A Solution to an Open Problem by Knuth*

DAVID PAGER

University of Hawaii, 2565 The Mall, Honolulu 96822, Hawaii

In Knuth (1965) the problem of minimizing the number of sets of states required for his parsing algorithm is raised as an open question. This question is discussed by among others, Lewi (1968), Korenjak (1969), Early (1970), and Loeckx (1970). We solve the problem by showing it is equivalent to that of finding the minimal representation for an incompletely specified finite automaton. A solution may thus be obtained using the known methods for the latter.1 This result may be viewed in contrast to Pager (1969) and (1970a) where it is shown that it is not generally possible to make optimizations of this kind.

We make free use of the notation defined in Knuth (1965) and, for finite automata, that of Booth (1967).2 We correct a minor technical error in Knuth by taking the zeroth production to be $S_0 \rightarrow S$ and employing $S \vdash^k$ as a goal string when forming the modification of $B$ required for a $LR(k)$ parser. The closure of $[0,0; \vdash^k]$ is denoted by $\mathcal{G}_0$. If $\alpha \Rightarrow \beta$ by a rightmost derivation, we say $\beta$ is c-derivable (canonically derivable) from $\alpha$. A sentential form c-derivable from the goal string is referred to as a CSF (canonical sentential form). If $\phi$ is a CSF, by $\phi(t)$ we mean the $t$th letter in $\phi$ from the left, whereas by $\phi(r,t)$ we mean the substring $\phi(r)\phi(r+1)\cdots\phi(t)$. If $\omega = a\omega$, $\sigma$ is called a head of $\omega$ and $\omega$ a tail. We denote the length of a string $\sigma$ by $|\sigma|$.

If $(n,p)$ is the handle of a CSF $\phi$ then $\text{CUT}(\phi) = n$. A $k$-state is one of the form $[p,j;\alpha]$ where $|\alpha| = k$. A state $[p,j;\alpha]$ is said to apply to $\phi(t)$, where $\phi$ is a CSF and $0 \leq t \leq \text{CUT}(\phi)$, if $\phi = \phi(1,t-j)X_{p_1}\cdots X_{p_n}\alpha$ for some terminal $\sigma$ and $\phi(1,t-j)A_{\sigma}\alpha$ is a CSF. If $\xi$ is a state $[p,j;\alpha]$ such that $j < n_\alpha$, then $\xi'$ is said to be an immediate successor of $\xi$ if $\xi'$ is of the form $[q,0;\beta]$ where $X_{p_(j+1)} = A_q$ and $\beta$ is in $H_k(X_{p_(j+2)} \cdots X_{p_n}\alpha)$; $\xi'$ is called a

* This work was supported by the National Science Foundation under Grant GJ-596. A shortened version of the paper was presented at the ACM Symposium on the Theory of Computing, Northampton, May 1970. The author would like to express his thanks to the referee for his helpful advice and criticism.

1 See, e.g., Grasselli and Luccio (1965), Luccio (1969), McClusky (1962), Meisal (1967), Miller (1966), Paull and Unger (1959), Troshin (1965), and Unger (1965).

2 A summary of this notation is given in the Appendix.
$Y$-successor of $\zeta$ if $X_{\rho(\cdot+1)} = Y$ and $\zeta^\prime = [p, j + 1; \alpha]$. A derivation of a k-state $\zeta$ is a sequence of states $\zeta_1, \ldots, \zeta_m$ such that $\zeta_1 = [0, 0; -1^k]$, $\zeta_m = \zeta$ and for $1 \leq i \leq m-1$ $\zeta_{i+1}$ is an immediate successor or, for some $Y$, a $Y$-successor of $\zeta_i$. Clearly the closure of a set of states $\mathcal{S}$, as defined in Knuth, is the smallest set of states which contains $\mathcal{S}$ and includes all the immediate successors of its own members. If $\mathcal{S}$ is a set of states such that at least one of its members has a $Y$-successor, the $Y$-successor of $\mathcal{S}$ is the closure of the $Y$-successors of its members (otherwise $\mathcal{S}$ has no $Y$-successor).

A sequence of characters $Y_1, \ldots, Y_t$ is said to be a derivation of a set of states $\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_t$ such that $\mathcal{S}_0$ is the closure of $[0, 0; -1^k]$, $\mathcal{S}_t = \mathcal{S}$ and for $0 \leq i \leq t-1$, $\mathcal{S}_{i+1}$ is a $Y_{i+1}$-successor of $\mathcal{S}_i$. A derivable state set is one which has a derivation.

Knuth describes a method for generating all the sets of states derivable from $\mathcal{S}_0$ for a given grammar $\mathcal{G}$. This leads to the formulation of a $LR(k)$ parsing algorithm for $\mathcal{G}$ which employs a set of rules associated with each set of states. Each rule is either of the form "If $\alpha$ is the current head of the input stream, apply reduction $p$," for some $\alpha$, $p$, or else, for some $Y$, $i$, "If the next input character is $Y$, go to state $i$," with rules of the first kind having precedence. For the purposes of parsing, once such a set of rules has been determined the information about what states are in the various state sets may be discarded, and the problem of minimizing the number of state sets required is expressible as one of minimizing the number of sets of rules. For a class $\Sigma$ of $k$-state sets to be associated in the manner described by Knuth with a class of sets of rules, it must be adequate in the following sense: (1) Corresponding to each $\mathcal{S} \in \Sigma$ and character $Y$ for which $\mathcal{S}$ has a successor, there must be some $\mathcal{S}' \in \Sigma$ such that $\mathcal{S}'$ contains the $Y$-successor of $\mathcal{S}$; (2) For each $\mathcal{S} \in \Sigma$ and $\zeta, \zeta' \in \mathcal{S}$, if $\zeta = [p, n_\zeta; \alpha]$ for some $p$, $\zeta' = (\text{say}) [q, j; \beta]$, and $\zeta \neq \zeta'$, then $\alpha \notin H_b(X_{\rho(0+1)} \cdots X_{\rho n})$.

We now show how one can construct a finite automaton to simulate the control of a $LR(1)$ parser provided by an adequate class of $1$-state sets. The case for $LR(k)$ parsers where $k \neq 1$ will be taken up later. If $\Sigma = \{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_r\}$ is an adequate class of 1-state sets for a grammar $\mathcal{G}$, let $A_\Sigma$ be the finite automaton

$$<\text{vocabulary of } \mathcal{G}, \{\mathcal{S}_0, \mathcal{S}_1, \ldots, \mathcal{S}_r, \mathcal{S}_s\}, \{z_1, \ldots, z_s, z_s, \delta_\Sigma, \omega_\Sigma\}>$$

such that for all characters $Y$ and $0 \leq i \leq r$ if $\mathcal{S}_i$ has a $Y$-successor, then $\delta_\Sigma(Y, \mathcal{S}_i') = \mathcal{S}_i'$ where $\mathcal{S}_i'$ is a state set which contains the $Y$-successor of $\mathcal{S}_i$; and if $\zeta \in \mathcal{S}_i$ where $\zeta = [p, n_\zeta; \alpha]$, then $\delta_\Sigma(b, \mathcal{S}_i) = \mathcal{S}_s$ and $\omega_\Sigma(b, \mathcal{S}_i') = z_{p}$; and for all characters $Y$, $\delta_\Sigma(Y, \mathcal{S}_s) = \mathcal{S}_s$ and $\omega_\Sigma(Y, \mathcal{S}_s) = z_{s}$. No other values of $\delta_\Sigma$ and $\omega_\Sigma$ are defined. We will call the outputs $z_i$ normal (as
opposed to $z_\#$). The symbols $\mathcal{S}_0'$, $\mathcal{S}_\#$, and $z_\#$ will be reserved to denote for machines $A_\Sigma$ the special two states and output which they represent above.

**Example.** Consider the 12 state sets listed in Table I of Knuth (1966, p. 620) for the grammar (26). Reading down let us label them as $\mathcal{S}_0$, ..., $\mathcal{S}_{12}$, let $\mathcal{S}_0 = \{83ab\}$, and let $\Sigma = \{\mathcal{S}_0, ..., \mathcal{S}_{12}\}$. Then $A_\Sigma$ is the machine shown in Fig. 1.

If in a grammar $\mathcal{G}$ for which $\mathcal{S}_\Sigma$ is an adequate class of 1-state sets, $\phi$ is a CSF to be pruned, then starting in the initial state $\mathcal{S}_0'$ we feed into $A_\Sigma$ the characters $\phi(1), \phi(2), ..., \phi(t)$, the first output is obtained. If this output is $z_\#$, then $\phi$'s handle is $(t - 1, p)$. Clearly by maintaining a stack of the kind described by Knuth, popping the stack when $\mathcal{S}_\#$ is reached, and then re-entering $A_\Sigma$ at the state appearing at the top of the stack, one can make use of a finite automaton such as $A_\Sigma$ to direct the parsing of the CSFs of $\mathcal{G}$.

We will show by means of a series of theorems that one can obtain the smallest class of state sets for $\mathcal{G}$ in the following way. Find the class $\mathcal{S}_\Sigma = \{\mathcal{S}_0' \}, $ of all derivable state sets and then determine the minimal

---

Knuth's table requires the following corrections. Replace 62$^{+}$ 82ab by 62$^{+}$ 82ab and 84ab by 83ab in the "go to" column. Add to the table the state {83ab} with an entry specifying that if $y_1 = b$ the action required is to shift and go to 84ab.
C-class for \( A_2 \). If \( \{B_1, ..., B_r\} \) are the blocks of the C-class and for \( 1 \leq i \leq r \), \( B_i = \{S_{i,1}, ..., S_{i,m_i}\} \) for some subscripts in \([0, u]\), then the minimal class of state sets for \( \mathcal{F} \) has \( r \) members and the sets are

\[
\mathcal{S}_{i,1} \cup \mathcal{S}_{i,2} \cdots \cup \mathcal{S}_{i,m_i} \hspace{1cm}, \hspace{1cm} \mathcal{S}_{r,1} \cup \mathcal{S}_{r,2} \cdots \cup \mathcal{S}_{r,m_r}
\]

Alternatively, to find the minimal class of sets of rules that can be used to direct a LR(1) parser, find the minimal C-class machine \( A_2' \) for \( A_2 \) and determine from it the sets of rules to which it corresponds.

It is not immediately clear that the procedure described above will produce the results intended. In the first place we need to demonstrate that \( A_2' \) can in all cases be used to direct a parser. \( A_2' \) must for instance have the property that no possible input sequence can result in more than one normal output. Secondly, the set of possible input sequences is in effect restricted to those of the form \( \phi(1) \cdots \phi(t) \) where \( \phi \) is a CSF of \( \mathcal{G} \) such that \( \text{CUT}(\phi) \geq t - 1 \); because of this restriction it is conceivable that some machine smaller than the minimal C-class machine for \( A_2 \) might adequately serve to direct a parser for \( \mathcal{G} \). We must show that this is not the case.

**Theorem 1.** Let \( \mathcal{S} \) be a set of \( k \)-states whose derivation is \( Y_1, ..., Y_t \). Then if \( \xi \in \mathcal{S} \), there exists a CSF \( \phi \) such that \( \phi(1, t) = Y_1 \cdots Y_t \) and \( \xi \) applies to \( \phi(t) \).

**Proof.** If \( \xi \in \mathcal{S} \), it must have a derivation \( \xi_1 = [P_1, j_1; \alpha_1], ..., \xi_m = [P_m, j_m; \alpha_m] \) corresponding to which there exists \( v_1 < v_2 \cdots < v_t \) such that, for \( 1 \leq i \leq t \), \( \xi_{v_i+1} \) is a \( Y_t \)-successor of \( \xi_{v_i} \), whereas for \( 1 \leq i \leq m - 1 \), if \( i \notin \{v_1, ..., v_t\} \), \( \xi_{v_i+1} \) is an immediate successor of \( \xi_i \).

Let \( h_i \) be the number of \( v \)'s < \( i \). Then define CSFs \( \phi_0, ..., \phi_m \) as follows.

Let \( \phi_0 = S_0 \), \( \phi_0, ..., \phi_m \), for \( 0 \leq i \leq m - 1 \), if \( i \in \{v_1, ..., v_t\} \), let \( \phi_{i+1} = \phi_i \). If, on the other hand, \( i \notin \{v_1, ..., v_t\} \), canonically derive \( \phi_{i+1} \) from \( \phi_i \) by first applying productions to \( \phi_i(h_i + 2, \phi_i) \) so as to obtain a terminal string whose head is \( \alpha_{i+1} \). If this results in the derivation of a CSF \( \phi_i \) from \( \phi_i \), then let \( \phi_{i+1} \) be the result of applying the production \( \phi_i(h_i + 1) \) to \( \phi_i \). It is not difficult to show by induction that for \( 1 \leq i \leq m \): (1) \( \phi_i(1, h_i) = Y_1 \cdots Y_{h_i} \); (2) \( \xi_i \) applies to \( \phi_i(h_i) \) and \( \text{CUT}(\phi_i) > h_i \) for \( i < m \) (using the fact that for \( 1 \leq i < m \) \( j_i \) must be \( < \alpha_i \)), whereas \( \text{CUT}(\phi_m) \geq h_m \). Clearly \( \phi_m \) is a CSF of the required kind.

In the theorems and corollaries which follow \( \mathcal{G} \) denotes some LR(1)-grammar, and \( \Sigma = \{\mathcal{S}_0, ..., \mathcal{S}_u\} \) the class of all its derivable 1-state sets.

---

643/17/5-4
The associated finite automaton \( A_z \) and its transition functions \( \delta_z \) and \( \omega_z \) will be written simply as \( A, \delta, \omega \) respectively. After we extend the definition of \( A_z \), it will be clear that these results can be generalized so as to apply to \( LR(k) \) grammars where \( k \neq 1 \).

**Corollary I.** No state \( S_i' \) of \( A \) is degenerate.

**Proof.** By Theorem 1, if \( \zeta \) is any member of \( S_i \) it is applicable to \( \phi(t) \) for some CSF \( \phi \) and number \( t \). Let \( \zeta = [p, j; a] \). If \( \text{CUT}(\phi) = t \), then \( \omega(\phi(t + 1), S_i') = z_p \); otherwise if \( \text{CUT}(\phi) = r > t \),

\[
\omega(\phi(r + 1), \delta(\phi(t + 1) \cdots \phi(r), S_i')) = z_p.
\]

The result follows, and from it we conclude further:

**Corollary II.** The state \( S_* \) of \( A \) is incompatible with any of the other states of \( A \).

**Corollary III.** Let \( \gamma = Y_1 \cdots Y_t \) be any input sequence such that while \( \delta(Y_1 \cdots Y_{t-1}, S_0') \neq S_* \), \( \delta(\gamma, S_0') = S_* \) (thus producing a normal output). Then \( \gamma \) is the head of a CSF whose \( \text{CUT} \) is \( \geq t - 1 \).

**Proof.** Let \( S_i' = \delta(Y_1 \cdots Y_{t-1}, S_0') \). Then clearly since \( \delta(Y_{t-1}, S_0') = S_* \), there must be some \( \zeta \in S_i \) such that \( \zeta \) is of the form \([p, n; \xi] \) for some \( p \). By Theorem 1, \( \zeta \) applies to \( \phi(t - 1) \) for some CSF \( \phi \) such that \( \phi(1, t - 1) = Y_1 \cdots Y_{t-1} \), and in this case \( \phi(t) \) must be \( Y_t \).

**Theorem 2.** Any C-class machine for \( A \) may be used to direct a \( LR(1) \) parser for \( \mathbb{G} \) in place of \( A \).

**Proof.** Let \( D \) be a C-class machine for \( A \) with next-state function \( \delta_D \) and present output function \( \omega_D \). Since \( A \leq D \) we need only show that no input sequence to \( D \) will produce more than one normal output. Let \( \Gamma \) be a compatible set which occurs as one of the blocks of the C-class for the state set of \( A \) from which \( D \) is formed, and let \( \Gamma_D \) be the corresponding including state of \( D \). For any \( Y \), if \( \omega(Y, S_i') \) is undefined for all \( S_i' \in \Gamma \), then so is \( \omega_D(Y, \Gamma_D) \). On the other hand, if for some \( S_i' \in \Gamma \), \( \omega(Y, S_i') = (\text{say})z_p \), then \( \delta_D(Y, \Gamma_D) \) is the state of \( D \) corresponding to a compatible set of \( A \) that includes \( S_* \). According to Corollary II, this compatible set must in fact be \( \{S_*\} \). The result follows.

**Theorem 3.** Any finite automaton \( B \) which can be used in place of \( A \) to direct a \( LR(1) \) parser for \( \mathbb{G} \) must include \( A \).
Proof. Let B’s next-state function be \( \delta_B \) and its present output function \( \omega_B \). Consider the initial states \( \mathcal{S}_0' \) and \( U \) (say) of \( A \) and \( B \) respectively. Clearly if \( Y_1 \cdots Y_t \) is an input sequence which forms the head of a CSF whose CUT is \( \geq t - 1 \) and which is such that \( \omega(Y_t, \delta(Y_1 \cdots Y_{t-1}, \mathcal{S}_0')) \) is a normal output, then so must be \( \omega_B(Y_t, \delta_B(Y_1 \cdots Y_{t-1}, U)) \), and the two outputs must be identical. By Corollary III this implies that \( \mathcal{S}_0' \leq U \). Further, if \( \mathcal{S}_i \) is any state set of \( \Sigma \) with derivation \( Z_1 \cdots Z_t \), then \( \delta(Z_1 \cdots Z_t, \mathcal{S}_0') \) must be \( \leq \delta_B(Z_1 \cdots Z_t, U) \). It follows that \( A \leq B \).

From Theorems 2 and 3 we obtain immediately the following corollary.

**Corollary.** The minimal C-class machine for \( A \) is the smallest finite automaton that can be used to direct a LR(1) parser for \( A \).

Applying the approach of McKeeman (1966), we can provide for variable look-ahead in the following way: If, for some \( p \), a state set \( \mathcal{S}_i \) of \( \Sigma \) consists entirely of states of the form \([p, n_y; Y]\), we can clearly modify the sets of rules associated with \( \Sigma \) by eliminating the rules corresponding to \( \mathcal{S}_i \) and adding instead to the rules corresponding to each state set of \( \Sigma \) of which \( \mathcal{S}_i \) is a \( Y_j \)-successor for some \( j \), a new type of rule of the form “If the next input character is \( Y_j \), first place \( Y_j \) on the stack and then apply reduction \( p \).” In this manner we in effect combine the rules of a LR(0) and LR(1) parser.

If the minimal C-class machine for \( A \) is \( A' \) with next state and present output functions \( \delta' \) and \( \omega' \), respectively, one can without difficulty make corresponding modifications to \( A' \). Let us call a state \( U \) of a machine such as \( A' \) a pure-output state if for some \( p \), \( \omega(Y, U) = z_p \) for all \( Y \) for which \( \delta(Y, U) \) is defined. Eliminate from \( A' \) all pure-output states \( U \). For each state \( V \) such that for some \( Z, \delta'(Z, V) = U \), let \( \delta'(Z, V) \) be instead the state that includes \( \mathcal{S}_0 \) and \( \omega'(Z, V) \) a new “normal” output denoted by \( z_p^0 \), with the interpretation that such an output represents an instruction to stack the next character and then apply reduction \( p \). It is more convenient to eliminate such pure-output states from \( A \) before evaluating \( A' \), but the precaution should then be followed that no state \( U \) should be eliminated if this results in any of \( U \)'s immediate predecessors becoming incompatible in the final modified machine with a state with which it is compatible in \( A \).

It is possible to carry the above approach a further step: for any \( k \), employ outputs \( z_p^k \) having the interpretation “Stack the next \( k + 1 \) characters and then apply reduction \( p \).” In this way we would be incorporating rules from what might be called a LR(\(-k\)) parser.

\(^6\) One can similarly eliminate state sets which have only a single successor and no states of the form \([p, n_y; x]\).
Example. Let $G$ be the grammar

\[
S_0 \rightarrow E \\
E \rightarrow T \mid +T \mid -T \mid E + T \mid E - T \\
T \rightarrow P \mid T^*P \mid T/P \\
P \rightarrow a \mid (E).
\]

Fig. 2. Production $P$ represents $z_p$. Production $P$ represents $z_p^0$. 
Then $A$ is represented by the transition diagram given in Fig. 2. To make the diagram easier to interpret, we have substituted the reduction concerned in place of the outputs $z_{p}$ and $z_{p}^{0}$, with the reduction encircled in a pointed or square box respectively. States are represented simply by encircled numbers, and the special state $S_{k}$ is omitted. Not counting $S_{k}$, $A$ has 28 states. The minimal $C$-class machine for $A$ given in Fig. 3 however has only 7 states:

![Transition Diagram](image)

Note that, using the results of McClusky (1962), Unger (1965), and Pager (1970b), one can show for most grammars, including that employed in the above example, that there is a minimal $C$-class which consists of maximum compatible sets.
The adaptation required of the definition of $A_\sigma$ for LR(0) grammars is trivial. For LR($k$) grammars where $k > 1$, the finite automaton model has to be modified. For each state of the form $[p, j; \alpha]$ introduce a new input symbol $Y_\alpha$. If $\mathcal{S}_i$ is a derivable set of states which contains a state $\xi = [p, n_p; \alpha]$ for some $p$, define $\omega_\sigma(Y_\alpha, \mathcal{S}_i')$ to be $x_p$. If $\phi$ is the CSF currently being parsed and $\phi(t + 1)$ the head of the input stack, and $\xi$ is applicable to $\phi(t)$ and $\alpha = \phi(t + 1, t + |\alpha|)$, then in these circumstances we consider that the special input $Y_\alpha$ occurs, and so produces output $x_p$. In a model $A_\sigma$ of this kind, a state such as $\mathcal{S}_i'$ must be regarded as being incompatible with any state $\mathcal{S}_j'$ of $A_\sigma$ for which $3z(\alpha, Y'_\alpha)$ is defined. This affects the maximum compatible sets we initially form before applying the algorithms for finding the minimal $C$-class, but clearly the algorithms themselves remain applicable, and the $C$-class machine obtained from the result is the required minimal machine.

Minimizing the number of sets of states in the manner described above leads to a very substantial savings of space, but there are certain qualifications to this savings and the benefit it brings which should be noted. Depending on how the LR($k$) algorithm is coded, some combinations of states, such as perhaps that of a pure-output state with some other state, might occupy more space than the uncombined states. If this is the case, the states entering into each combination should first be checked. To avoid accepting a non-sentence, it is necessary in some circumstances to verify that the top of the stack contains the expected RHS before making a reduction, or, if outputs of the form $x_p x$ are being employed, that the head of the input stream contains the expected $k + 1$ characters. When one combines state sets, error detection in general becomes less precise. It is, however, possible to add extra sets of rules for error-checking without making a significant inroad into the spatial economy that minimizing the number of state sets produces. There are, furthermore, circumstances where little error detection is required, such as in multipass compilers that do most of their error-checking in the first pass and generate code in the later ones. Also an increasing use is being made of specialized compilers, where one uses a compiler with very wide diagnostic abilities during debugging, and a separate highly optimized one that does little error-checking for production runs. Quite large grammars including e.g. ALGOL can be handled directly by computer; however if the number of state sets generated by Knuth’s algorithm is too large, our minimalization procedure can instead be applied to the (in general) smaller class of state sets obtained by the partitioning method of Korenjak (1969).
There is room for further research in improving and producing variants of the basic LR(k) algorithm. Consider, for example, the grammar

\[ S \rightarrow ba_1 \cdots a_n \quad \text{for some } n. \]

\[ S \rightarrow ca_1 \cdots a_n \]

Knuth's algorithm is inefficient here because it "remembers" whether \( b \) or \( c \) was the first symbol scanned by going into a distinct set of states in each case. An obvious alternative is to set a flag indicating whether \( b \) or \( c \) was the first symbol scanned, and to check this flag when \( \mid \) is later read so as to determine which reduction should be made.

Appendix

Knuth's Notation (1965)

Greek letters denote strings, the letters \( a, b, c \) denote terminal characters, and \( X, Y, Z \) denote either terminals or nonterminals. \( \alpha \rightarrow \beta \) means \( \beta \) is an immediate consequent of \( \alpha \). \( \alpha \Rightarrow \beta \) means \( \beta \) is derivable from \( \alpha \). \( \alpha \Rightarrow \beta \) means \( \alpha = \beta \) or \( \alpha \Rightarrow \beta \). \( H_k(\phi) \) denotes the set of strings \( \{ \omega \mid \omega \text{ is terminal, } |\omega| = k, \text{ and there exists some } \psi \text{ such that } \phi \Rightarrow \omega \psi \} \). Knuth considers a hypothetical grammar \( \mathcal{G} \) whose productions are expressible in the form

\[ A_p \rightarrow X_{p1}X_{p2} \cdots X_{pn_p} \quad \text{for } 1 \leq p \leq s \]

and whose goal symbol is \( S \). \( S_0 \) and \( \mid \) are symbols new to \( \mathcal{G} \). A state is a triple \([p, j; \alpha] \) such that \( 0 \leq p \leq s, 0 \leq j \leq n_p \), and \( \alpha \) is terminal. If \( \phi A_p \sigma \) is a sentential form where \( \sigma \) is terminal, then the handle of its consequent \( \phi X_{p1} \cdots X_{pn_p} \sigma \) is the pair of numbers \((|\phi X_{p1} \cdots X_{pn_p} |, p)\).

Finite Automata Terminology Employed in Booth (1967)

A finite automaton is designated by a 5-tuple \( \langle I, Q, Z, \delta, \omega \rangle \) where \( I \) is the input alphabet, \( Q \) is the state set, \( Z \) is the output alphabet, \( \delta \) is the next-state function, and \( \omega \) is the present-output function. A state \( q \) of a machine \( A \) is included by a state \( u \) of a machine \( H \), \( q \leq u \), if every input sequence which leads to a final output starting with state \( q \) of \( A \) also produces that output starting with state \( u \) of \( H \). A machine \( A \) is included by a machine \( H \), \( A \leq H \), if every state of \( A \) is included by some state of \( H \). A \( C \)-class for a finite automaton is what is usually referred to as a closed cover for its states (i.e., \( C \) is a set of subsets \( \{B_1, \ldots, B_t\} \) of the automaton's state set \( Q \), such
that (1) each $B_i$ is a compatibility set, (2) $\bigcup_{i=1}^r B_i = Q$, (3) no $B_i$ properly includes another, (4) for each $Y$ and $i$, $\delta(Y, B_i) \subseteq B_j$ for some $j$; here $\delta(Y, B_i)$ means $\{\delta(Y, q) \mid q \in B_i\}$. The C-class machine corresponding to a C-class for a finite automaton $A$ is the finite automaton $H$ constructible by well known techniques from the C-class, such that $A \sim H$. (viz: if the C-class is $\{B_1, \ldots, B_r\}$ and $A = \langle I, Q, Z, \delta, \omega \rangle$, then $H = \langle I, \{B_1', \ldots, B_r'\}, Z, \delta_H, \omega_H \rangle$ where, for each $Y$ and $i$, if $\delta(Y, B_i) \neq A$, $\delta_H(Y, B_i') = \text{some } B_j'$ such that $\delta(Y, B_i') \subseteq B_j$; and if $\omega(Y, q) = z$ for some $q \in B_i$, then $\omega_H(Y, B_i') = z$; no other values for $\delta_H$ and $\omega_H$ are defined.)

Received: October 13, 1969; Revised: June 15, 1970

References

