Implementation of State Elimination Using Heuristics

Jae-Hee Ahn[1] and Yo-Sub Han[2]

[1] NHN Corporation, Korea
jaehee.ahn@nhncorp.com
[2] Department of Computer Science, Yonsei University, Korea
emmous@cs.yonsei.ac.kr

Abstract. State elimination is an intuitive and easy-to-implement algorithm that computes a regular expression from a finite-state automaton (FA). The size of a regular expression from state elimination depends on the state removal sequence. Note that it is very hard to compute the shortest regular expression for a given FA in general and we cannot avoid the exponential blow-up from state elimination. Nevertheless, we notice that we may have a shorter regular expression if we choose a good removal sequence. This observation motivates us to examine heuristics based on the structural properties of an FA and implement state elimination using the heuristics that run in polynomial time. We demonstrate the effectiveness of our algorithm by experiments.

1 Introduction

It is well known that finite-state automata (FAs) have the same expressive power as regular expressions [11]. This well-known statement is proved by showing that we can construct FAs from regular expressions and that we can compute regular expressions from FAs. There are many algorithms for constructing FAs and obtaining regular expressions [5,12,16]. Note that we construct a linear-size nondeterministic finite-state automaton (NFA) from a regular expression [16]. On the other hand, we often have an exponential-size regular expression from an FA: For example, given an FA A with n states over k letter alphabet, the size of a corresponding regular expression can be $O(nk4^n)$ in the worst-case [4,9].

Jiang and Ravikumar [10] proved that it is PSPACE-complete to compute a minimal regular expression and NP-complete to find a minimal regular expression for an acyclic FA. Note that the regular expression minimization problem in general is PSPACE-complete [13]. Ellul et al. [4] showed that if A is a planar graph, then we can obtain a regular expression whose size is less than $e^{O(\sqrt{n})}$. Recently, based on this work, Gruber and Holzer [6] demonstrated that we can compute a regular expression whose size is $O(1.742^n)$ for an n-state deterministic FA. However, the running time for these algorithms are exponential and, therefore, are not suitable for implementation. We examine some known heuristics that may lead to shorter regular expressions and design a new state elimination algorithm using heuristics.

S. Maneth (Ed.): CIAA 2009, LNCS 5642, pp. 178–187, 2009.

In Section 2, we define some basic notions. In Section 3, we briefly describe state elimination and demonstrate the importance of state removal sequence. Then, we revisit known heuristics and related issues in Section 4. Based on these heuristics, we design a new state elimination algorithm, implement the algorithm in Grail+[1] and show experimental results in Section 5.

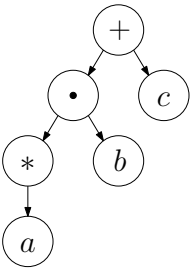

Fig. 1. The syntax tree representation for a regular expression $E = a^*b + c$. We define the size $|E|$ of E to be the number of nodes in the corresponding syntax tree. For instance, $|E| = 6$.

2 Preliminaries

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ. The size $|\Sigma|$ of Σ is the number of characters in Σ. A language over Σ is any subset of Σ^*. The symbol $\emptyset$ denotes the empty language and the symbol λ denotes the null string.

An FA A is specified by a tuple $(Q, \Sigma, \delta, s, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. If F consists of a single state f, then we use f instead of $\{f\}$ for simplicity. Let $|Q|$ be the number of states in Q and $|\delta|$ be the number of transitions in δ. Then, the size of A is $|A| = |Q| + |\delta|$. For a transition $\delta(p, a) = q$ in A, we say p has an *out-transition* and q has an *in-transition*. Furthermore, we say that A is *non-returning* if the start state of A does not have any in-transitions and A is *non-exiting* if all final states of A do not have any out-transitions. If $\delta(q, a)$ has a single element q', then we denote $\delta(q, a) = q'$ instead of $\delta(q, a) = \{q'\}$ for simplicity.

A string x over Σ is accepted by A if there is a labeled path from s to a final state such that this path spells out x. We call this path an *accepting path*. Then, the language $L(A)$ of A is the set of all strings spelled out by accepting paths in A. We say that a state of A is *useful* if it appears in an accepting path in A; otherwise, it is *useless*. Unless otherwise mentioned, in the following we assume that all states of an FA are useful.

[1] Grail+ is a symbolic computation environment for finite-state FAs, regular expressions and finite languages. Homepage: http://www.csd.uwo.ca/Research/grail/

We define the size of a regular expression E to be the number of characters of Σ and the number of operations[2]. Note that we often omit the catenation symbol in a regular expression. For instance, we write ab instead of $a \cdot b$ and the sizes of both regular expressions are 3. For the precise definition, we can think of the size of E as the number of nodes in the corresponding syntax tree. Fig. 1 gives an example.

For complete background knowledge in automata theory, the reader may refer to textbooks [9,17].

3 State Elimination

We define the *state elimination* of $q \in Q \setminus \{s, f\}$ in A to be the bypassing of state q, q's in-transitions, q's out-transitions and q's self-looping transition with equivalent expression transition sequences. For each in-transition (p_i, α_i, q), $1 \leq i \leq m$, for some $m \geq 1$, for each out-transition (q, γ, r_j), $1 \leq j \leq n$, for some $n \geq 1$, and for the self-looping transition (q, β, q) in δ, construct a new transition $(p_i, \alpha_i \cdot \beta^* \cdot \gamma_j, r_j)$. If there exists transition (p, ν, r) in δ for some expression ν, then we merge two transitions to give the bypass transition $(p, (\alpha_i \cdot \beta^* \cdot \gamma_j) + \nu, r)$. We then remove q and all transitions into and out of q in δ. For more details on state elimination, refer to the literature [2,17].

One interesting property in state elimination is that the resulting regular expression from state elimination depends on the removal sequence. Therefore, depending on which removal sequence we choose, we may have a shorter regular expression for the same FA. Fig. 2 illustrates this idea.

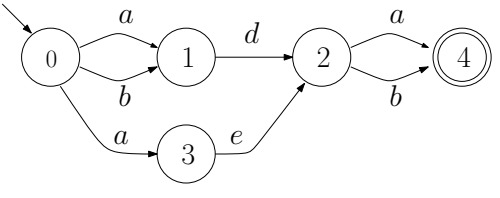

Fig. 2. An example of different regular expressions by different removal sequences for a given FA. $E_1 = ae(a + b) + (a + b)d(a + b)$ is the output of state elimination in $1 \to 2 \to 3$ order and $E_2 = ((a + b)d + ae)(a + b)$ is the output of state elimination in $1 \to 3 \to 2$ order, where $L(E_1) = L(E_2)$.

For an n-state FA A, there are $n!$ removal sequences. It is undesirable to try all possible sequences for shorter regular expressions. Instead, we use the structural properties of A and design a fast heuristic for state elimination that can give a shorter regular expression.

[2] Grail+ also defines the size of regular expression in this way.

4 Heuristics for State Elimination

There are several heuristics for finding a removal sequence for state elimination. For example, Gruber and Holzer [6] suggested graph separator techniques and Delgado and Morais [3] relied on state weight. Recently, Moreira and Reis [14] presented an $O(n^2 \log n)$ time algorithm that obtains an $O(n)$ size regular expressions from an n-state acyclic FA A. Gulan and Fernau [7] proposed a construction of regular expression from a restricted NFA via extended automata. Note that some heuristics for state elimination run in exponential time. Since we intend to compute a shorter regular expression from an FA quickly, we only consider polynomial running time heuristics for our implementation. We use the decomposition heuristic by Han and Wood [8] and the state weight approach by Delgado and Morais [3]. Both approaches run in polynomial time.

4.1 The Decomposition Heuristic

Han and Wood [8] suggested two decomposition approaches, one is a horizontal decomposition and the other is a vertical decomposition, based on the structural properties of a given FA.

First, we use the vertical decomposition since it always guarantees the shortest regular expression by state elimination. For the vertical decomposition, we first identify bridge states.

Definition 1. *We define a state q in an FA A to be a* bridge state *if it satisfies the following conditions:*

1. *State q is neither a start nor a final state.*
2. *For each string $w \in L(A)$, its path in A must pass through q at least once.*
3. *State q is not in any cycle except for the self-loop.*

Note that the bridge state condition is more restricted than the original condition proposed by Han and Wood [8][3]. This is because we find a counter example that does not guarantee an optimal solution under the original conditions. In Fig. 3, the removal sequence $1 \to 2$ gives $E_1 = cd(b + ad)^*a$ whereas the removal sequence $2 \to 1$ gives $E_2 = c(db^*a)^*(db^*a)$. Note that $|E_1| < |E_2|$.

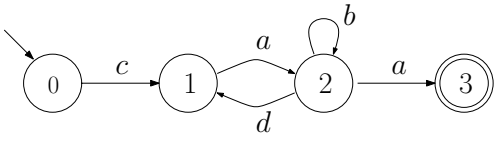

Fig. 3. State 1 satisfies the bridge conditions by Han and Wood [8]. However, it is not a bridge state according to Definition 1.

Han and Wood [8] presented an algorithm that finds bridge states in linear time in the size of a given FA based on the DFS algorithm. We can slightly modify the algorithm and find all bridge states in Definition 1 in linear time as well.

[3] The original condition allows q to be in a cycle.

Proposition 1. *Given an FA $A = (Q, \Sigma, \delta, s, f)$ and a set B of bridge states of A, the optimal removal sequence must eliminate all states in $Q \setminus B \cup \{s, f\}$ before eliminating any bridge states.*

Given an FA A, we find bridge states in liner time and apply the vertical decomposition. Once we obtain several decomposed subautomata for A, we try the horizontal decomposition before computing a regular expression for each subautomaton.

Proposition 2 (Han and Wood [8]). *Given a finite-state automaton $A = (Q, \Sigma, \delta, s, f)$, we can discover all subautomata that are disjoint from each other except s and f in $O(|Q| + |\delta|)$ time using DFS.*

Fig. 4 gives an example of a horizontal decomposition. We notice that the horizontal decomposition is a good heuristic for state elimination since the removal sequence for each separated subautomaton does not influence any other removal sequence for other subautomata. For example, in Fig. 4, the removal sequence $2 \to 5 \to 3 \to 4 \to 6$ and the removal sequence $2 \to 3 \to 4 \to 6 \to 5$ give the same regular expressions. Namely, we only look at each subautomaton to find a proper removal sequence and merge the resulting regular expressions using unions. Therefore, if possible, we always decompose a given FA into several horizontally disjoint subautomata, and compute the corresponding regular expressions for subautomata and merge them.

Proposition 3. *We can use the horizontal decomposition for finding a short regular expression using state elimination. Note that the horizontal decomposition does not affect the optimal removal sequence.*

Moreover, as shown in Fig. 4, states 3, 4 and 5 become bridge states in A_l that are not bridge states in A. In other words, we can repeat the vertical decomposition, if possible, and the horizontal decomposition again. Since there

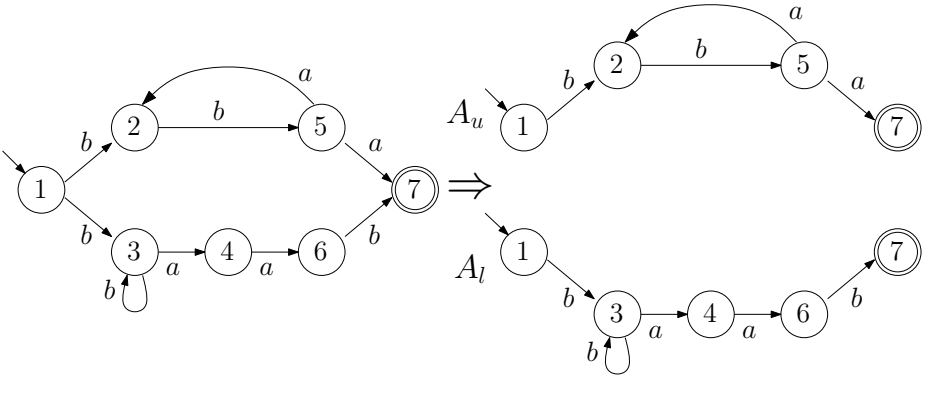

Fig. 4. An example of a horizontal decomposition for a given FA without bridge states

are only finite number of states, this process runs in polynomial time. Overall, the decomposition heuristic is a classical divide-and-conquer approach for state elimination.

Proposition 4. *Given an FA A, we can decompose A, if possible, into several subautomata in which both horizontal and vertical decomposition are not feasible in $O(|A|^2)$ worst-case time.*

4.2 The State Weight Heuristic

Delgado and Morais [3] proposed the sate weight heuristic. They defined a state weight be the the size of new transition labels that are created by eliminating the state. We borrow their notion and define the weight of a state q in an FA $A = (Q, \Sigma, \delta, s, f)$ as follows:

$$\sum_{i=1}^{In}(W_{in}(i) \times Out) + \sum_{i=1}^{Out}(W_{out}(i) \times In) + W_{loop} \times (In \times Out), \tag{1}$$

where In is the number of in-transitions excluding self-loop, Out is the number of out-transitions excluding self-loop, $W_{in}(i)$ is the size of the transition label on the ith in-transition, $W_{out}(i)$ is the size of the transition label on the ith out-transition and W_{loop} is the self-loop label size for q. Note that our weight definition is slightly different from Delgado and Morais [3]: We define the weight be to the total size of transition labels after eliminating q. We can compute the weight of all states in A in polynomial time.

Delgado and Morais [3] noticed that the state weight heuristic does not guarantee the shortest regular expression. For instance, in Fig. 5, the state weight heuristic suggests $1 \to 3 \to 2$ removal sequence, which gives $E_1 = abc(((a+b+c)+(b+bb)c))^*(b+bb)a$ whereas the removal sequence $1 \to 2 \to 3$ gives $E_2 = ab(c(a+b+c)^*(b+bb))^*a$. Note that we can select a least weight state and remove it, and recompute the state weight and choose a new least weight state in the resulting FA. This approach does not guarantee the shortest regular expression either but it often gives shorter regular expressions compared

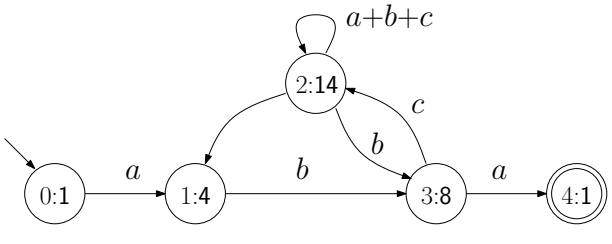

Fig. 5. Each state has a state index and the state weight by Equation (1). In this FA, the state weight heuristic suggests $1 \to 3 \to 2$ removal sequence but it is not the best removal sequence.

with the one-time state weight heuristic. On the other hand, since we calculate state weight every step, it may take more time than the one-time state weight heuristic. We implement both approaches and analyze the experimental results in Section 5.

5 Implementation and Experimental Results

Given an FA $A = (Q, \Sigma, \delta, s, F)$, we first remove all unreachable states, merge multiple transitions between two states into a single transition and make A to have a single final state using λ-transitions. This preprocessing takes $O(|A|)$ time. We use combinations of heuristics in Section 4 for state removal sequences as follows.

1. We eliminate states in state order without any heuristics. Let C-I denote this case.
2. We use both the vertical decomposition and the horizontal decomposition until both decompositions are not feasible. Once the decomposition step is over, we eliminate states in order. Let C-II denote this case.
3. We compute the state weight of all states and eliminate a state with less weight. Note that we compute the state weight only once. Let C-III denote this case.
4. We first use the vertical and horizontal decompositions and decide the removal sequence for each decomposed subautomaton using the state weight heuristic as C-III. Let C-IV denote this case.
5. We select a least weight state and eliminate it. Then, we compute the state weight again for the resulting FA and eliminate the new least weight state. We repeat this until there is no more state to remove. Namely, we have to compute the state weight roughly $|Q|$ times, where $|Q|$ is the number of states in an input FA. Let C-V denote this case. Note that C-V is different from C-III.
6. We use the vertical and horizontal decompositions. Then, for each decomposed subautomaton, we use the repeated state weight heuristics to decide the removal order as C-V. Let C-VI denote this case.

Our implementation is based on Grail+. C-I is the current state elimination algorithm implemented in Grail+ as well as JFLAP [15]. We implement the other 5 heuristics in Grail+. We randomly generate FAs and run the 6 different algorithms for each of FAs on a Pentium-5 PC. Table 1 shows some of the experimental results. (We omit most cases because of the space limit.) Notice that if the number of transitions is large, then the straightforward state elimination approach (C-I) cannot compute a regular expression because of the exponential blow-up. (Our empirical experience is that if the number of transitions is more than 200, then C-I often fails.)

Fig. 6 shows the relation graph between the number of states and the size of regular expressions. This shows that C-VI is best followed by C-IV → C-V → C-II → C-III → C-I.

Table 1. Experimental results

		C-I	C-II	C-III	C-IV	C-V	C-VI
$\|\delta\|$	$\|Q\|$	the size of resulting regular expression for each case					
		the real running time in second					
43	10	1773	779	644	399	504	399
		0.079	0.031	0.015	0.016	0.016	0.015
48	10	2605	687	1119	468	427	427
		0.125	0.032	0.046	0.016	0.016	0.015
59	12	1654	1148	1186	899	722	599
		0.078	0.047	0.062	0.031	0.032	0.015
65	14	1072	300	969	300	311	300
		0.047	0.016	0.031	0.016	0.015	0.016
67	15	16048	9609	1572	711	699	699
		0.906	0.5	0.078	0.016	0.031	0.031
99	17	15268	717	964	469	475	469
		0.937	0.047	0.047	0.015	0.032	0.015
157	24	826669	5664	15867	3104	9425	2892
		70.141	0.25	1.141	0.14	0.609	0.109
175	23	2007485	28855	25472	21566	16430	13940
		169.984	1.609	2.125	1.203	1.078	0.625
265	35	-	2022	47007	1614	7614	1592
		-	0.078	4.781	0.062	0.532	0.047
296	40	-	1797	46803	1508	1853	1448
		-	0.063	4.875	0.063	0.219	0.062
673	56	-	-	-	38117	36090	27920
		-	-	-	2.078	4.563	1.25

Proposition 5. *We propose the following suggestions for the state elimination implementation based on our experimental results:*

1. *It is better to apply the decomposition heuristic first and the state weight heuristic later for each decomposed subautomaton.*
2. *Heuristics for state elimination enable to obtain a regular expression faster compared with state elimination without heuristics although they require additional processing time. This is because heuristics help to have smaller transition labels while running state elimination.*
3. *The number of transitions is more closely related to the size of regular expressions than the number of states. (The correlation between the size of regular expressions by C-VI and the number of states is 0.66 whereas the correlation between the size of regular expressions by C-VI and the number of transitions is 0.79.)*

Proposition 5 suggests to investigate the relation between the number of transitions and the number of states. Thus, it is natural future work to examine a tight bound for the size of a regular expression from an n-transition FA. Moreover, we can use some other heuristics that run in polynomial time. For example, we

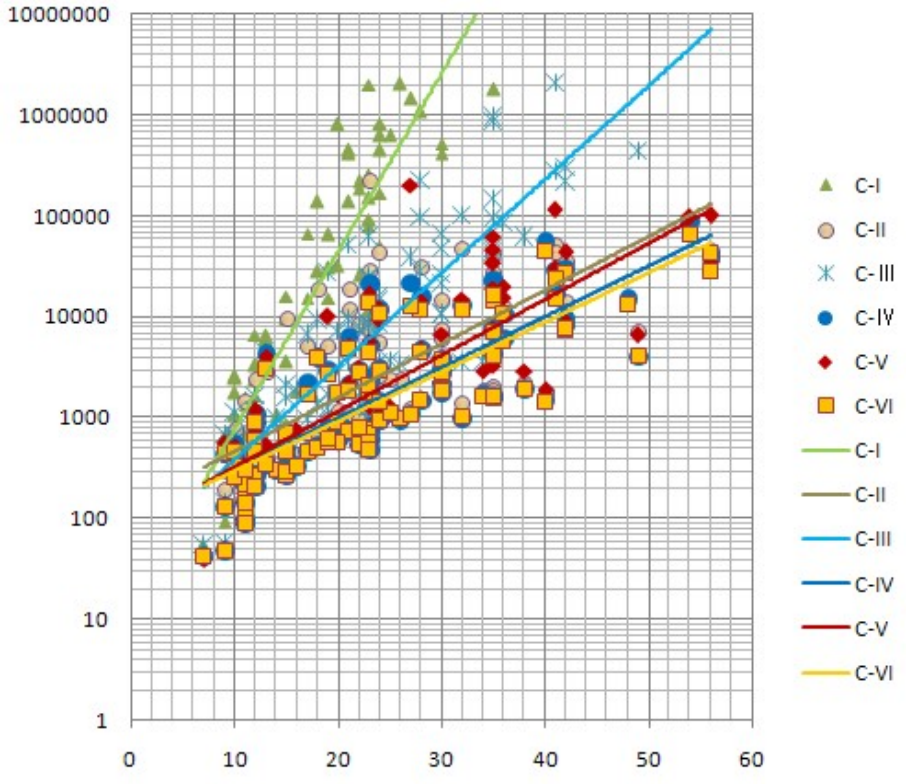

Fig. 6. Experimental results for 6 cases: number of states and size of regular expressions

can use the orbit property established by Brüggemann-Klein and Wood [1] that gives a certain removal order and the Kleene star operation.

Acknowledgment

We wish to thank the referees for the care they put into reading the previous version of this manuscript.

References

1. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Information and Computation 140, 229–253 (1998)
2. Brzozowski, J., McCluskey Jr., E.: Signal flow graph techniques for sequential circuit state diagrams. IEEE Transactions on Electronic Computers EC-12, 67–76 (1963)
3. Delgado, M., Morais, J.: Approximation to the smallest regular expression for a given regular language. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317, pp. 312–314. Springer, Heidelberg (2005)

4. Ellul, K., Krawetz, B., Shallit, J., Wang, M.-W.: Regular expressions: New results and open problems. Journal of Automata, Languages and Combinatorics 9, 233–256 (2004)
5. Glushkov, V.: The abstract theory of automata. Russian Mathematical Surveys 16, 1–53 (1961)
6. Gruber, H., Holzer, M.: Provably shorter regular expressions from deterministic finite automata. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 383–395. Springer, Heidelberg (2008)
7. Gulan, S., Fernau, H.: Local elimination-strategies in automata for shorter regular expressions. In: Proceedings of SOFSEM 2008, pp. 46–57 (2008)
8. Han, Y.-S., Wood, D.: Obtaining shorter regular expressions from finite-state automata. Theoretical Computer Science 370(1-3), 110–120 (2007)
9. Hopcroft, J., Ullman, J.: Introduction to Automata Theory, Languages, and Computation, 2nd edn. Addison-Wesley, Reading (1979)
10. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM Journal on Computing 22(6), 1117–1141 (1993)
11. Kleene, S.: Representation of events in nerve nets and finite automata. In: Shannon, C., McCarthy, J. (eds.) Automata Studies, pp. 3–42. Princeton University Press, Princeton (1956)
12. McNaughton, R., Yamada, H.: Regular expressions and state graphs for automata. IEEE Transactions on Electronic Computers 9, 39–47 (1960)
13. Meyer, A., Stockmeyer, L.: The equivalence problem for regular expressions with squaring requires exponential time. In: Proceedings of the Thirteenth Annual IEEE Symposium on Switching and Automata Theory, pp. 125–129 (1972)
14. Moreira, N., Reis, R.: Series-parallel automata and short regular expressions. Fundamenta Informaticae (accepted for publication, 2009)
15. Rodger, S.H., Finley, T.W.: JFLAP: An Interactive Formal Languages and Automata Package. Jones & Bartlett Pub. (2006)
16. Thompson, K.: Regular expression search algorithm. Communications of the ACM 11, 419–422 (1968)
17. Wood, D.: Theory of Computation. John Wiley & Sons, Inc., New York (1987)