type (3) rules of the form

$P_{kj} = A_{kj} \rightarrow A_k a_{kj}$ for $j = 1,\ldots,n_k$.

If all the P_{kj} have distinct right-hand sides, then the compressed table for state A_k has exactly n_k entries, one for each distinct rule. Now, suppose that there exists a subset of A_k rules, all having the same right-hand side, i.e., rules

$$A_{kj} \rightarrow A_k a' \quad \text{such that } |\{A_{kj} \rightarrow a'\}| > 1 .$$

We can transform the grammar to one in which

$$A_{kj} \rightarrow [A_k a'] \quad , \text{ with } [A_k a'] \text{ an invented nonterminal}$$

and

$$[A_k a'] \rightarrow A_k a' \quad \text{a rule of state } A_k .$$

Now, the compressed A_k state has exactly one entry for each terminal symbol scanned, and the resolution of which rule to apply in a reduction of $A_k a'$ has been transferred to a state $[A_k a']$ involving renaming rules.

It is immediately obvious that the above analysis applies also to a state A_m composed exclusively of type (4) rules of the form

$$P_{mj} = A'_{mj} \rightarrow A_{mj} A_m \quad \text{for } j = 1,\ldots,n_m .$$

where, if all rules have distinct right-hand sides, n_m is the number of entries to express the reductions of state A_m, and, otherwise, the grammar can be transformed to yield unique right-hend sides. Difficulty occurs in our compression algorithm when a state having some arbitrary mixture of type (2), type (3), and type (4) rules is encountered. (Type (1) rules always occur together in the initial state, and the compression algorithm essentially treats them as described in the example above.)

In our system, a state composed of a mixture of type (2), type (3), and type (4) rules is depicted as one quadrant of a cartesian plane involving y-coordinates that represent terminal symbols and x-coordinates that represent nonterminal (stack) symbols. As an example, consider the following hypothetical state constructed from (1,1) BRC 1NFG rules:

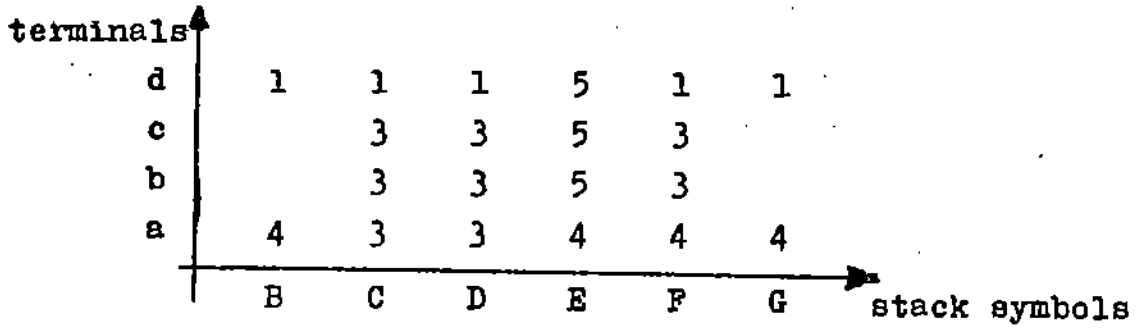


Figure 1. Contexts and Rules for State A of a Parser

In Figure 1, rule 5 must obviously be a type (4) rule, rules 1 and 4 must be type (3) rules, and rule 3 must be a type (2) rule. If we look at state A of the compressed parser, then the following entries will be used to represent the actions of the parser:

A: if B on stack then transfer to substate B' ;
if d on input then apply rule 1 ;
if a on input then transfer to substate a' ;
if anything else then apply rule 3 ;
B': if a on input then apply rule 4 ;
if anything else then apply rule 5 ;
a': if C on stack then apply rule 3 ;
if D on stack then apply rule 3 ;
if anything else then apply rule 4 ;

The compressed state A thus contains 9 twenty-four bit entries, representing a total of 20 possible symbol pairs seen as contexts in the original uncompressed version of the state.

In what follows, we will introduce definitions of the terms used in the preceding discussion, and give a proof of the optimality of the compression algorithm under certain constraints.

NOTATION AND DEFINITIONS

Let L(G) be a context-free language defined over a source-program vocabulary T and generated by a context-free grammar G = (N,T,S,P) using nonterminal alphabet N, starting symbol S in N, productions P of the form $A \rightarrow w$ in P, where A in N is the left-hand side and w in $(N \cup T)^+$ is the right-hand side, or handle, of the rule. Let $\overset{*}{\Rightarrow}$ represent $(\Rightarrow)^*$, where $w_1 \Rightarrow w_2$ if and only if w_2 can be generated in one step from w_1 using a production in grammar G. Then, $L(G) = \{x: x \text{ in } T^+ \ \& \ S \overset{*}{\Rightarrow} x\}$.

At this point, we assume that the reader has some familiarity with the standard notation and definitions used in papers on formal languages, such as [1], [6], and [12]. We are particularly concerned with the notion of (m,n) BRC (bounded right context) grammars originally developed in [4]. As is demonstrated in [13] and [14], there is an effective, localized test for the non-(m,n) BRC property of any bracketed grammar, and, if the grammar is LR(k) for some finite k, there is an effective algorithm for converting that grammar to a (1,1) BRC covering grammar. From this (1,1) BRC grammar, there is a simple conversion into our 1NFG that preserves the (1,1) BRC property and retains all structural information in the original grammar. We give the following special definition of a 1NFG, followed by a simplified conversion algorithm:

Definition A 1NFG (normal-form grammar) G is a 5-tuple

G = (N,Q,T,S,P)

where N is a finite set of symbols called the stack alphabet

Q is a finite set of symbols called the states

T is a finite set of symbols called the terminals or input alphabet

S is a distinguished element of Q called the initial symbol

P is a finite set of productions of the following forms:

Type (1): $A_i \rightarrow a_i$ with A_i in (N U Q) and a_i in T

Type (2): $A_j \rightarrow A_{jl}$ with A_j in (N U Q) and A_{jl} in Q

Type (3): $A_k \rightarrow A_{kl}a_k$ with A_k in (N U Q) , A_{kl} in Q , and a_k in T

Type (4): $A_m \rightarrow A_{m1}A_{m2}$ with A_m in (N U Q), A_{m1} in N, and A_{m2} in Q.

We further require that, for any two rules of types (1), (3), and (4), if

$$P_i = A_i \rightarrow w_i \text{ and } P_j = A_j \rightarrow w_j$$

then $w_i = w_j$ implies that $P_i = P_j$. Only type (2) rules (the "renaming rules") are allowed to violate this restriction, which, however, entails no loss in generality.

Given this definition of a 1NFG, we can easily present an algorithm for converting any (1,1) BRC grammar G = (N,T,S,P) into an equivalent (1,1) BRC 1NFG G' = (N',Q,T,S,P') such that G' covers G:

Algorithm 1: The notation [x] denotes a new symbol in N" = (N' U Q) where x is in $V^+ = (N \text{ U } T)^+$ in what follows. Let

$N'' = \{[x]: A \rightarrow xy \text{ is in P for x in } (V*V^2 \text{ U T}) \text{ \& y in } V^+\} \text{ U N}$

$P_1 = \{A \rightarrow B : A \rightarrow B \text{ is in P \& B is in V}\}$

$P_2 = \{A \rightarrow BC : A \rightarrow BC \text{ is in P \& B is in N \& C is in V}\}$

$P_3 = \{A \rightarrow [x]B : A \rightarrow xB \text{ is in P \& x is in } (V*V^2 \text{ U T}) \text{ \& B is in V}\}$

$P_4 = \{[xA] \rightarrow [x]A : A \text{ is in V \& x is in } (V*V^2 \text{ U T}) \text{ \& } [xA] \text{ is in N"}\}$

$P_5 = \{[BC] \rightarrow BC : B \text{ is in N \& C is in V}\}$

$P_6 = \{[a] \rightarrow a : a \text{ is in T \& } ([aA] \rightarrow [a]A \text{ is in } P_4 \text{ or } A \rightarrow [a]B \text{ is in } P_3)\}$

Then, $P' = P_1 \text{ U } P_2 \text{ U } P_3 \text{ U } P_4 \text{ U } P_5 \text{ U } P_6$,

$$N' = \{B : A \to BC \text{ is in } P_2 \text{ or } [BC] \to BC \text{ is in } P_5\}$$
$$\cup \{[x] : A \to [x]B \text{ is in } P_3 \text{ or } [xA] \to [x]A \text{ is in } P_4\}$$
$$Q = \{C : (A \to BC \text{ is in } P_2 \text{ or } A \to [x]C \text{ is in } P_3 \text{ or } [xC] \to [x]C \text{ is in } P_4 \text{ or } ([BC] \to BC \text{ is in } P_5 \ \& \ C \text{ is in } N)\}$$

To ensure that the stack and state alphabets are distinct in the resulting 1NFG, we transform the 1NFG as follows: For every J in $N' \cap Q$, replace the occurrences of J in the $A_i \to JB_i$ rules by [J], yielding rules $A_i \to [J]B_i$ and the additional rule $[J] \to J$ in P'. This transformation makes [J] a stack symbol and J the state symbol of the resulting 1NFG, and preserves the covering property of the grammar. It is proved in [13] and [14] that Algorithm 1 can be augmented to prevent any increase in context bounds during the conversion process, yielding a (1,1) BRC 1NFG from a (1,1) BRC context-free grammar. As a last step in the conversion, to ensure unique right-hand sides for all type (1), (3), and (4) rules in the resulting 1NFG, if there exist any subset of type (1), (3), or (4) rules in P' such that

$$P_{ik} = A_{ik} \to w \quad \text{for} \quad k = 1,\ldots,n$$

we replace these rules in P' by

$$P_{ik} = A_{ik} \to [w] \quad \text{for} \quad k = 1,\ldots,n$$

and add the rule $[w] \to w$ to P'. ([w] is clearly in Q.) The resulting 1NFG has the following interesting properties exploited by the compression algorithm (and proved in 13):

Theorem 1 ([13]) Let G be a (1,1) BRC 1NFG. Then, during a leftmost parse, at most two symbols must be inspected to determine the correct reduction for each step of the parse. I.e.,

1. The handle of each type (1) rule can be distinguished from the remaining rules of the grammar by inspection of left contexts only.
2. The handle of each type (3) rule can be distinguished from the remaining rules of the grammar by inspection of left contexts (topmost stack symbols) only.
3. The handle of each type (4) rule can be distinguished

from the remaining rules of the grammar by inspection of right contexts (current input symbols) only.

4. The handle of each type (2) rule can be distinguished from the remaining rules of the grammar by inspection of both left and right contexts.

References [15], [16], and [17] provide simple algorithms for calculating the pairs of context symbols (A,a) in N x T for which each rule in a (1,1) BRC 1NFG may be applied during a leftmost parse.

We can now indicate the natural correspondence between (1,1) BRC 1NFG's and their leftmost parsers (which we call (1,1) BCA's).

Definition A deterministic (1,1) BCA is a 6-tuple

$A = (N,Q,T,M,S_0,S)$

where

N is a finite set of symbols called the stack vocabulary

Q is a finite set of symbols called the states of the acceptor

T is a finite set of symbols called the input vocabulary or terminal vocabulary

M is a partial function from N x Q x T into (N U Q) x {0,1,2}

S_0 is a distinguished element of Q called the initial state of the acceptor

S is a distinguished element of Q called the final state of the acceptor, in which acceptance of the input string occurs.

The M-function specifies the actions taken by the BCA from a configuration (B,A,a) in N x Q x T:

$M(B,A,a) = (C,i)$

specifies

1. erase no symbols if i = 0 or
2. erase the next input symbol (namely, a) if i = 1 or
3. erase the topmost stack symbol (namely, B) if i = 2,

and then

1. stack the symbol C and continue operation in the initial state, if C is in N, or
2. continue operation in the state C, if C is in Q.

Given a 1NFG $G = (N', Q', T', S', P')$, the corresponding (1,1) BCA is defined as follows:

1. Let $N = N'$, $Q = Q'$, $T = T'$, and $S = S'$.
2. Let S_0 be defined uniquely (not in $N \cup Q \cup T$) and include it in Q.
3. Define M by the following four cases:
   a. Type (1) rules: If $P_i \rightarrow a_{i1}$ with a context pair (B, a_{i1}), then
      $M(B, S_0, a_{i1}) = (A_i, 1)$
   b. Type(2) rules: If $P_j = A_j \rightarrow A_{j1}$ with a context pair (B, b), then
      $M(B, A_{j1}, b) = (A_j, 0)$
   c. Type (3) rules: If $P_k = A_k \rightarrow A_{k1}a_{k2}$ with a context pair (B, a_{k2}), then
      $M(B, A_{k1}, a_{k2}) = (A_k, 1)$
   d. Type (4) rules: If $P_m = A_m \rightarrow A_{m1}A_{m2}$ with a context pair (A_{m1}, b), then
      $M(A_{m1}, A_{m2}, b) = (A_m, 2)$

With the M-function defined in this way, it can be easily proved that the language defined by G is the same as that accepted by A (acceptance by final state S). It also follows that A is deterministic, since, for each effective application of a rule of P, A examines both the handle and sufficient bounded context so that the rule may be uniquely distinguished.

MINIMIZING THE NUMBER OF TESTS TO SPECIFY A REDUCTION

The implemented BCA computes M by a linear search through the region of the parsing table that contains the contexts and reductions of the current state in a leftmost parse. There are $|Q|$ subtables of the M-table, counting one subtable for each BCA state and an initial state containing all the type (1) rules. Consider the subtable of M for a particular state A of the BCA (which we call the M^A table). This table contains entries of the form

$$M^A(B,a) = (A_i, j).$$

As in Figure 1, these M^A entries can be represented as a quadrant in a cartesian plane, with y coordinates representing terminal symbols and x coordinates representing stack symbols. At most one 1NFG rule is associated with each point in the plane because of the (1,1) BRC property of the grammar, and typical 1NFG rules appear as a row of contexts (type (3) rules) or a column of contexts (type (4) rules) or a blob of contexts (type (2) rules).

If we design the parser to inspect all rule contexts in sequence for each state, we obtain a very bulky parsing table, typically 5000 or more entries for a programming language. The better strategy is to observe that, in many rows, the entire set of contexts belong to one type (3) rule and, in many columns, the entire set of contexts belong to one type (4) rule. For one of these type (3) rules, we would like to ignore all tests of stack contexts with the understanding that such tests will occur anyway at a later point in the parse when those stack symbols participate in type (4) reductions. We might then use one entry of the form

$$M^A(*,a_1) = (A_1,1) \qquad \text{(a } \underline{\text{right}} \text{ } \underline{\text{entry}}\text{)}$$

for row a_1 having no conflicting type (2) or type (4) contexts, where the asterisk (*) denotes ignoring a context symbol. We could likewise use the entry

$$M^A(B_1,*) = (A_2,2) \qquad \text{(a } \underline{\text{left}} \text{ } \underline{\text{entry}}\text{)}$$

in place of the column of contexts corresponding to stack symbol

$\bar{M}^A(*,*) = (A_p, 0)$

Plausible as this outline of a compression scheme may appear, it ignores a number of problems:

1. In certain cases, generation of a minimum number of table entries to represent the context tests of each row and column does not minimize the total number of entries generated for the resulting compressed state.
2. The presence of two or more type (2) rules in state A is not adequately handled by this simple scheme.
3. It is not clear which ordering of rules to be inspected in state A yields the smallest compressed subtable.

Because of the combinatoric problems involved in finding the smallest compressed $\bar{M}^A$ subtable over all possible sequences of input and stack symbol tests, it was decided to impose some constraints on the problem to make it more tractable. Surprisingly, two very simple constraints suffice:

Constraint 1: Each $\bar{M}^A$ subtable will be generated by a process that compresses left entries first, and then compresses right entries on the basis of prior decisions.

Constraint 2: In order to preserve the language accepted by the compressed BCA, M^A entries due to stack-reducing rules (type (4) rules) must be compressed as left entries or entries in the substates of right or left entries. M^A entries due to input-reducing rules must be compressed as right entries or entries in the substates of right or left entries. Furthermore, the "don't care" entry of a state or its substate must be preceded by all the entries that test for its conflicting contexts.

Constraints 1 and 2 imply that each test in the compressed $\bar{M}^A$ table inspects only one symbol. Under these constraints, we can outline the steps involved in the operation of the optimal compression algorithm.

Compression Scheme: The optimal compression algorithm works for each state within constraints 1 and 2 by

1. Producing left entries, which may force substates to test for rules whose contexts conflict with the rules

chosen for left compression;

2. Producing right entries, which may force substates to test for rules whose contexts conflict with rules chosen for right compression;
3. Producing at most one unspecified entry, which may force right or left entries to test for rules whose contexts conflict with the one type (2) rule chosen for compression.

The resulting $\bar{M}^A$-table is ordered so that, for each state,

1. All left entries are inspected first in sequence;
2. All right entries are inspected next;
3. The one unspecified entry, corresponding to either a type (2) rule or a transfer to an "error-message state", is inspected last.

Given a fully specified BCA B' and its compressed version B", we can easily demonstrate that B" performs only those reductions possible in B' during a parse. Thus, acceptance of an input string by B' implies acceptance by B". Furthermore, acceptance by B" of an input string implies acceptance by B', since the sequence of reductions performed by B" can always be duplicated on B'. Hence, acceptance by B' is equivalent to acceptance by B", although errors in the input string will generally be detected earlier in the B' parse.

THE OPTIMAL COMPRESSION ALGORITHM-ROWS

This section describes the strategies used in compressing rows of the M^A-table, and proves, in claims 1 through 4, that these strategies result in a compressed parsing table of minimum size within constraints 1 and 2. The algorithm is developed along the lines of the three-step compression scheme of the previous section. To simplify the presentation, we first develop strategies for steps 2 and 3 of the compression scheme, and then present two alternatives for step 1. Thus, in what follows, it will be assumed that step 1 has already been applied, and that the N-T plane for state A has been cleared of all columns containing type (2) and type (4) rules to be represented by left entries in the $\overline{M}^A$-table. What remains in the N-T plane after step 1 will be represented by points in the set W, denoted by

$$W = \{(B,a,i) \mid M^A(B,a)=i \text{ is still in the } M^A\text{-table}\}.$$

(We assume that W is "compacted" so that we will examine only those rows corresponding to $a \in \overline{T} \subseteq T$ which are non-empty.)

In each row ($a' \in \overline{T}$) there are entries:

$$H(a') = \{(B,a',i) \in W\}.$$

Within a row of entries, there are groups of entries with the same particular M^A-value, i':

$$R(a',i') = \{(B,a',i') \in H(a')\} .$$

If all entries in a row are input-reducing (type (3) rules) and have the same M^A-value, i', then one compressed $\overline{M}^A$ entry results, namely,

$$\overline{M}^A(*,a') = i' \quad .$$

If we choose to compress the entries corresponding to the i' rule in a row with entries corresponding to other rules, then a substate results, consisting of one entry for rule i' preceded by entries for each rule in row a' with M^A value other than i':

$$CR(a',i') = H(a') - R(a',i')$$

We define

$$h_a = |H(a)|$$

$$r_{a,i} = |R(a,i)|$$

$$cr_{a,i} = |CR(a,i)| = h_a - r_{a,i} \ .$$

Using the $r_{a,i}$ and the $cr_{a,i}$ as measures, we can decide how to produce compressed entries from each row a in an optimal fashion.

Clearly, if a renaming rule u is chosen as the candidate for the unspecified entry (where, if $u = 0$, no rule is chosen), and, if, for each row a, the rule FR(a) is chosen as a candidate for a right entry, then there are

(1) $\sum_{a \text{ in } \overline{T}} (cr_{a,FR(a)} + 1)$ substate entries in $\mathbb{M}^A$

& $cr_{a,FR(a)} \neq 0$

(2) $\sum_{a \text{ in } \overline{T}} (1) - \sum_{(a \text{ in } \overline{T})} (1)$ right entries in $\mathbb{M}^A$

& $(0 = cr_{a,FR(a)})$

& $(u = FR(a))$

(3) 1 unspecified entries in $\mathbb{M}^A$ corresponding to a renaming rule (type (2)), if $u \neq 0$, or to an "error-message state" transfer if $u = 0$.

In what follows, we use the notation

$$\delta_{a,b} = \begin{cases} 1 & \text{if } a=b \\ 0 & \text{if } a \neq b \end{cases}$$

Thus, the number of entries passed on to the compressed table from steps (2) and (3) is

$$N(W) = \sum_{a \text{ in } \overline{T}} (2 + cr_{a,FR(a)}) = \sum_{a \text{ in } \overline{T}} \delta_{0,cr_{a,FR(a)}} (1 + \delta_{u,FR(a)}) + 1$$

Noting that part of the summation for N(W) is invariant, we let

$$m = 1 - \sum_{a \text{ in } \overline{T}} \delta_{0,cr_{a,FR(a)}}$$

and

$$N((a,b)) = cr_{a,b} + 2$$

whence

$$N(W) = \sum_{a \text{ in } \bar{T}} N(a, FR(a)) + m - \sum_{a \text{ in } \bar{T}} \delta_{u,FR(a)} \delta_{0,cr_{a,FR(a)}}$$

We are now in a position to prove the correctness of Strategy 1 of the compression scheme through Claims 1 and 2:

Claim 1: If in some row a there are rules i and j, $i \neq j$ such that

$$r_{a,j} < r_{a,i}$$

then

$$N|_{FR(a)=j} \geq N|_{FR(a)=i} .$$

Proof:

$$N|_{FR(a)=j} - N|_{FR(a)=i} = N(a,j) - N(a,i)$$

$$= cr_{a,j} - cr_{a,i}$$

$$= (h_a - r_{a,j}) - (h_a - r_{a,i})$$

$$= r_{a,i} - r_{a,j}$$

$$\geq 1.$$

Claim 2: If in some row a there are rules i and j, $i \neq j$, such that

$$r_{a,i} = r_{a,j}$$

then

$$N|_{FR(a)=i} = N|_{FR(a)=j} .$$

Proof: Like Claim 1.

Strategy 1 follows immediately from Claims 1 and 2.

Strategy 1: (Apply to rows having contexts derived from more than one rule or having contexts derived exclusively from one type (3) rule.) If, in a particular row, some type (3) rule j is the only one present, then the compression of row a in $\bar{M}^A$ is a single entry of the form

$$\bar{M}^A(*,a) = j.$$

Otherwise, we let FR(a) = j, where j is the rule having the largest

number of contexts in row a, or j is randomly chosen from among the subset of row a rules all having the (same) largest number of contexts. Then, the compression of row a in $\overline{M}^A$ is a single entry of the form

$$\overline{M}^A(*,a) = (a',0)$$

leading to a substate a' of the form

$$\left.\begin{array}{l} \overline{M}^{a'}(B_1,*) = i_1 \\ \vdots \qquad\qquad \vdots \\ \overline{M}^{a'}(B_k,*) = i_k \end{array}\right\} \quad (B_j,a,i_j) \text{ all in } CR(a,j)$$

$$\overline{M}^{a'}(*,*) = j$$

After Strategy 1 has been applied to all applicable rows, the only nonempty rows that remain to be considered in M^A are the rows containing contexts from single type (2) rules (i.e., one type (2) rule per remaining row). To compress these rows, we choose one of the remaining rules to be the unspecified entry of $\overline{M}^A$, and the rest of the type (2) rules are represented by right entries, one row at a time. Given this situation, we can prove Claim 3 about the number of $\overline{M}^A$ entries resulting from compressing the remaining type (2) rules:

Claim 3: After removing all M^A entries treated by Strategy 1, if, for rules i and j, $i \neq j$,

$$(1) \quad \sum_{a \text{ in } \overline{T}} (1 - \delta_{0,r_{a,i}}) > \sum_{a \text{ in } \overline{T}} (1 - \delta_{0,r_{a,j}})$$

then

$$N|_{u=i} < N|_{u=j} .$$

Proof: We partition $\overline{T}$ as $\overline{T} = \overline{T}_1 \cup \overline{T}_2 \cup \overline{T}_3 \cup \overline{T}_4$ such that

$$\overline{T}_1 = \{a \in \overline{T} \mid r_{a,i} \neq 0 \ \& \ r_{a,j} \neq 0\}$$

$$\overline{T}_2 = \{a \in \overline{T} - \overline{T}_1 \mid r_{a,i} \neq 0\}$$

$$\overline{T}_3 = \{a \in \overline{T} - \overline{T}_1 \mid r_{a,j} \neq 0\}$$

$$\overline{T}_4 = \overline{T} - (\overline{T}_1 \cup \overline{T}_2 \cup \overline{T}_3) .$$

Clearly (1) implies that

$$|\overline{T}_2| > |\overline{T}_3| .$$

$$
\begin{aligned}
N|_{u=j} - N|_{u=i} &= \sum_{a\in\overline{T}_1} (N(a,FR(a))|_{u=j} - N(a,FR(a))|_{u=i}) \\
&+ \sum_{a\in\overline{T}_2} (N(a,FR(a))|_{u=j} - N(a,FR(a))|_{u=i}) \\
&+ \sum_{a\in\overline{T}_3} (N(a,FR(a))|_{u=j} - N(a,FR(a))|_{u=i}) \\
&+ \sum_{a\in\overline{T}_4} (N(a,FR(a))|_{u=j} - N(a,FR(a))|_{u=i}) \\
&= \sum_{a\in\overline{T}_1} (N(a,j)|_{u=j} - N(a,i)|_{u=i}) \\
&+ \sum_{a\in\overline{T}_2} (N(a,i)|_{u=j} - N(a,i)|_{u=i}) \\
&+ \sum_{a\in\overline{T}_3} (N(a,j)|_{u=j} - N(a,j)|_{u=i}) \\
&+ \sum_{a\in\overline{T}_4} (N(a,FR(a))|_{u=j} - N(a,FR(a))|_{u=i}) \\
&= 0 + \sum_{a\in\overline{T}_2} (1) + \sum_{a\in\overline{T}_3} (-1) + 0 \\
&= |\overline{T}_2| - |\overline{T}_3| \\
&> 0.
\end{aligned}
$$

Claim 4: After removing all M^A entries treated by Strategy 1, if, for rules i and j, $i \neq j$,

$$\sum_{a\in\overline{T}} (1 - \delta_{0,r_{a,i}}) = \sum_{a\in\overline{T}} (1 - \delta_{0,r_{a,j}})$$

then

$$N|_{u=i} = N|_{u=j}.$$

Proof: As in Claim 3, we partition $\overline{T}$ as $\overline{T}_1 \cup \overline{T}_2 \cup \overline{T}_3 \cup \overline{T}_4$ with $|\overline{T}_2| = |\overline{T}_3|$. By the same analysis as in Claim 4, we find

$$N|_{u=j} - N|_{u=i} = |\bar{T}_2| - |\bar{T}_3| = 0.$$

Claims 3 and 4 then imply the following strategy for compressing the remaining type (2) rules in the M^A-table:

Strategy 2: For all rules remaining in the M^A-table after application of Strategy 1, choose the rule with index u such that

$$M_u = \sum_{a \in \bar{T}} (1 - \delta_{0,r_{a,u}}) = \max_k \left[\sum_{a \in \bar{T}} (1 - \delta_{0,r_{a,k}}) \right].$$

If no rules remain, then u = 0. If there is a choice among renaming rules for u, then we can select the unspecified entry in $\bar{M}^A$ at random from among these rules. The rest of the type (2) rules present are represented by right entries,

$$\left.\begin{array}{l} \bar{M}^A(*,a_1) = i_1 \\ \quad\vdots \qquad\quad \vdots \\ \bar{M}^A(*,a_k) = i_k \end{array}\right\} \quad (B_{jm}, a_j, i_j) \text{ all in } W - u$$

$$\bar{M}^A(*,*) = u$$

In the last entry of the compressed $\bar{M}^A$-table, if u = 0, then the unspecified entry is a transfer to an "error-message state" which then returns to S_0 to continue scanning for further errors.

THE OPTIMAL COMPRESSION ALGORITHM-COLUMNS

We now turn to the problem of specifying step 1 of the compression scheme. In the same way that we determined candidates FR(a) (a in $\overline{T}$) for right compression, we will determine candidates FL(B) (B in $\overline{N}$) for left compression. The difference here is that we do not compress every FL(B) candidate, since a smaller $\overline{M}^A$-table may result from ignoring some of the FL(B) columns and incorporating their ignored entries into the substates generated from the row compression steps of the scheme. Furthermore, type (2) (renaming) rules are considered as candidates for compression in step 1, as well as in steps 2 and 3 of the compression scheme, since it may be more economical to compress an unfavored renaming rule by columns, rather than by rows.

We will consider the full N-T plane, letting W represent the set of nonempty contexts in this plane. In each column B' in $\overline{N}'$ there are entries:

$$V(B') = \{(B',a,i) \in W\}.$$

Within a column of entries there are groups of entries with the same particular M^A-value, i':

$$L(B',i') = \{(B',a,i') \in V(B')\}.$$

If we were to compress the entries of L(B',i') into a left entry, then a substate would be produced consisting of entries in the column B' with M^A-value other than i':

$$CL(B',i') = V(B') - L(B',i).$$

We define

$$v_B = |V(B)|$$

$$l_{B,i} = |L(B,i)|$$

$$cl_{B,i} = |CL(B,i)| = v_B - l_{B,i}.$$

We consider the problem of deciding for each column $B \in N$ a candidate FL(B) for left compression (where FL(B)=0 means that there is no candidate in column $B \in N$). Having determined the FL(B) we must then decide for each $B \in N$ for which $FL(B) \neq C$ whether to accept the left compression candi-

dacy of FL(B) and produce a left entry linked to a substate of $\bar{M}^{AA}$ or to reject the candidacy (set FL(B) = 0). We may denote the minimum cost (in terms of number of entries in $\bar{M}^{A}$) for left compression recursively by defining:

1) the set of left compression candidates:

$$F = \{B \in N \mid FL(B) \neq 0\};$$

2) the cost of compressing entries W when no left compression candidates remain:

$$C(W,\Phi) = N(W);$$

3) the cost of compressing entries W when $F \neq \Phi$ includes a left compression candidate B:

$$C(W,F) = \min \begin{cases} C(W,F - \{B\}) \\ cl_{B,FL(B)} + 2 + C(W-\{B\}xT,\ F - \{B\}) - \delta_{0,cl_{B,FL(B)}} \end{cases}.$$

We are now in a position to state Claim 5:

Claim 5: Within constraints 1 and 2, the number of entries in the compressed $\bar{M}^{A}$-table is minimized by choosing for left compression in step 1 those FL(B) rules for which C(W,F) is minimal. Applying steps 2 and 3 then yields the minimum table.

Proof:

It is easy to see by induction on |F| that the order in which the elements of F are considered has no effect on the minimum cost of compression. The quantity

$$cl_{B,FL(B)} + 2 + C(W-\{B\}xT,\ F - \{B\}) - \delta_{0,cl_{B,FL(B)}}$$

is called the cost of accepting B candidacy and is denoted by p_B. The quantity

$$C(W,F - \{B\})$$

is called the cost of ignoring B candidacy and is denoted by $\bar{p}_B$. Clearly if C(W,F) is minimized with C(W,F - {B}), then the choice of FL(B) is immaterial. Otherwise, C(W,F) is minimized by choosing FL(B) such that $cl_{B,FL(B)}$ is minimal, i.e. such that $l_{B,FL(B)}$ is maximal. If in some column B there are non-input-erasing rules i and j, $i \neq j$ such that $l_{B,i} = l_{B,j}$, then the choice between i and j for left candi-

dacy may be made randomly.

Strategy 3: For each B∈N, we choose FL(B)=i where i is non-input-erasing and where

$$l_{B,i} = \max_k (l_{B,k}).$$

If all rules in column B are input-erasing, then we choose FL(B)=0.

One method for calculating C(W,F) is to employ a recursive algorithm which directly encodes the recursive definition of C(W,F), thus essentially trying all accept/ignore combinations for candidates in F. We will find that at times such a combinatorial approach is our only alternative. But, often, we will be able to decide upon acceptance of candidacy without recursion.

Even though we cannot always determine $\bar{p}_B$ explicitly, we can always determine quantities $\tilde{p}_B$ and $\hat{p}_B$ such that

$$\tilde{p}_B \leq \bar{p}_B \leq \hat{p}_B.$$

Thus if

$$p_B \leq \tilde{p}_B$$

we should accept B candidacy, since

$$C(W,F-\{B\})-(cl_{B,FL(B)}+2-\delta_{0,cl_{B,FL(B)}}+C(W-\{B\}xT,F-\{B\}))$$

$$= \bar{p}_B - p_B$$

$$\geq \tilde{p}_B - p_B$$

$$\geq 0.$$

Similarly if

$$\hat{p}_B \leq p_B$$

we should ignore B candidacy, since

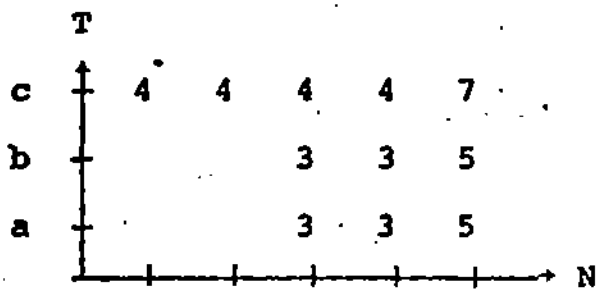
$$cl_{B,FL(B)}+2-\delta_{0,cl_{B,FL(B)}}+C(W-\{B\}xT,C-\{B\})-C(W,F-\{B\})$$

$$= p_B - \bar{p}_B$$

$$\geq p_B - \hat{p}_B$$

$$\geq 0.$$

If $\tilde{p}_B = \bar{p}_B = \hat{p}_B = p_B$ then we choose to accept B candidacy. It is in the calculations of $\tilde{p}_B$ and $\hat{p}_B$ that the favored/unfavored denotations of entries for right compression become useful. As a motivating example, consider the full M^A-table with graphic representation



FL(B):	0	0	3	3	5
p_B:	-	-	3	3	3

It is easy to see that <u>regardless</u> of what happens with left compression, row c will be left with <u>at least</u> two entries "4" (in columns B_1 and B_2). Since rule "7" cannot possibly survive left compression with that many entries, the surviving entries "4" will be favored for right compression. Moreover, if the entry "7" survives, it will be unfavored. Since the entries "5" are stack-reducing, they will be unfavored if they survive left compression. If we choose to ignore B_3 candidacy, then it will be necessary to

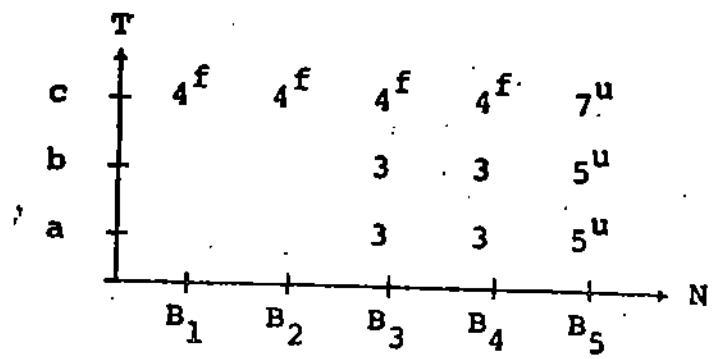
represent the entries of $V(B_3)$ as either unspecified, right, or substate entries. It is easy to see that the entry "4" at (c,B_3) will be included with the remaining "4"'s in row c and represented by a single right entry. Thus, there is no additional cost incurred by the "4" at (c,B_3). If we choose to ignore B_5 candidacy, then it will be necessary to represent the entries of $V(B_5)$ as substate entries of rows a, b, and c, since all entries in $V(B_5)$ are unfavored. Thus, we may calculate $\tilde{p}_B$, the minimum cost of ignoring a B candidacy, as

$$\tilde{p}_B = |\{(B,a,i) \in V(B) \text{ which are unfavored for right compression}\}|$$

and $\hat{p}_B$, the maximum cost of ignoring a B candidacy as

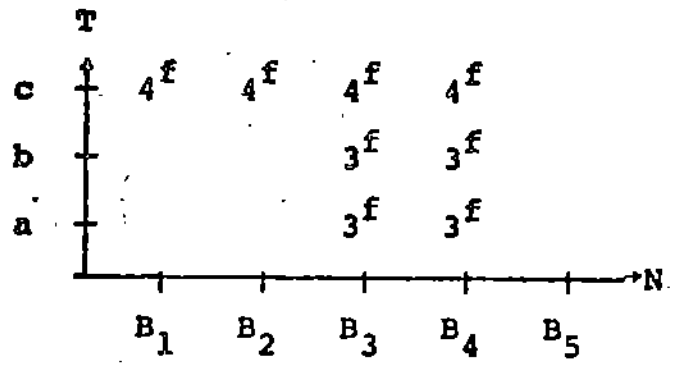
$$\hat{p}_B = v_B - |\{(B,a,i) \in V(B) \text{ which are favored for right compression}\}|.$$

Marking favored/unfavored entries by a superscript f/u, our example becomes



FL(B):	-	-	3	3	5
$\hat{p}_B$:	-	-	3	3	6
p_B:	-	-	3	3	3
$\tilde{p}_B$:	-	-	0	0	6

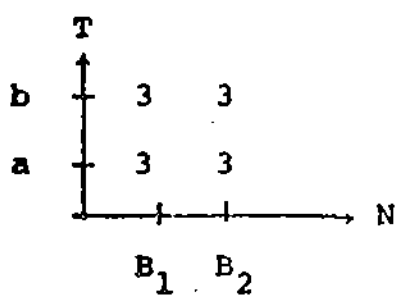
We see then that we should ignore B_3 and B_4 candidacies and accept B_5 candidacy. After doing so we have



Since no left compression candidates remain, we perform right compression and obtain the final $\vec{M}^A$-table

$$\vec{M}^A(B_5,*) = (A',0)$$
$$\vec{M}^A(*,c) = (A_4,1)$$
$$\vec{M}^A(*,*) = (A_3,0)$$
$$\vec{M}^{A'}(*,c) = (A_7,0)$$
$$\vec{M}^{A'}(*,*) = (A_5,2)$$

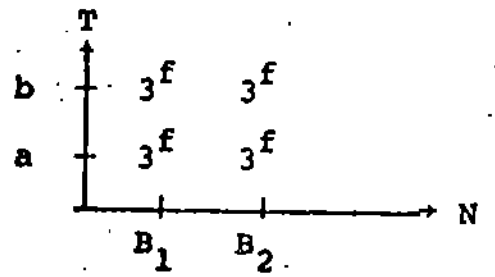
Of course it is not always possible to determine whether to ignore or accept candidacy in such a closed fashion. When such cases occur, it is necessary to select a candidate and try both possibilities. For example



FL(B):	3	3
$\hat{p}_B$:	2	2
p_B:	1	1
$\tilde{p}_B$:	0	0

Since for both candidates, $\tilde{p}_B < p_B < \hat{p}_B$, it follows that

the acceptance decision cannot be made <u>a priori</u>. So, we first try ignoring B_1 candidacy. But then the "3"s in column B_1 (and all "3"s in corresponding rows) become favored:

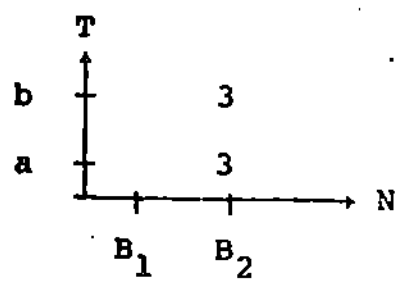


FL(B):	0	3
$\hat{p}_B$:	-	0
p_B:	-	1
$\tilde{p}_B$:	-	0

whence we ignore B_2 candidacy as well. Since no left compression candidates remain, we perform right compression and obtain the $\bar{M}^A$-table consisting of the single entry

$$\bar{M}^A(*,*) = (A_1,0).$$

Now we try the second alternative, that of accepting B_1 candidacy:



$$\bar{M}^A(B_1,*) = (A_1,0)$$

FL(B):	-	3
$\hat{p}_B$:	-	2
p_B:	-	1
$\tilde{p}_B$:	-	0

Since this second alternative has already produced a (partial) $\overline{M}^A$-table whose size equals that of the previously obtained $\overline{M}^A$-table, we abandon this alternative path.

We now formalize left-compression.

For $a' \in T$ define

$$\hat{R}(a',i) = \{(B,a',i) \in R(a',i) \mid FL(B)=0\}$$

$$\hat{r}_{a',i} = |\hat{R}(a',i)|.$$

Definition. Each element of R(a',i) is said to be unfavored if and only if either

a) i is stack-reducing, or;

b) there is a rule j for which $\hat{r}_{a',j} > r_{a',i}$.

Definition. Each element of R(a',i) is said to be favored if and only if

a) i is not stack-reducing, and;

b) for each rule $j \neq i$, $\hat{r}_{a',i} > r_{a',j}$.

We develop a recursive procedure which requires as parameters the $\overline{M}^A$-table, W, and a set of left-context, left-compression-candidate-pairs, $F = \{(B,FL(B)) \mid B \in N\}$, where some of the FL(B)'s may be zero. The procedure returns a set of pairs

$$CL = \{(B,i) \mid B \in N \ \& \ i=0 \text{ or } i=FL(B)\}$$

indicating which of the FL(B)'s are optimally accepted as candidates. The procedure also returns the COST (the number of entries for the optimal $\overline{M}^A$-table.) The full, recursive compression algorithm is given as follows (where we do not indicate the use of $\hat{p}_B$ and $\tilde{p}_B$ in abandoning recursive descents where possible within the optimization procedure).

procedure Compress(W) ;

1. set F = ϕ;
2. for every $B \in \bar{N}$ do
   2.1 determine FL(B);
   2.2 set F = F $\cup$ {(B,FL(B))};
   end;
3. call optimize (W, F, COST, CL);
4. for every (B,j) $\in$ CL with $j \neq 0$ do
   4.1 create a substate containing right entries for the elements of CL(B,j) and one last entry for L(B,j);
   4.2 create a left entry for L(B,j) leading to the substate of 4.1;
   4.3 set W = W - B x T;
   end
5. perform right and unspecified compression on W;
6. halt;

end compress;

where the optimization procedure is:

procedure Optimize(W, F, COST, CL) ;

1. set W' = W, COST = 0, IG = {(B,0) $\in$ F}, F' = F - IG, AC = ϕ;
2. using IG $\cup$ F' for determining the FL(B)'s, mark as favored/unfavored as many elements of W' as possible;
3. while there exists (B,FL(B)) $\in$ F' such that either $\hat{p}_B \leq p_B$ or $\tilde{p}_B \leq p_B$ do
   3.1 set F' = F' - {(B,FL(B))};
   3.2 if $p_B \leq \tilde{p}_B$
   then do
   3.2.1 set AC = AC $\cup$ {(B,FL(B))};
   3.2.2. set W' = W' $\leftarrow$ {B} x T;

3.2.3 set COST = COST + p_B;

end;

else

3.2.4 set IG = IG ∪ {(B,0)};

3.3 using IG ∪ F' for determining the FL(B)'s, mark as favored/unfavored as many elements of W' as possible;

end;

4. if F' = Φ

then do

4.1 set COST = COST + N(W');

4.2 set CL = AC ∪ IG;

4.3 return;

end;

else do

4.4 select some (B,FL(B))∈F';

4.5 set F" = F' - {(B,FL(B))};

4.6 call optimize (W', IG ∪ F" ∪ {(B,0)}, $COST_I$, CL_I);

4.7 call optimize (W'−{B}xT, IG ∪ F", $COST_A$, CL_A);

4.8 if $COST_I \leq COST_A + p_B$

then do

4.8.1 set COST = COST + $COST_I$;

4.8.2 set CL = AC ∪ CL_I;

4.8.3 return;

end;

else do

4.8.4 set COST = COST + p_B + $COST_A$;

```
            4.8.5  set CL = AC ∪ {(B,FL(B))} ∪ CL_A;
            4.8.6  return;
            end;
        end;
end optimize;
```

MERGING OF SUBSTATES WITHIN COMPRESSED STATES

Having described the compression of parsing table entries for a fixed inspection sequence, we can next exploit the similarity of states in the resulting parser and states in a Floyd-Evans parser.(9) Because of this similarity, we are able to apply the Ichbiah-Morse read-entry merging algorithm to the merging of substates in the compressed parser. In principle, the merging process could be applied to the entire M-table at once, rather than to each $\mathbf{M}^A$ subtable. However, such a merging process would obscure the next development; namely, the discovery of rearrangements in the inspection sequence for a given state that violate the original fixed sequence assumption, but yield merged and compressed states whose size cannot be reduced further.

As an example of merging, if the substates corresponding to two rows of a state are identical, they are merged into a single substate. If they are identical except for their last (no context) entries, they are merged into a single substate whose last entry is replaced by two partial right-context entries, one for each of the original two substates. The minor differences between our merging process and the original reported in Ichbiah and Morse (9) are that

(a) We treat the process of merging read entries of the set of row substates and the set of column substates as two separate merging problems.

and

(b) For each of those merged row and column substates not having the same last (no context) entry, we attach a distinguishing partial right context (or left context) entry at the end of the merged substate.

The following definitions formalize the correspondence with this merging algorithm.

Definition For each a in $\bar{a}$ of $\mathbf{M}^A$, let

$$\bar{r}(a) = \{(B_1,*),\ldots,(B_{n_a-1},*),(*,*)\}$$

be the partial left contexts of the a' substate of $\mathbf{M}^A$.

Definition For each a in $\bar{a}$ of $\mathbb{M}^A$, let

$$\bar{H}(a) = ((B_1,*),p_1),\ldots,((B_{n_a},*),p_{n_a-1}),((*,*),p_{n_a})$$

be the set of partial-context rule pairs that define the actions of the a' substate of $\mathbb{M}^A$.

In the same way, we can define $\bar{r}(B)$ and $\bar{V}(B)$ for B in $\bar{B}$ of $\mathbb{M}^A$. Then, from these definitions, we can arrive at the counterpart of the Ichbiah-Morse "graph associated with the read entries".(9, p.505):

Definition Consider the set $\{\bar{H}(a) : a \text{ in } \bar{a} \text{ of } \mathbb{M}^A\}$ (respectively, the set $\{\bar{V}(B) : B \text{ in } \bar{B} \text{ of } \mathbb{M}^A\}$). If two or more rows (columns) have identical entries, they are considered as one row (column). If two or more rows (columns) have identical entries except for the (*,*) no context entry, then they are considered as one row in which the last (*,*) entry is replaced by the corresponding $(B_i,*)$ (respectively, $(*,a_j)$) partial context tests.

The graph G = (X,W), where S is a set of nodes and W is a set of arcs, can be defined as follows:

(a) A node of the set X is associated with each of the remaining rows $\bar{H}(a)$ (or columns $\bar{V}(B)$). In addition, X contains a node $\emptyset$ associated with a fictitious empty row (column).

(b) Consider now the rows (columns) associated with the distinct nodes α and β of the set X. The arc (α,β) is contained in the set W if the entries of the row (column) corresponding to β are included in the row (column) corresponding to α. (Obviously, if $\alpha \neq 0$, there is an arc $(\alpha,\emptyset)$ in W.)

It follows from the way we defined nodes that no two nodes may correspond to identical rows (columns). Hence, for each arc (α,β) of the set W, it is possible to define a positive cost which represents the number of entries in the row (column) associated with α that are not in the row (column) associated with β, as well as the cost of an additional partial context test at the end of the merged row (column) to distinguish α from β, if necessary.

Suppose that $\{\alpha, \beta, \emptyset\}$ is a path (notation from Ichbiah and Morse (9,p.505)) between α and $\emptyset$ in G. We may then generate cost$(\beta,\emptyset)$ entries for the entries of β. An additional cost(α,β) entries are needed for the entries corresponding to α but not to β. This is the graph equivalent of substate mergers. We get cost(α,β) + cost$(\beta,\emptyset)$ productions instead of cost$(\alpha,\emptyset)$ + cost$(\beta,\emptyset)$.

Consider now the conditions that must be satisfied by a partial graph G' of G so that substates may be associated with G'. Clearly, no more than one arc should be incident out of each node. In addition, for any node α, there should be a path in A from α to $\emptyset$. Hence, exactly one arc should be incident out of each node other than $\emptyset$. Thus, the partial graph G' can be shown to be an inverse arborescence with $\emptyset$ as its root.

Thus, the minimum number of merged substate entries that are associated with a partial graph will be obtained for the minimum-cost inverse arborescence.

Algorithm for the Minimum-Cost Inverse Arborescence (MIA)

The minimum-cost inverse arborescence G' = (X,W') of G = (X,W) is constructed as follows:

(a) W' contains no arc incident out of $\emptyset$

(b) For every node α in X other than $\emptyset$, W' contains a single arc incident out of α. This arc is the arc of W which has the smallest cost of all the arcs incident out of α.

From the preceding discussion, we see that the MIA algorithm generates a minimum-cost arobrescence when applied separately to the row substates and to the column substates of $\overline{M}^A$.

Example

Figure 2. A State for Which Both Merging and Compression Are Desirable

e	1	1	1	1	1	8	8	8	8	8	8
d	1	1	1	1	1	7	7	7	7	7	7
c	2	3	4	5	6						
b	2	3	4	5	6						
a	2	3	4	5	6						
	B	C	D	E	F	G	H	I	J	K	L

Then, the compressed and merged table for the state shown in Figure 2 contains a total of 15 entries, as shown below:

Figure 3. The Compressed and Merged Table for Figure 2

State A: $M^A(B,*) = B'$ States B':C':D':E':F': $M^A(*,d) = 1$

$M^A(C,*) = C'$ $M^A(*,e) = 1$

$M^A(D,*) = D'$ $M^A(B,*) = 2$

$M^A(E,*) = E'$ $M^A(C,*) = 3$

$M^A(F,*) = F'$ $M^A(D,*) = 4$

$M^A(*,d) = 7$ $M^A(E,*) = 5$

$M^A(*,e) = 8$ $M^A(*,*) = 6$

$M^A(*,*) = \text{error}$

RELAXATION OF CONSTRAINTS IN THE COMPRESSION PROCESS

The final problem treated concerns the relaxation of the ordering constraint for the main entries in the M^A table, in order to arrive at a compressed M^A table in which further mergers and/or reorderings can at best leave the total number of entries unchanged. We consider the question of which subset $\bar{T}'$ of $\bar{T}$, the terminals read by state M^A, might better be inspected prior to the tests for $\bar{N}$. It should be noted that the converse question concerning some subset of $\bar{N}$ to be inspected after symbols from $\bar{T}$ need not be answered, since moving a subset of $\bar{T}$ in front of the $\bar{N}$ tests is equivalent to moving a subset of $\bar{N}$ to follow the $\bar{T}$ tests of state M^A. As an example of where this reorganization is useful, consider the following variation of Figure 2:

Figure 4. A Parser State for Which Reorganization Minimizes The Number of Entries

e	1	9	10	11	12	8	8	8	8	8	8
d	1	9	10	11	12	7	7	7	7	7	7
c	2	3	4	5	6						
b	2	3	4	5	6						
a	2	3	4	5	6						
	B	C	D	E	F	G	H	I	J	K	L

Without reorganization the state of Figure 4 yields a minimized M^A table having entries as in Figure 5:

Figure 5. Preliminary Minimization of the State in Figure 4

A: $M^A(B,*) = B'$
$M^A(C,*) = C'$
$M^A(D,*) = D'$
$M^A(E,*) = E'$
$M^A(F,*) = F'$
$M^A(*,d) = 7$
$M^A(*,e) = 8$
$M^A(*,*) =$ error

B': $M^A(*,d) = 1$
$M^A(*,e) = 1$
$M^A(*,*) = 2$

C': $M^A(*,d) = 9$
$M^A(*,e) = 9$
$M^A(*,*) = 3$

D': $M^A(*,d) = 10$
$M^A(*,e) = 10$
$M^A(*,*) = 4$

E': $M^A(*,d) = 11$
$M^A(*,e) = 11$
$M^A(*,*) = 5$

F': $M^A(*,d) = 12$
$M^A(*,e) = 12$
$M^A(*,*) = 6$

In Figure 5, if the (*,d) and (*,e) entries of the subtables respectively involved the same reductions, we could merge these subtables as in the first example above. The characteristic property of the Figure 5 subtables is that a subset of the symbols consumed are identical in these subtables and that, for each subtable, all such reductions are identical. Put formally, let $U \subseteq \overline{T}$ be the set of terminals such that

$\mathbb{M}^i(*,u) = \text{action}_1$ for u in U and i a subtable of $\mathbb{M}^A$

Then, this implies that reorganization of the "main" portion of the $\mathbb{M}^A$ table to scan all u in U before scanning all X in $\overline{N}$ will result in u-subtables whose stack-reducing entries can be merged. This is because all these stack-reducing entries correspond to the distinct subtables in the original $\mathbb{M}^A$ state from which the set U was constructed.

Figure 6. Reorganization of the Compressed State in Figure 5

A: $\mathbb{M}^A(*,d) = d'$
$\mathbb{M}^A(*,e) = e'$
$\mathbb{M}^A(B,*) = 2$
$\mathbb{M}^A(C,*) = 3$
$\mathbb{M}^A(D,*) = 4$
$\mathbb{M}^A(E,*) = 5$
$\mathbb{M}^A(F,*) = 6$
$\mathbb{M}^A(*,*) = \text{error}$

d':e': $\mathbb{M}^A(B,*) = 1$
$\mathbb{M}^A(C,*) = 9$
$\mathbb{M}^A(D,*) = 10$
$\mathbb{M}^A(E,*) = 11$
$\mathbb{M}^A(F,*) = 12$
$\mathbb{M}^A(*,d) = 7$
$\mathbb{M}^A(*,*) = 8$

The Figure 6 reorganization involves a total of 15 entries, where the table of Figure 5 contains 23 entries. Furthermore, any reorganization of Figure 6 does not lead to a smaller $\mathbb{M}^A$ table.

This reorganization process can be expressed in terms of the following algorithm:

Notation: $\overline{T}'$ is the subset of $\overline{T}$ consisting of terminals tested in more than one subtable of $\mathbb{M}^A$. $\overline{U}$ is the subset of powerset($\overline{T}'$) in which each element u in $\overline{U}$ is a largest subset of $\overline{T}'$ for which the reductions of u are identical within each subtable. Then, the following reorganization consists of the folowing steps:

while $\bar{U} \neq 0$ **do**

1. Select an element u in $\bar{U}$.
2. Set $\bar{U} = \bar{U} - u$.
3. Construct a reorganized A-state table $\bar{M}^A$ with the u symbols compressed before the stack symbols of the original compression algorithm.
4. Merge the newly-created subtables in the resulting $\bar{M}^A$ table.
5. **If** $\bar{M}^A$ M^A, **then** abandon the reorganization
   else set $M^A = \bar{M}^A$
   end

FURTHER REFINEMENTS OF THE COMPRESSION PROCESS

Some further second-order improvements might be expected from attempts to merge and reorganize the entire parsing table rather than each separate state. From all observations, however, little if any improvement can be realized from such global minimization because the states of a BCA parser tend to be disjoint. Furthermore, undertaking such minimization in an implemented parser-generating system forces the resulting system to execute much more slowly than the corresponding state-by-state version of the system. For this reason, our implementation of these algorithms reflected in the descriptions above is a state-by-state process.

COMPARISON WITH RESULTS OBTAINED FROM OTHER SCHEMES

In order to obtain some feeling for the success of these compression techniques, we compared our PGS with the weak precedence method of Ichbiah and Morse, as well as the LALR method of DeRemer. As a basis for comparison, we chose 1) a simple grammar for arithmetic expressions, 2) a grammar for XPL, and 3) a grammar for Algol 60 (as supplied to us by J. D. Ichbiah [11]). A method for calculating the minimum table sizes for weak-precedence parsers and LALR parsers is developed in [7]. The figures resulting from those calculations, as well as available statistics on actual table sizes, are presented in Table 1. Since the lower bound calculations presume no wasted bits in a table entry, we present our PGS results as total table size without wasted bits along with total table size with wasted bits (including space for specifying semantic routine indices).

Table 1. Summary of Parser Table Sizes (in bits).

Grammar	lower bound on table size		PGS actual table size	
	weak precedence	LALR	without waste	with waste
Expression	914	447	386	930
XPL	10090	7493**	5180	8160
Algol 60	14746*	9495***	8546	12810

* According to the printout supplied to us by J. D. Ichbiah [11], and incorporating optimizations indicated by him, the weak precedence parser for Algol 60 contains 774 table entries. Using 24 bits as the entry size [12] we find that the actual size of a typical Algol 60 weak precedence parser table is 18576 bits.

** Horning and Lalonde [10] report a size of 10000 bits for a typical XPL LALR parser table.

*** Horning and Lalonde [10] report a size of 22568 bits for a typical Algol 60 LALR parser table.

BIBLIOGRAPHY

1. Aho, A. V., Denning, Pl J., and Ullman, J. D. Weak and mixed-strategy precedence parsing, J. ACM 19 (1972),225-243.

2. DeRemer, F. L., Practical translators for LR(k) languages, doctoral dissertation, MIT, Cambridge, Mass., September, 1969.

3. ____, Simple LR(k) grammars, Comm. ACM 14 (1971), 453-460.

4. Floyd, R. W., Bounded-context syntactic analysis, Comm. ACM 7 (1964), 62-67.

5. ____, Syntactic analysis and operator precedence, J. ACM 10 (1964), 316-333.

6. Gray, J. N. and Harrison, M. A., On the covering and reduction problems for context-free grammars, J. ACM 19 (1972), 675-698.

7. Harrison, M. A., On the parsing of deterministic languages, Computer Science Conference, Columbus, Ohio, February 20, 1973.

8. Horning, J. J. and LaLonde, W. R., Empirical comparison of LR(k) and precedence parsers, ACM SIGPLAN Notices 5 (November 1970), 10-24.

9. Ichbiah, J. D., Computer printout of an Algol 60 grammar run through the Ichbiah-Morse system, personal communication, 1971.

10. ____ and Morse, S. P., A technique for generating almost optimal Floyd-Evans productions for precedence grammars, Comm. ACM 13 (1970), 501-508.

11. Knuth, D. E., On the translation of languages from left to right, Information and Control 8 (1965), 607-639.

12. McAfee, J. and Presser, L., An algorithm for the design of simple precedence grammars, J. ACM 19 (1972), 385-395.

13. Mickunas, M. D., Lancaster, R. L., and Schneider, V. B., Transforming LR(k) grammars to LR(1), SLR(1), and (1,1) bounded right context, J. ACM (to appear).

14. ____ and Schneider, V. B., On the ability to generate (1,1) bounded right context grammars from LR(k) grammars. IEEE Switching and Automata Theory Symposium (1973).

15. ____ and Schneider, V. B., A parser generating system for constructing compressed compilers, Comm. ACM 16 (1973), 669-676.

16. Samelson, K. and Bauer, F. L., Sequential formula translation, Comm. ACM 3 (1960), 487-492.

17. Schneider, V. B., A system for designing fast programming language translators, Proc. AFIPS 1969 SJCC, 777-792.

18. Wirth, N. and Weber, H., EULER, a generalization of Algol and its formal definition, Comm. ACM 9 (1966), 13-25 and 89-99.