Hashing Techniques For Finite Automata

Hady Zeineddine
Logic Synthesis Course Project - Spring 2007
Professor Adnan Aziz

1. Abstract

This report presents two hashing techniques - Alphabet and Power-Set hashing- that can be applied simultaneously on deterministic finite automata, to minimize their space complexity. It is shown that alphabet hashing minimizes the number of possible transitions per state and reduces, to some extent, the number of states. Power-set hashing is invoked upon building an automaton from regular expressions and is concerned with reducing the total number of states. The correctness, minimization capability and error performance of both techniques are studied throughout the report.

2. Introduction

Automata theory -rooted basically in theoretical computer science- is a study of an abstract machine, the automaton, and the specific computational or recognition function it can have. The theory of finite automata constitutes the theoretical basis for string matching algorithms that are widely deployed in text editors, compilers, and networks security software.

2.1. Definition and Terminology

Formally defined, a deterministic automaton (DFA), M is a 5-tuple$(Q, q_0, A, \Sigma, \delta)$ where [1]:

1. Q is a finite set of states.
2. $q_0 \in Q$ is the start state.
3. $A \subseteq Q$ is a distinguished set of accepting state.
4. Σ is a finite input alphabet.
5. δ is a function from $Q \times \Sigma$ into Q, called the transition function of M.

The transition function δ is extended to apply to a state and a string rather than a state and symbol. Let Σ^* be the set of all finite-length strings formed using characters from the alphabet Σ, define a new function ϕ from $Q \times \Sigma^*$ to Q such that:

1. $\phi(q, \varepsilon) = q$ where ε denotes the empty string.
2. for all strings w and input symbols $a \in \Sigma$: $\phi(q, wa) = \delta(\phi(q, w), a)$.

The finite automaton begins in state q_0 and reads the elements of the input string, S, one at a time. If the automaton is in state q and reads input symbol $a \in \Sigma$, it makes a transition from q to $\delta(q, a)$. For a sub-string S_i, if $\phi(q_0, S_i) \in A$, the sub-string S_i is said to be accepted by M. A language L is any subset of Σ^*, and the language accepted by M is designated $L(M) = \{w | \phi(q_0, w) \in A\}$. A language is a regular set if it is the set accepted by some finite automaton.

A nondeterministic finite automaton (NFA) is a 5-tuple$(Q, q_0, A, \Sigma, \delta)$, where Q, q_0, A, and Σ have the same meaning as for a DFA, but δ is a mapping from $Q \times \Sigma$ into 2^Q, the power set of Q. The function $\phi : Q \times \Sigma^* \rightarrow 2^Q$ is then extended as follows:

1. $\phi(q, \varepsilon) = \{q\}$.
2. $\phi(q, wa) = \{p|$ for some state $r \in \phi(q, w), p \in \delta(r, a)\}$.

A fundamental theorem on the relation between DFA and NFA is [2]:

Theorem 1 *Let L be the set accepted by a non-deterministic finite automaton, then there exists a deterministic finite automaton that accepts L.*

2.2. Hashing for Space Optimization

The space required by a deterministic finite automaton can be modeled by a $|Q| \times |\Sigma|$ transition table. While $|\Sigma|$ is predefined, the number of states, Q can grow up exponentially upon transforming a NFA to an equivalent DFA, making the space for storing the DFA transition table prohibitively large. The NFA occupies considerably less space, however, this comes at the expense of a considerable increase in the processing time of the string matching and language recognition algorithms. To break this deadlock, it is proposed in this paper to utilize hashing techniques to optimize the size of the transition table while preserving the performance advantage of DFA over NFA in terms of processing time. Formally stated, an efficient hashing mechanism transforms an automaton M to another automaton M', such that:

1. $|Q_{M'}| \times |\Sigma_{M'}| \ll |Q_M| \times |\Sigma_M|$.
2. $L(M) \subseteq L(M')$.

The second condition is a fundamental condition for the correctness of hashing. The automata transformation involved in hashing comes at the expense of an extra processing stage, the recovering stage defined as: when a sub-string S_i is accepted by M', a testing procedure is applied to see if $S_i \in L(M)$. If $S_i \notin L(M)$, this is called a false positive hit, and constitutes the hashing overhead. Thus, developing an appropriate hashing mechanism involves a space-time-accuracy tradeoff. The problems and techniques of the recovering stage is beyond the scope of this report.

Two hashing mechanisms are studied in this report: *alphabet hashing* and *power-set hashing*. Alphabet hashing optimizes the $|\Sigma|$ factor and may also decrease the number of states, while power-set hashing is targeted at minimizing the number of states during the transformation of an NFA to its equivalent DFA. The rest of the report is organized as follows: Section 3 studies alphabet hashing on a specific class of automata, the pattern matching automata, and shows some simulation results on the effectiveness of this method. Section 4 introduces the concept of regular expression and proves that alphabet hashing concept can be extended to any finite automata.Next, the power-set hashing method is proposed and its efficiency is illustrated through simulation results.

3. Alphabet Hashing for Pattern Matching

The pattern-matching problem can be formalized as follows: Given an alphabet Σ, set of patterns $P \subset \Sigma^*$, and string $S \in \Sigma^*$, find all integer pairs, (i,m), such that $S_{i+1-m \cdots i} \in P$.
The pattern matching algorithm typically works by building a finite automaton M as a pre-processing step and feeding S as the input string to M. If a substring S_i has $\phi(q_0, S_i) \in A$, this means a pattern of P matches a suffix of $|S_i|$. A method to construct the finite automaton M is described next. A string S_i is called a prefix of S_j if $S_j = S_iW$ for $W \in \Sigma^*$ (and a suffix if $S_j = WS_i$). Define the set $Pe = \{p \in \Sigma^* | p$ is a prefix of a pattern in P$\}$. Establish a bijection b between Pe and the set of states, Q, and let $\delta(b(p_i), a) = \{b(p_j) | p_j$ is the longest suffix of p_ia and $p_j \in Pe\}$.

3.1. Definition

The concept of alphabet hashing is based on mapping the alphabet Σ to a smaller alphabet Σ'. The pattern set, P, and the input strings are mapped using the same mapping function and an automaton M' is built for the new pattern set P'. A direct result of this mapping is a decrease in the horizontal dimension of the transition table by $|\Sigma|/|\Sigma'|$ times. To show the validity of this approach, the following lemma is proven.

Lemma 2 *Let M' be the automaton obtained by applying the mapping function h on the alphabet Σ, then $h(L(M)) \subseteq L(M')$*

Proof. : Let a string $x \in L(M)$, then there exist a suffix y of x such that $y \in P$. Since x and y are mapped by the same function h, $h(y) \in h(P) = P'$ and $h(y)$ is a suffix of $h(x)$, therefore $h(x) \in L(M')$. ∎

3.2. Analysis

In addition to the obvious reduction in the horizontal dimension of the transition table, alphabet hashing reduces the number of states (i.e. vertical dimension), while -on the other hand- it introduces false negative hits where the automaton M' accepts a string that in not in $L(M)$. For a first order analysis, we assume that the symbols of the patterns and of the input string occur independently according to a pre-defined probability distribution. Let $|\Sigma| = a$, $|\Sigma'| = a'$, $\alpha_1 \cdots \alpha_a$ be the symbols of Σ, and p_i and q_i denote the probability of occurrence of α_i in the patterns and input string respectively, then by the above assumption, $p(S = \alpha_i \cdots \alpha_k) = q_i \cdots q_k$.
Let P_i and P'_i be the set of distinct prefixes of length i in the original and image pattern sets respectively($|P_i| = X_i$ and $|P'_i| = X'_i$), then the total reduction in the number of states is $\sum(X_i - X'_i)$. Consider the prefix of length m, to every possible prefix v associate the random variable $R_v = 1$ if $v \in P'_i$ and 0 otherwise. Take $X = X_m$ and $X' = X'_m$, then under the reasonable assumption that $X << a^m$:

$$E(X'|X) = \sum_{v \in \Sigma'^m} E(R_v|X)$$

$$\approx \sum_v (1 - (1 - p_v)^X) = a'^m - \sum_v (1 - p)^X.$$

where p_v is the probability of occurrence of v as determined by the probability distribution p'. Applying the inequalities:

$$1 - xp \leq (1-p)^x \leq 1 - xp + 0.5x^2p^2$$

we get:

$$X - (1/2)X^2 \sum p_v{}^2 \leq E(X'|X) \leq X$$

Table1 below shows how the lower bound stated above is varied along a' and X, for a uniform distribution. As can be noticed from the data above, for $a' > 4$ no real reduction happens in the ratio of X'/X:

Table 1.

a’	2	4	8	16	32
X=10^3	0.8779	1	1	1	1
X=10^4	-0.2207	0.9997	1	1	1
X=10^5	-11.2070	0.997	1	1	1
X=10^6	-121.0703	0.97	1	1	1

Considering the hashing function: $h(\alpha_i) = \alpha_{imoda'}$, it is reasonable for $a' << a$ to assume that the probability distribution of the elements of Σ' (i.e. p_i') is uniform regardless of the probability distribution in the original alphabet, therefore: $E(X'|X) = a'^m - a'^m(1-a'^{-m})^X$.

Using the fact that for $\sum_{i=1\cdots n} r_i = c$, $min\{\sum r_i{}^x| = c(c/n)^x$, it is can be shown -by setting r to $1 - p_v$ where $c = a^m - 1$- that the uniform probability distribution of the alphabet characters yields the maximum ratio of X'/X. To illustrate this point, 2 hashing methods are applied on a set of 1000 strings of length 6 through 12: the first is the conventional $moda'$ hashing function, while the second sorts the alphabet characters in terms of decreasing probability and hashes a fraction $r(0.3)$ of them to the a smaller alphabet subset $\Sigma" \subset \Sigma'$, i.e. $a" < a'$, while the rest of the characters are hashed using the conventional $moda'$ method. As can be seen from fig1, the second hashing technique yields further reductions in the number of prefixes, however, a quantitative description of this reduction is dependant on the probability distribution of Σ and the hashing metrics (i.e. $a", r, \cdots$). It must be noticed, that $X' < min(X, a'^m)$, so if $X >> a'^m$, a low value of X'/X means that the set of prefixes is saturated, and thus if the length of patterns is m, approximately every string will be accepted and thus the hashing technique breaks down.

To estimate the number of false negative hits caused by hashing, we consider a pattern set of X' patterns of length m, then the expected number of hits (hit occurrence of matched pattern) for a string S is $(|S| - m + 1)E(W)$, where given a string s of length m sampled according to the distribution q'(defined previously), W is the random variable that is 1 if s matches any of the X patterns in the pattern set and 0 otherwise, therefore:

$$E(W|X,X') = X'. \sum_{v \in \Sigma'^m} (q_v p_v)$$

$$\Rightarrow E(W|X) = \sum (X.p(X). \sum_{v \in \Sigma'^m} (q_v p_v))$$

$$= (\sum X.p(X))(\sum (q_v p_v)) = E(X). \sum (q_v p_v)$$

While $E(V)$ gives the number of false negative and true positive cases, it is reasonable to assume that the probability of the latter case is too small, and thus that $E(V)$ is approximately equal to the average number of false negative hits. As can be noticed from the above equations, hashing methods play a central role in the rate of error expected: informally stated, if statistical hashing can obtain q_v and p_v that have incompatible probability distributions i.e. low frequency characters in the patterns have relatively high frequency in the string and/or vice-versa, it can reduce the error significantly, however if the two patterns are compatible, uniform hashing will be obviously perform better in terms of error probability.

3.3. Experiments and Conclusion

To investigate the performance of alphabet hashing, 2 sets of patterns were tested across a string of 10^8 randomly generated characters: the first one contains patterns with a minimum length of 14, while the second contains patterns with a minimum length of 8 (set1 $\in$ set2). Uniform hashing was applied over the 2 sets. Figure 2 shows the reduction in the number of states for the automata of the 2 sets across the degree of hashing. As can be seen, this reduction is negligible for large a' but the curve become steeper as a' decreases. Figure 3 shows the number of errors associated with the degree of hashing, which illustrates clearly the steep nature of the error performance curve, as well as the inefficiency of hashing to very small alphabet subsets. As expected, pattern sets with small lengths are more prone to error than the larger-length pattern sets. Next, statistical hashing is applied on the the pattern set 1, with an extreme choice of $\Sigma" = \{0\}$ and r 0.25. The table below shows the number of states in the case of statistical versus uniform hashing. while extreme statistical hashing (i.e. having high r and low $a"$) decreases significantly the number of states, the error performance depends upon the compatibility of the probability distributions of the characters in the string as compared to that of the patterns. To illustrate this point, 2 hypothetical cases were considered in the Figure4. The

Table 2.

a'	statistical hashing	uniform hashing
16	702	6055
8	682	5964
4	588	5613
2	295	4556

first case is when the probability distribution of characters in the string is equivalent to that in the pattern set(compatibility), across the case when the characters are uniformly chosen from the alphabet Σ(partial non compatibility). As is shown, the compatibility case has a significantly large error rate as compared to the error rate of the incompatible case, which are both inferior to the uniform hashing. The common error floor is the result of the fact that high frequent characters in the patterns are also frequent in the strings(with difference in frequency). While statistical hashing conditions may be unrealistic, its idea can be extended to multi-character hashing(2 to 4), where the probability distribution then may be significantly different (this, however, adds to the string hashing complexity).

4. Hashing for Regular Expressions

The languages accepted by finite automata can be described by simple expressions called regular expressions. These expressions are based on three operations that are applied on 1 or 2 languages to result in a new language. Let R and S be two languages accepted by finite automata with regular expressions r and s, the operations that can be applied on them are:

1. Union $(r+s)$: $L = \{x \in \Sigma^* | x \in R$ or $x \in S\}$
2. Concatenation(rs): $L = \{xy \in \Sigma^* | x \in R$ and $y \in S\}$
3. Closure$(r*)$: $L = \{x \in \Sigma^* | x \in R^i$ for $i \geq 0\}$

A basic theorem on the relation between finite automata and regular expressions is:

Theorem 3 *Let L be the set of languages accepted by finite automata and let L' be the set of languages that are denoted by regular expressions, then $L \equiv L'$. [2]*

In this section the idea of alphabet hashing is extended to regular expressions and some results on its performance are shown. Then, power-set hashing is introduced and some primary results concerning its effectiveness are presented.

4.1. Alphabet Hashing in Regular Expressions

Applying Alphabet hashing over regular expressions satisfy the fundamental condition that $L(M) \subseteq L(M')$, as is proven in the following lemma:

Lemma 4 *Let r be a the regular expression accepted by some automaton M, then applying alphabet hashing on r, results in an expression r' that is expressed by the automaton M' such that $h(L(M)) \subseteq L(M')$*

Proof. : The proof is based on induction on the number of operators in a regular expression.

Consider a regular expression with 0 operators, then by definition $h(L(M)) \equiv L(M') \Rightarrow h(L(M)) \subseteq L(M')$. Suppose this is true for languages R and S having a number of operators less than i. Consider the three expressions

1. r+s: let $t \in h(L(M_{r+s}))$ then there exist a string t_0, such that $t_0 \in L(M_{r+s})$ and $h(t_0) = t$. Therefore, $t_0 \in L(M_r)$ or $t_0 \in L(M_s)$, then $h(t_0) \in L(M'_{r'})$ or $h(t_0) \in L(M'_{s'})$, so $h(t_0) \in L(M'_{r'}) \cup L(M'_{s'})$, so $t = h(t_0) \in L_(M'_{r'+s'})$.
2. rs : let $t \in h(L(M_{rs}))$ then there exist some string t_0, such that $t_0 \in L(M_{rs})$ and $h(t_0) = t$ therefore $t_0 = t_{01}t_{02}$ such that $t_{01} \in L(M_r)$ and $t_{02} \in L(M_s)$. Thus, $h(t_{01}) \in L(M'_{r'})$ and $h(t_{02}) \in L(M'_{s'})$, so $t = h(t_{01})h(t_{02}) \in L(M'_{r's'})$.
3. r^* : Let $t \in h(L(M_{r^*}))$ then there exist some string t_0, such that $t_0 \in L(M_{r^*})$ and $h(t_0) = t$. Therefore, there exist some integer $n \geq 0$ such that $t_0 = t_{01} \cdots t_{0n}$ and $t_{0i} \in L(M_r)$, so $h(t_0) = h(t_{01}) \cdots h(t_{0n})$ and $h(t_{0i}) \in L(M'_{r'})$, then $t = h(t_0) \in L(M'_{r'*})$. ■

To test the change in the number of states and the error rate incurred by alphabet hashing, regular expressions are generated randomly and a string having character uniform probability distribution is input to the created automata. The samples show similar behavior, a sample of which is shown in the table below.

Table 3.

a'	number of states	number of false positives
32	250	0
16	250	0
8	262	0
4	286	1
2	314	812

As is noticed, hashing has increased the number of states slightly, the reason of which is not analyzed yet.

However, supported by the analysis and results of the previous part, it can be predicted that alphabet hashing does not produce a major error-efficient reduction, if ever, in the number of states. To deal with this limitation in alphabet hashing, another hashing method, power-set hashing, is presented next.

4.2. Power-Set Hashing

The power-set hashing method tries to minimize the number of states in a deterministic finite automata through linking it to its equivalent non-deterministic automaton, which is typically used in building the automaton from a regular expression. Consider a NFA M_n, and its equivalent DFA M, then a 1-to-1 mapping f can be made from set of states of M, Q, to the power set of the set of states of M_n, Q_n. Power-set hashing is defined as: A state $q \in Q$ can be hashed to another state q' if $f(q) \subseteq f(q')$.
The following lemma shows that power-set hashing preserves the fundamental property of hashing.

Lemma 5 *If an automaton M' is obtained from automaton M by power-set hashing a state q to another state q', then $L(M) \subset L(M')$.*
Proof. : To prove the above lemma we prove another lemma:

Lemma 6 *for $q_1, q_2 \in Q$, if $f(q_1) \subseteq f(q_2)$, then $f(\phi(q_1, w)) \subset f(\phi(q_2, w))$, for any string w.*

Proof. :The proof is done by induction on the number of characters of w. If $w = 1$, let $qn_c \in f(\phi(q_1, w))$; therefore, there exist a state $qn \in f(q_1)$ such that $qn_c \in \delta(qn, w)$. We have $qn \in f(q_1) \Rightarrow qn \in f(q_2)$, and thus $qn_c \in f(\phi(q_2, w))$ -(the n index means an NFA state)- then $f(\phi(q_1, w)) \subset f(\phi(q_2, w))$. Next, we apply the induction step on the string length L+1 assuming that it holds for string lengths $\leq L$. Let $v = w_1 \cdots w_L$, then $f(\phi(q_1, v)) \subset f(\phi(q_2, v))$. Now consider $q_{11} = \phi(q_1, v)$ and $q_{22} = \phi(q_1, v)$, we have $f(q_{11}) \subseteq f(q_{22})$, therefore, $f(\phi(q_{11}, w_{L+1})) \subset f(\phi(q_{22}, w_{L+1}))$. ■

Now let $x \in L(M)$, and consider two cases:

1. x has no prefix $x_{1 \cdots i}$ such that $\phi(q_0, x_{1 \cdots i}) = q_1$, then no change will happen to $\phi(q_0, x_{1 \cdots i})$ for $i = 1 \cdots |x|$, and thus $\phi(q_0, x) = \phi'(q_0, x)$.
2. x has at least one prefix $x_{1 \cdots i}$ such that $\phi(q_0, x_{1 \cdots i}) = q_1$, let i be the minimum of these indices, then $\phi(q_0, x_{1 \cdots i}) = q_1$, and since $f(q_1) \subseteq f(q_2)$ by definition of power-set hashing, then $f(\phi(q_1, x_{i+1, \ldots, |x|})) \subset f(\phi(q_2, x_{i+1, \ldots, |x|}))$. Therefore, if $f(\phi(q_0, x))$ contains an accepting NFA state sn, then $sn \in f(\phi'(q_0, x))$, and thus $x \in L(M')$. ■

The performance of power-set hashing depends on the details of the hashing mechanism. For example, power-set hashing may provide a bridge between the states that are frequently passed by and the states that are infrequently passed by but that have a high probability of reaching an accepting state. Since no hashing method is developed, no informative results about the error performance of power-set hashing are presented. To estimate the reduction in the number of states, power-set hashing was applied on a DFA obtained from an NFA of a regular expression, under the following rule: state q_1 is hashed to a non-accepting state q_2 if 1) $f(q_1) \subseteq f(q_2)$, 2) $|fq_2| \leq |fq_2| + 1$, and 3) if q_1 is already not hashed into(i.e. no state q_3 was hashed into $q1$. The results are shown in the table below:

Table 4.

m	Q	Q'	Q'/Q
16	1618	658	0.4
16	3233	1040	0.32
16^1	3233	139	0.03
16^2	470	230	0.49
2	643	271	0.42
256	409	211	0.5

(^{1})*Condition 3 is relaxed.*
(^{2})*Condition 2 is relaxed into $|fq_2| \leq |fq_2| + 4$.*

As can be seen, even under the conservative and un-optimized hashing described above, the number of states is reduced into 1/2 to 1/3 of their original value (and even reaches 0.03 when the third condition was relaxed) which indicates that power-set hashing is highly efficient for state minimization.

5. Conclusion and Future Work

Two hashing techniques aimed at optimizing the space occupied by a finite automaton were investigated. Primary results show that these two mechanisms minimize the transition table associated with the automaton across both vertical and horizontal dimensions. More results and analysis for both types of hashing in the case of regular expressions is still needed, to get more insight into the error-performance as well as the issues of state minimization.

References

[1] T. H. Cormen, C.E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[2] J. E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.