Finite State Automata from Regular Expression Trees

ROBERT R. GOLDBERG

Department of Computer Science, Queens College of CUNY, 65–30 Kissena Boulevard, Flushing, NY 11367-0904, USA

Thompson [13] introduced an innovative method for obtaining non-deterministic finite state automata (*nfa*) from regular expressions. His formulation of *nfa*s makes use of ε-transitions (null symbol input) and requires in the worst case $2\sigma + 2$ OPS states, where σ is the number of occurrences of alphabet symbols and OPS is the number of operands in the original regular expression. We modify this algorithm to obtain a *nfa* M without ε-transitions that has in the worst case $\sigma + 1$ states. Using multi-branch expression trees to store the regular expressions efficiently, the algorithm presented here is directly parallelizable. The algorithm necessitates that we maintain a finite state automata which has no ε-transitions and has a starting node of zero in-degree. The role of ε-transitions in finite state automata is examined and, based on the technique of *bypassing* [11], two alternative approaches are suggested. Three advantages manifest themselves during the improved construction: many fewer nodes are needed in the equivalent finite state automata; ε-transitions are not needed; the construction is highly parallelizable. With the advent of parallel processing, *nfa*s can be simulated by multiple processors or in VLSI design [14]. The algorithm presented here results in the creation of a *nfa* which requires less hardware resources to simulate.

Received January 1993, revised June 1993.

1. INTRODUCTION

Kleene [9] introduced the notion of a regular set for describing regular languages (type 3 of the Chomsky hierarchy) and showed that a finite state automata can be constructed to accept exactly the words of a particular regular language. This string-based approach allowed for the concise expression of structures from many different application domains. The diverse areas which can benefit from these results include compiler theory [1], computational linguistics [7], constructive solid geometry [15], pattern recognition [6], program translation [3], switching theory [10] and VLSI design [14]. However, while the proof of Kleene's Theorem was constructive, the related construction methods were not efficient.

The classical Thompson construction [5] provides an easy method for obtaining non-deterministic finite state automata (*nfa*) from regular expressions by using ε-transitions, transitions between states of the machine which do not absorb input. His algorithm introduced the basic framework for such transformations based on an innovative use of recursion. However, the ε-transitions are not easily implementable and their use causes an 'unnecessary' number of extra states. In addition, because the regular expression input was assumed to be stored in a linear array, the algorithm was not directly parallelizable.

The regular expression input in this paper is assumed to be stored in expression trees [12]. These expression trees have the alphabet symbols as the leaves of the tree and the regular operators stored in the interior nodes. This data structure was originally suggested for efficiently storing regular expressions and is based on the common method of storing arithmetic expressions, which resemble regular expressions. We extend this structure so that it is a multi-way tree, and not strictly binary. Hence, $a \cup b \cup c \cup d$ would be represented by the tree in Figure 1(a) and not, for example, as in Figure 1(b). This introduces the possibility of further parallelization of the algorithm because the operators now act on multiple operands at the same time and the operators treat the sub-operands individually. A bottom-up approach would first deal with the symbols of the regular expression (the leaves). The results obtained at one level then act as operands for the next level operators, and we recurse towards the root.

In the search for a reduction in the number of states required, the role of ε-transitions in finite state automata is examined. The familiar techniques of *bypassing* (discussed in Section 3.2) [11] suggests two alternative approaches. A substantially improved method is presented for obtaining a *nfa* that utilizes in the worst case no more than $\sigma + 1$ states to recognize a regular expression with σ operands. The method requires that we recursively maintain a finite state automata, with no ε-transitions, and in which the starting node has in-degree zero. With the advent of parallel processing, *nfa*s can be simulated by multiple processors or by a suitable VLSI design [14]. The algorithm presented here can reduce the resources necessary to simulate *nfa* machines.

In the next section of the paper, we introduce the

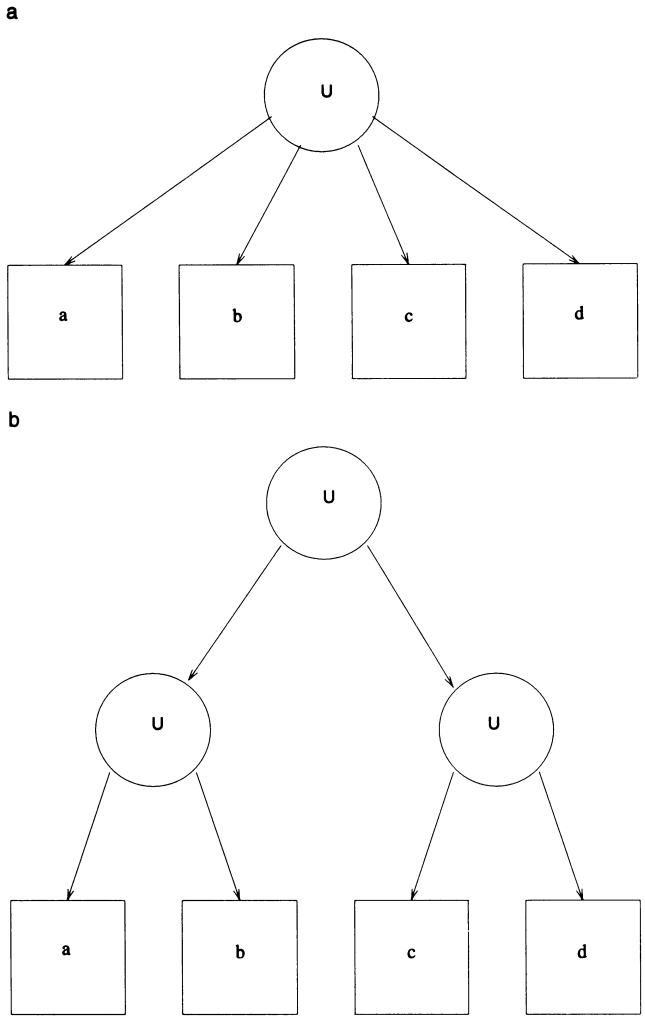

FIGURE 1. (a) Multiway expression tree for $a \cup b \cup c \cup d$ and (b) standard regular expression tree for the same.

classical Thompson construction based on the regular expression tree input. This provides the background for the presentation of the new algorithm. Both algorithms are presented recursively, dealing first with the base cases and then with the union, concatenation and Kleene star operators. The bypassing technique for removing ε-transitions [11] will be needed in the operator cases. Utilizing the notion of predecessor and successor states, two alternative bypassing transformations are presented and used to deal with concatenation. Bypassing to successor states will allow for the extra states incurred by the Thompson construction to be removed, whereas bypassing from predecessor states is not always applicable, but is time consuming and does not affect such changes. A similar point applies to the Kleene star operator which combines concatenation and union. Finally, two additional modifications are presented which further improve the algorithm by lowering the number of states required by the *nfa*. These involve merging the final states which have out-degree zero and, a special case in which bypassing from the predecessor states actually does yield a reduction in the number of states.

2. THE THOMPSON CONSTRUCTION WITH ε-TRANSITIONS

Thompson observed that any finite state automata can be transformed to a *nfa* with one start node and one final node provided that ε-transitions (Figure 2) are allowed. In addition, the initial node of the new *nfa*-ε has in-degree of zero and *the* final node of the new *nfa*-ε has out-degree of zero. The base cases of this approach appear in Figure 3(a–c) and require two nodes for each symbol in $\varnothing, \varepsilon, \Sigma$. Using the same approach two arbitrary automata could be combined under the regular operators $\cdot, \cup, *$ and the resultant machine will be a *nfa*-ε (Figure 4a–c). Thus, recursively a *nfa*-ε can be obtained from a regular expression. This would require

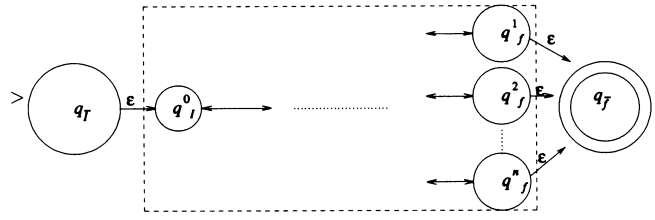

FIGURE 2. *Nfa*-ε conforming to its restrictions.

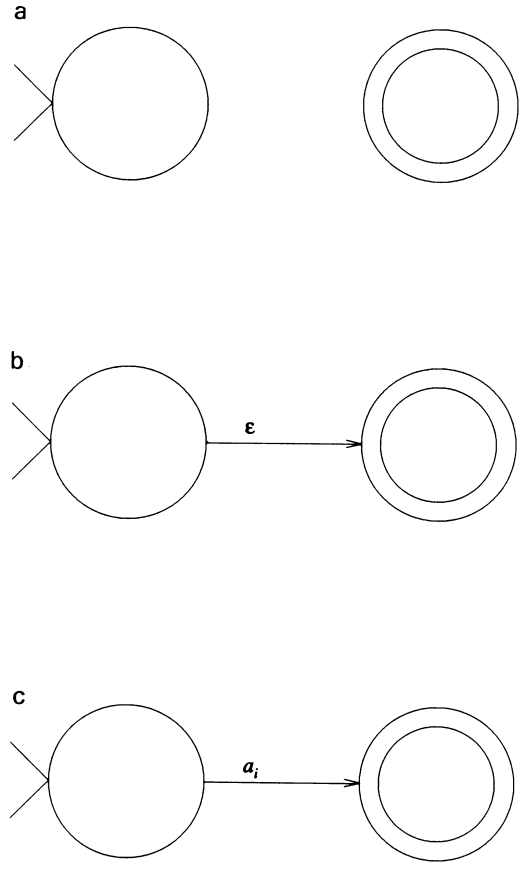

FIGURE 3. Base cases for the recursive algorithm for creating a *nfa*-ε from a regular expression stored in a binary expression tree. (a) $L(M) = \varnothing$. $\mathbf{F} \overline{\subset} \mathbf{Q}$. (b) $L(M) = \{\varepsilon\}$. (c) $L(M) = \{a_i\}$.

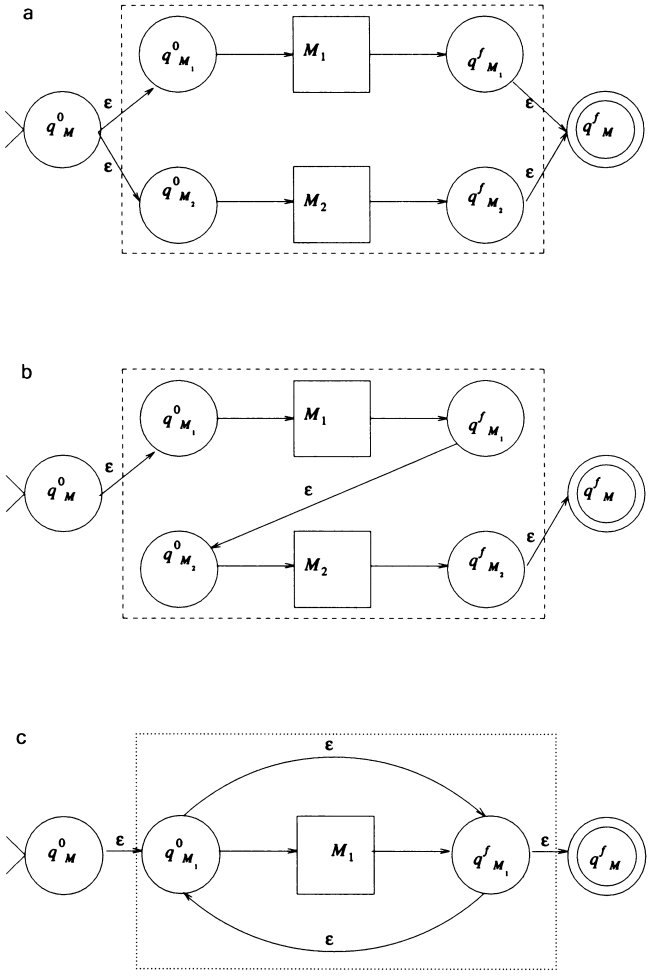

FIGURE 4. Thompson construction for combining arbitrary finite state automata under the regular operators. The resulting machines have one initial node with in-degree zero and one final node with out-degree zero. (a) *Nfa-ε* for $L = L(M_1) \cup L(M_2)$. (b) *Nfa-ε* for $L = L(M_1) \cdot L(M_2)$. (c) *Nfa-ε* for $L = L(M_1)^*$.

two additional nodes for each operator. The closure of this algorithm is obtained by the finite recursion necessary for the traversal of the expression tree.

The above algorithm is a simple way of constructing finite state automata from regular expressions. However, it requires $2\sigma + 2\mathbf{OPS}$ states (where the regular expression contains σ occurrences of symbols and **OPS** operations). Another problem is that this approach requires ε-transitions; it is thus harder to determine which states are entered, upon transition. An example of Thompson's construction for the regular expression $(a \cup b)^* \cdot (c \cup d)$ is given in Figure 5(a).

Consider a regular expression tree from user input (as in Figure 1a). Using the expression tree constructed, the Thompson algorithm, based on the above discussion, may easily be implemented to obtain the non-deterministic finite state automata *nfa-ε* equivalent to the inputted regular expression. This is accomplished by doing a simple traversal of the tree. (As presented here this algorithm will be recursive. A non-recursive approach would implement a post-order traversal of the expression tree since the values of the children subtrees are required to process the parent.)

3. AN IMPROVED CONSTRUCTION WITHOUT ε-TRANSITIONS

The following observations allow the modification of the algorithm which will produce an *nfa* without ε-transitions:

1. Any algorithm that uses non-deterministic automata requires first the calculation of the equivalent transitions without the ε-transitions.
2. The previous construction adds unnecessary extra nodes in order to facilitate the ε-transitions.

3.1. Base cases

We now modify this algorithm so that no ε-transitions are needed, which will avoid adding a number of nodes. The base case (Figure 3a–c) is similar to the Thompson construction but ε-transitions are not needed. The empty language could be represented by a single non-final node. $L = \{\varepsilon\}$ could be accepted by a single node as initial/final. Thus, a slight modification will be required in Figure 3(a) and (b). Then, the base cases for $\varnothing$ and $\{\varepsilon\}$ now have one symbol and only need one node (which will also be the final node if $L = \{\varepsilon\}$). For the case of a single symbol a_i of Σ, the alphabet, we have one symbol and two states (Figure 3c). Therefore, for all three base cases we have that the number of nodes (states) required by the *nfa* construction is at most $\sigma + 1$, where σ is the total number of occurrences of alphabet symbols occurring in the input.

The rest of the algorithm and proofs will be presented in the following sections, continuing inductively for the regular operators. This recursive approach was the innovation of the above Thompson method. In order to achieve an *nfa* with at most $\sigma + 1$ states efficiently, we will need the fact that the input is stored in a multi-way expression tree. As a practical implementation, using standard tree construction algorithms from infix expressions, it can be shown that the construction of the regular expression tree can be obtained in one pass of the input.

3.2. Union

Consider the union of the m sub-machines $L(M) = \cup_{k=1}^{m} L(M_k)$, as in Figure 6 (all nodes except q^0_{new} and its incident arcs). The direction that the ε-transitions follow from (q^0_{old}) is only one-way to the initial nodes of the individual sub-machines. Consider machine M_k with its original initial node $q^0_{M_k}$. Let q^j_k be a node that the initial node reaches by some symbol σ_{ik} of the alphabet. Then, the regular expression represented by the path from the new initial node q^0_{new} in Figure 6 followed by the transition from $q^0_{M_k}$ to q^j_k is precisely $\varepsilon \cdot \sigma_{ik} = \sigma_{ik}$. Therefore, instead of having m ε-transitions from node q^0_{old} to the individual starting nodes of each

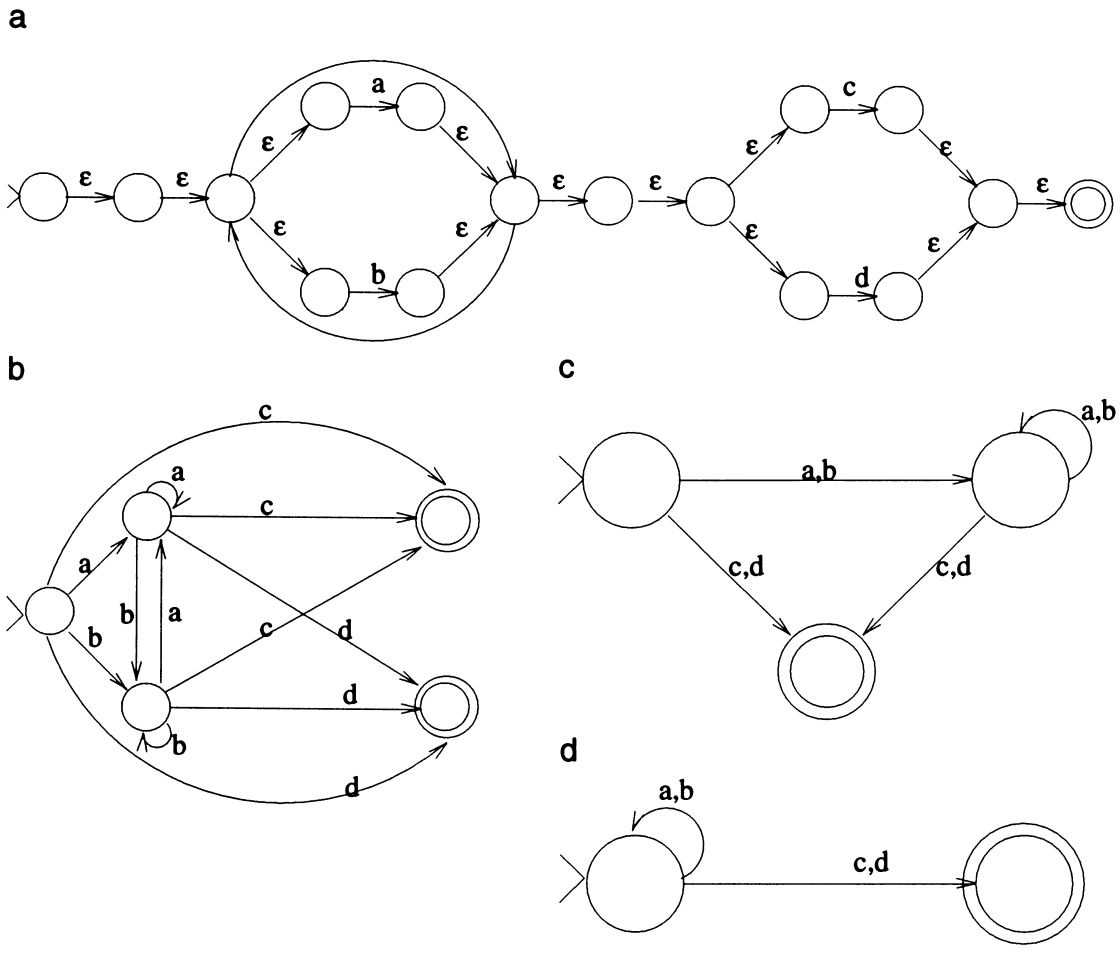

FIGURE 5. Automata for the regular expression $(a \cup b)^* \cdot (c \cup d)$. (a) Original Thompson construction with ε-transitions; (b) without ε-transitions; (c) while also merging final nodes in union operation; (d) improved Kleene star procedure.

of M_k, we can have the new initial node (q^0_{new}) have m arcs pointing to the nodes following these initial nodes in their respective machines with the corresponding symbols σ_{ik}. This *bypassing* operation was used successfully in obtaining regular expressions from finite state automata [11]. (See Figure 7 for an illustrative example of equivalent bypasses.) Since, by induction, the zero in-degree of these respective (former) initial nodes was maintained, then those nodes can safely be removed; the new initial node (q^o_{new}) and its associated arcs now encompasses all of the required transitions. Thus, the resultant machine will have the same language.

Suppose that σ_i was the number of alphabet symbols in the regular expression that M_i was constructed from. Then, inductively, the number of states in M_i so far is $\sigma_i + 1$. In total for m machines we have $\Sigma_{i=1}^{m}(\sigma_i + 1) = (\Sigma_{i=1}^{m}\sigma_i) + m$. We bypass the m initial nodes (removing them) and add one new initial node, thus a savings of $m - 1$ nodes. Hence we subtract $m - 1$ from the previous total and we obtain the total number of nodes in this conglomerate machine M, i.e.

$$1 + \left(\sum_{i=1}^{m} \sigma_i\right)$$

Thus, this operation preserves the required number of nodes for the *nfa*. If $\varepsilon \in L(M_k)$, then the new initial node will be final. If $L(M_k) = \{\varepsilon\}$, the associated sub-machine M_k has a single node which is also final.

3.3. Concatenation

Let us now consider the concatenation operator which will be stored as a roof of a subtree with possibly many children as in $L(M_1) \cdot L(M_2) \cdot \cdots \cdot L(M_m)$. The associated regular expression tree contains the top levels as depicted in Figure 1(a) but with concatenation as root. Thompson's construction would add two new nodes with ε-transitions for every consecutive pair of submachines (Figure 4b). However, by performing bypass operations with extra arcs as explained in the union case, ε-transitions are not needed and the total number of nodes will be reduced.

Consider Figure 7(a). The ε-transition creates a path from those nodes entering into the final nodes of machine M_{i-1} followed by the ε-transition from the final nodes of machine M_{i-1} to the initial nodes of machine M_i. There are two ways of avoiding the ε-transitions (Figure 7b and c). The first approach (Figure 7b) simply adds an arc from the predecessors of the final nodes of M_{i-1} to the initial nodes of machine M_i with the

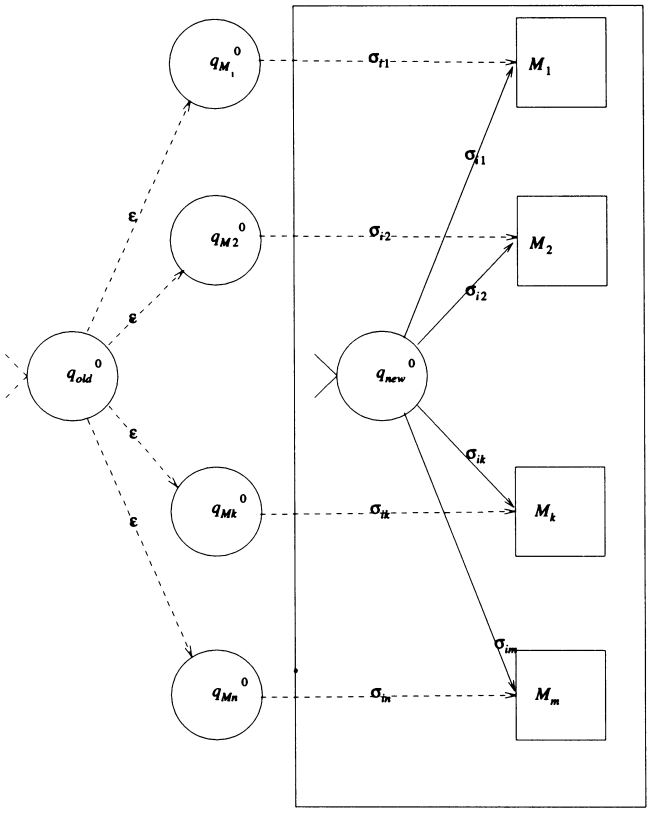

FIGURE 6. Thompson construction with ε-transitions for union of n machines in parallel, $L = L(M_1) \cup \cdots \cup L(M_m)$. This original automata includes all nodes and arcs except q^0_{new} and its incident arcs. The automata in the enclosed box represents an equivalent machine bypassing the old initial nodes by direct arcs from a new initial node (q^0_{new}) to successors.

corresponding symbols. However, this approach has two problems associated with it:

1. Obtaining the predecessors of the final nodes could be costly.
2. The initial node of M_i cannot be removed; this would worsen the tight upper bound.

Consider (Figure 7c) the path from the final nodes of M_{i-1} through the ε-transition to the successors of the initial node of machine M_i. This suggests that the arcs from M_{i-1} leave the final nodes of M_{i-1} and not its predecessors. Then these transitions, with the corresponding alphabet symbols, go to the nodes originally pointed to by the initial node of machine M_i. Applying these bypassing operations overcomes the first problem. The second problem is also solved. We can now remove the initial node of machine M_i, because by induction the in-degree of that node is zero. Fortunately, in the constructions till now we have preserved that property. Based on the above process, the only final states will be those of the last machine in the concatenated list, M_m, and possibly the first initial node if $\varepsilon \in L(M_1)$.

However, the algorithm must carefully handle the possibility that an individual submachine M_i has $\varepsilon \in L(M_i)$. Consider $L = L(M_{i-1}) \cdot L(M_i) \cdot L(M_{i+1})$ and $\varepsilon \in L(M_i)$. The initial node of machine M_i will be removed

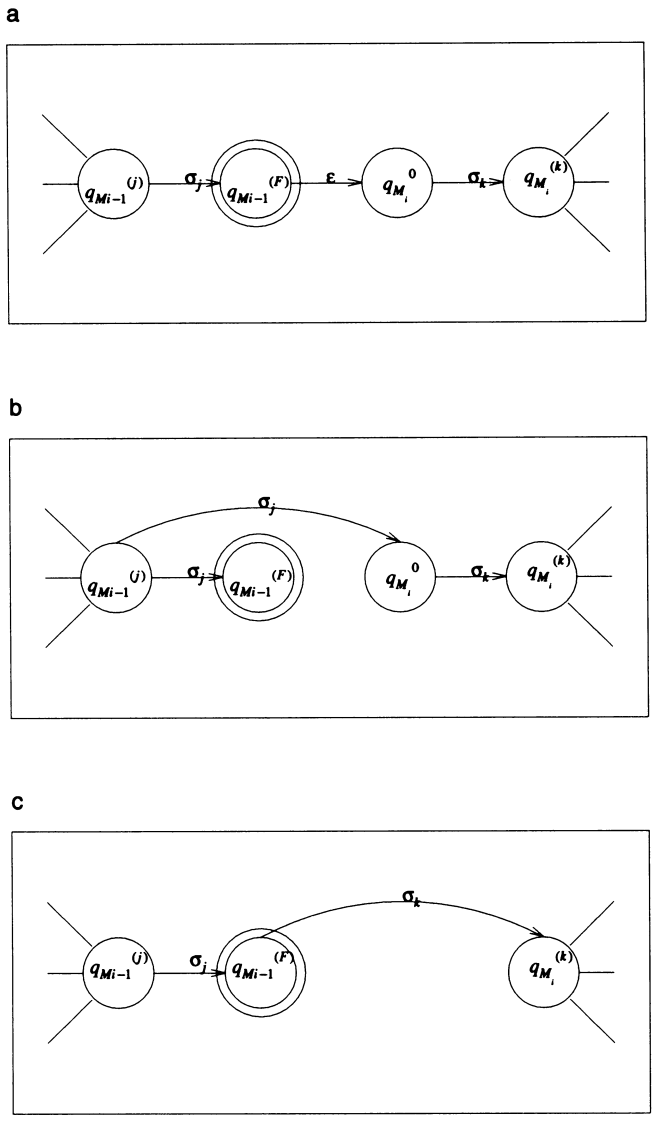

FIGURE 7. Two alternate views (b and c) of bypassing the ε-transitions from the Thompson construction of concatenation (a). Notice that whereas both (b) and (c) do not need the ε-transition, approach (c) removes the initial node of the second machine if zero in-degree. Both approaches can be done in parallel for every pair of concatenated machines A, B.

and ε then will not be accepted by M_i because of the concatenation process above. This causes a problem since a non-final initial node will force the transitions to always pass through machine M_i even though $L(M_{i-1}) \cdot L(M_{i+1}) \subseteq L$, without M_i. This problem could be generalized to a list of machines $M_1 \ldots, M_i, \ldots, M_j, \ldots, M_m$ whose languages will be concatenated. Assume that $M_i, M_{i+1}, \ldots, M_j$ all accept the null string ε, but that machines $M_1, \ldots, M_{i-1}$ and machines $M_{j+1}, \ldots, M_m$ do not. The situation can easily be remedied by the following two step process:

1. Since M_{j+1} does not accept the null string, the input symbols must pass through machine M_{j+1} to reach the final states; thus, none of the former final states

from the individual machines $M_i, \ldots, M_j$ should remain final in the conglomerate construction;

2. Perform the above construction (Figure 7c) for every consecutive pair of machines $M_k, M_{k+1}, i \leqslant k \leqslant j-1$ in the sublist $M_i, \ldots, M_k, \ldots, M_j$; this adds arcs from the former final nodes of machine M_{k-1} to the successor nodes of the initial node of machine M_{k+1} with the corresponding symbols.

If the last k machines of the concatenated list $M_{m-k+1}, \ldots, M_m$ accept ε, then the final nodes of those individual machines that must remain final are those that were not also initial nodes of those same machines. Thus, in all cases, the construction does not require the former initial nodes of the individual machines $M_2, \ldots, M_m$ to remain final. Therefore, these initial nodes can now be removed.

Applying the above procedure, if σ_i is the number of symbols in the regular expression that machine M_i was based on, then prior to applying the concatenation we have $(\Sigma_{i=1}^{m} \sigma_i) + m$ nodes. Now, after applying these concatenation operations to the successive machines, we can remove the $m-1$ former initial nodes $(q^0_{M_2}, \ldots, q^0_{M_m})$, yielding again the required number of nodes as in the union case:

$$1 + \left(\sum_{i=1}^{m} \sigma_i \right)$$

This construction preserves then the stated number of nodes for the *nfa*.

3.4. Kleene star

Let us now consider the Thompson construction of the Kleene star operation case. Although this operator is unary, it causes problems in our approach because of the looping back of the final nodes to the former initial nodes (see Figure 4c). A simple reflection on the definition of the Kleene star operator will help us solve this problem. (See Figure 8 for an illustrative example for the following discussion.)

$$L^* = \sum_{i=0}^{\infty} L^{[i]}$$

where $L^0 = \{\varepsilon\}$, $L^1 = L$, and $L^{[i]} = L \cdot L \cdot \cdots \cdot L$, i times. Instead of bypassing the final node (Figure 8b), we can use the bypassing technique similarly, as we performed in the concatenation case (Figure 7c), i.e. add arcs from the final nodes of this machine to the successors of the initial node of this machine (Figure 8c). This preserves the property that the in-degree of the initial node is zero. Since $L^0 = \{\varepsilon\}$, the former initial node will also be a final node. However, this approach does not add any new nodes. Therefore if the machine has the correct number of nodes, presently at most $\sigma + 1$, then after this operation it will still have the same required number of nodes.

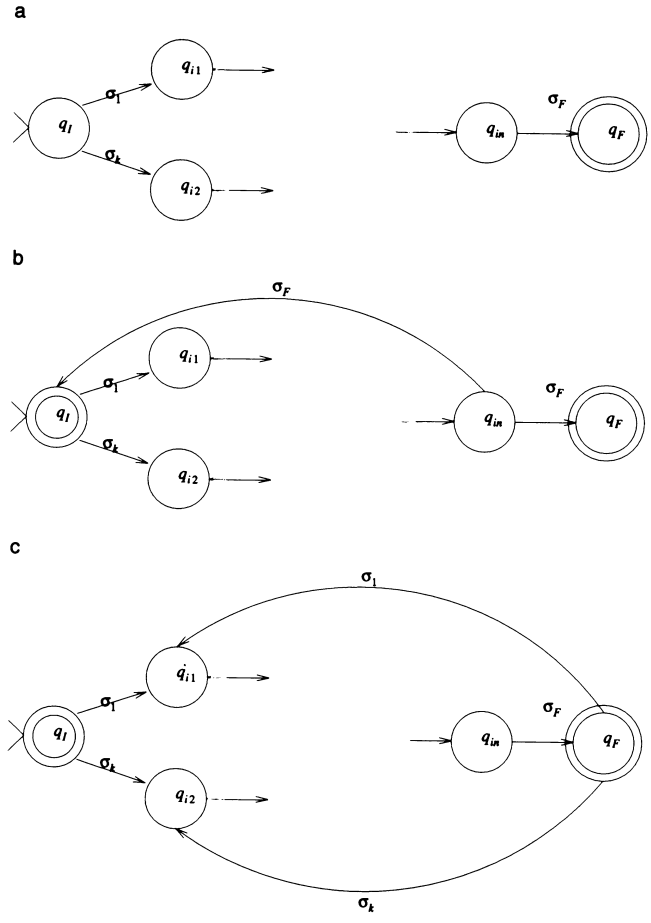

FIGURE 8. Kleene star operation M^*. (a) Original machine M. (b) M^* bypassing the final node. (c) M^* bypassing the start node.

4. FURTHER IMPROVEMENTS

Consider again the regular expression $(a \cup b)^* \cdot (c \cup d)$. The original Thompson construction resulted in the machine with 16 states, depicted in Figure 5(a). Applying the improved algorithm of the previous section results in a *nfa* with five states (Figure 5b). A number of nodes can be removed in the union case if the out-degree of the final states of the individual machines is zero. In this case, after creating the initial node and applying the union process we may merge together all final nodes that have out-degree zero. This suggestion will improve the results of the algorithm by obtaining a further reduction of two nodes. The automaton obtained is that of Figure 5(c).

A final source of reduction is in the Kleene star operator case. Consider $L = L_1^* \cdot L_2 \cdot \cdots \cdot L_m$ and its associated *nfa* M obtained by the above algorithm. In constructing the submachine M_1^* corresponding to the application of Kleene star from M_1, the bypassing algorithm added arcs from the final nodes of M_1 to the successors of its initial node, leaving the in-degree of the start node of M zero. This was the preferred method above instead of bypassing from the final node's predecessors. In this case, however, we should consider the first bypass operation as depicted in Figure 7(b). Here,

the predecessors of the final nodes go to the initial node, which is then final. Now all zero out-dgree final nodes of M_1 may be removed resulting in less nodes. An example of this occurs in Figure 5(d), resulting in a machine with two states, a further improvement over the above (Figure 5c). In total, the complete algorithm reduced the number of states necessary for the language expressed by $(a \cup b)^* \cdot (c \cup d)$ from Thompson's 16 states to our two states.

As described in the Introduction, the implementation presented here assumed that the regular expression was stored in a multi-branch expression tree. This stores the operators of the regular expression in the interior nodes and the operands in the leaves. In general, algorithms performed on expression tree structures can be done somewhat in parallel because the operator can usually be individually applied to each of the operands. This is certainly the case for the regular operators in a regular expression. The union operation involves adding arcs from the new initial node which may be performed in parallel to all of the individual submachines. Similarly, the concatenation operation adds arcs from the former final nodes in each consecutive pair of the concatenated list of machines; this can also be done efficiently in parallel. In both of these cases, the removal of the former initial nodes can be done in parallel after adding in the above arcs. Finally, if enough processors are available, each operator at a given level of the expression tree (same parenthesis level of regular expression input) can be applied to its operands in parallel because each operator behaves independently of the others.

We could modify the above algorithm to lower the number of final states required by any *nfa*. The union operator singularly causes the resultant machine to have the same number of final states as of the individual submachines. However note that in the union case (Section 3.2) if some final nodes have nonzero out-degree (from the Kleene star operation) and at least one final node has zero out-degree then we could obtain a single final node from this operation by having the predecessors of the final states with non-zero out-degree point to *the* final node with out-degree zero. Thus, all prior final states can be merged into one state. If all final nodes have nonzero out-degree (resulting from terms ending with a Kleene star operation), then a new node could be added as the only final node in a similar manner. For example in Figure 9, $a_1^* \cup a_2^* \cdots \cup a_{m-1}^* \cup a_m^*$ (Figure 9A1) would require one additional node (Figure 9A2) but $a_1^* \cup a_2^* \cdots \cup a_{m-1}^* \cup a_m$ (Figure 9B1) would not (Figure 9B2). Although this actually adds

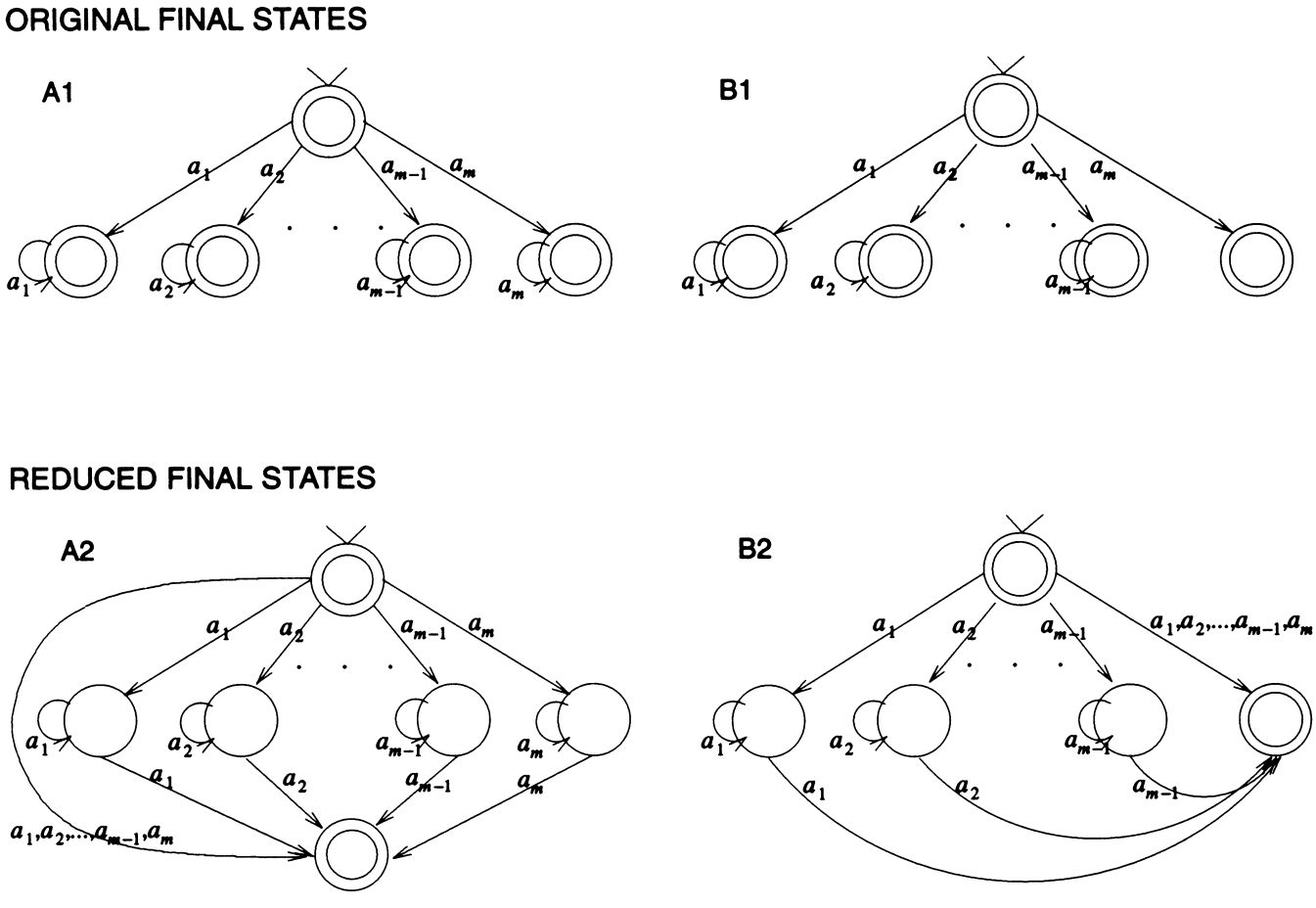

FIGURE 9. Machines for the languages $a_1^* \cup \cdots \cup a_m^*$ (machine A) and $a_1^* \cup \cdots \cup a_m$ (machine B), before (A1, B1)/after (A2, B2) reducing the number of final states. Thus, machine A requires an additional state, while machine B does not.

states, sometimes it is more important to reduce the size of F, the set of states with final status rather than the size of Q.

5. CONCLUSIONS

We presented an improved version of Thompson's construction for obtaining a *nfa* from regular expressions, stored in a multi-branch tree structure. It is demonstrated that this approach does not require ε-transitions. Instead, bypassing to successor states by adding extra arcs allows for the removal of the initial nodes of the individual sub-machines. At most $\sigma + 1$ states are required, where σ is the number of times alphabet symbols appear as operands in the regular expression. The number of states required by our approach is a fraction of what is needed by Thompson.

The acceptance status of the initial node (whether or not final) and its in-degree (whether or not zero) will respectively depend on whether $\varepsilon \in L(M)$ and where the Kleene star operator appears in the regular expression. Similarly, the number of final nodes (one or more) and their out-degree will depend on whether the Kleene star operator appears in the regular expression input. If $\varepsilon \notin L(M)$, this *nfa* requires only one final node, with an out-degree of zero. Since the construction of *nfa*s from regular expressions has many applications in areas that require pattern matching and parsing [1, 3, 6, 7, 10, 13, 14], the savings gained here (in terms of space and time) could have practical applications in those areas.

ACKNOWLEDGEMENTS

The author would like to thank Steven J. Ohsie and Dr Jacob Shapiro, and the anonymous reviewers for their useful comments.

REFERENCES

[1] A. V. Aho, R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, MA (1988).
[2] J. G. Brookshear, *Theory of Computation: Formal Languages, Automata and Complexity*. Benjamin/Cummings, Redwood City, CA (1989).
[3] P. Calingaert, *Program Translation Fundamentals: Methods and Issues*. Computer Science Press, Rockville, MD (1988).
[4] M. Davis and M. Weyuker, *Computability, Complexity and Languages*. Academic Press: New York, NY (1983).
[5] P. J. Denning, J. B. Dennis and J. E. Qualitz, *Machines, Languages and Computations*. Prentice Hall, Englewood Cliffs, NJ (1978).
[6] S. Fu, *Syntactic Pattern Recognition*. Prentice Hall, Englewood Cliffs, NJ (1982).
[7] M. D. Harris, *Natural Language Processing*. Reston Publishing, Reston, VA (1985).
[8] E. Horowitz and S. Sahni, *Fundamentals of Data Structures Using Pascal*. Computer Science Press, Rockville, MD (1984).
[9] S. C. Kleene, Representation of events in nerve nets. In *Automata Studies*. Princeton University Press, Princeton, NJ (1956).
[10] S. C. Lee, *Modern Switching Theory and Digital Design*. Prentice Hall: Englewood Cliffs, NJ (1978).
[11] T. Sudkamp, *Language and Machines*. Addison-Wesley: Reading, MA (1989).
[12] S. Thatcher, Tree automata. In *Current Trends of Computing*, Vol. 4. Prentice Hall, Englewood Cliffs, NJ (1973).
[13] K. Thompson, Regular expression search algorithm. *Communications of the ACM*, **11**, pp. 419–422 (1968).
[14] J. D. Ullman, *Computational Aspects of VLSI*. Computer Science Press, Rockville, MD (1984).
[15] H. Voelcker and G. Requicha, Constructive solid geometry. In *Information Sciences*, Vol. 8, 1984. Tou (ed). Plenum Press, Chicago, IL.