Measures on Languages

1 Introduction

1.1 Notation

Let Σ denote a non-empty, countable alphabet. Unless otherwise specified, assume $|\Sigma| < \infty$. Write ϵ to mean the empty string.

Let $\Sigma^\star$ be the set of all finite strings from Σ. Write strings as w or s, whichever happens to be more convenient.

Let L denote a language, implicitly over Σ. In other words, $L \subseteq \Sigma^\star$.

We treat the empty language $\emptyset$ as distinct from the language with a single empty string $\{\epsilon\}$.

Let $\mathcal{L}$ be a family of languages.

Write deterministic finite automatons shorthand as DFA, and non-deterministic finite automatons shorthand as NFA.

Write regular expressions shorthand as regex.

If X is a set, then $\mathcal{P}(X)$ is the powerset of X.

Unless otherwise noted, vectors are implicitly in column format.

1.2 Regular Expressions

A regular expression over an alphabet Σ describes regular languages over Σ. Regular expressions are inductively generated, and we borrow heavily from Savage [3].

Definition 1 (Regular Expression)**.** *A regular expression over the finite alphabet Σ is defined inductively:*

(1) The empty language $\emptyset$ is a regular expression.

(2) *The empty string* ϵ *is a regular expression denoting* $\{\epsilon\}$.

(3) *For each* $a \in \Sigma$, *the standalone* a *is a regular expression denoting the singleton set* $\{a\}$.

(4) *If* r *and* s *are regular expressions, then so are* rs *(string concat),* $r + s$ *(string choice), and* $r^\star$ *(string repeat).*

Theorem 1 (Regular Expression Axioms)**.** *Regular expressions satisfy the following axioms:*

(1) $r\emptyset = \emptyset r = \emptyset$

(2) $r\epsilon = \epsilon r = r$

(3) $r + \emptyset = \emptyset + r = r$

(4) $r + r = r$

(5) $r + s = s + r$

(6) $r\,(s + t) = rs + rt$

(7) $(r + s)\,t = rt + st$

(8) $r(st) = (rs)t$

(9) $\emptyset^\star = \epsilon$

(10) $\epsilon^\star = \epsilon$

(11) $(\epsilon + r)^+ = r^\star$

(12) $(\epsilon + r)^\star = r^\star$

(13) $r^\star\,(\epsilon + r) = (\epsilon + r)\,r^\star = r^\star$

(14) $r^\star s + s = r^\star s$

(15) $r\,(sr)^\star = (rs)^\star\,r$

(16) $(r + s)^\star = (r^\star s)^\star\,r^\star = (s^\star r)^\star\,s^\star$

Outside of the Kleene star $\star$ operation, axioms (1 - 8) effectively state that regular expressions are an idempotent semiring with additive constant $\emptyset$ and multiplicative constant ϵ [2].

1.3 Representation of Regular Languages

There are several ways to represent regular languages, many of which can be found in literature [3]. We work with whatever is convenient for the problem at hand. For a regular language L over an alphabet Σ, there are a few notable ones:

(1) **Sets**: sometimes if the language is finite or has a simple structure, a complete set presentation may be convenient.

(2) **Regular expressions**: compact representation, also commonly used in practice when trying to do string matching.

(3) **Finite state machines**: Savage [3] gives a fairly standard representation.

Definition 2 (DFA)**.** *A DFA A is a five-tuple $A = (\Sigma, Q, \delta, q_0, F)$, where Σ is the alphabet, Q is the finite set of states, $\delta\colon Q \times \Sigma \to Q$ is the transition function, q_0 is the initial state, and F is the set of final states.*

For convenience, we might also write q_0 as q_1, especially when talking about matrix indices. We'll try to remember to make note of when this rewriting is done.

Definition 3 (NFA)**.** *A NFA A is identically defined except for the transition function, which is now $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$. Each transition non-deterministically picks one state from the set.*

(4) **Matrices**: transition matrices can be constructed from both DFAs and NFAs. First, take $Q = \{q_1, q_2, \ldots, q_n\}$. There are two primary possibilities:

(a) A matrix M_A corresponding to an automata A, with q_1 the initial state. Write $+\{\ldots\}$ do denote the summation regular expression over a set. We construct the matrix as follows:

$$M_{A,i,j} = +\{a \,:\, ((q_i, a), q_j) \in \delta\}$$

Here a is any character of Σ. In short, each entry of the matrix $M_{A,i,j}$ is the $+$ of all the characters that permit the transition from q_i to q_j.
If we take $v = ((1, 0, \ldots)$ as an n-dimensional vector, where each coordinate i represents state q_i. Suppose that u is an n-dimensional vector indicating the final states, where $u_i = 1_{q_i \in F}$, then:

$$v^T M_A^k u$$

Will corresponds to the regular expression of the sub-language of strings of precisely length k.

(b) Alternatively we may see regular expressions as a set of matrices, each corresponding to a letter of Σ. In essence, for each $a \in \Sigma$, the associated matrix M_a has form: $M_{a,i,j} = 1_{((q_i,\cdot),q_j)\in\delta}$, and indicates an adjacency transition matrix.
The matrix described earlier can be recovered by observing that:

$$M_A = \sum_{a\in\Sigma} a M_a$$

2 Measures on Languages

Given a family of languages $\mathcal{L}$, let $\sigma(\mathcal{L})$ be the σ-algebra generated on $\mathcal{L}$ satisfying the following:

(1)
$$\emptyset, \Sigma^\star \in \sigma(\mathcal{L})$$

(2)
$$L \in \sigma(\mathcal{L}) \implies L^c = \Sigma^\star \setminus L \in \sigma(\mathcal{L})$$

(3)
$$L_0, L_1, \ldots \in \sigma(\mathcal{L}) \implies \bigcup_{k=0}^{\infty} L_k \in \sigma(\mathcal{L})$$

Then $(\mathcal{L}, \sigma(\mathcal{L}))$ is a measurable space.

Remark 1. *If $\mathcal{L}$ happened to be a family of regular languages, there is no guarantee that $\sigma(\mathcal{L})$ will still be a family of regular languages. A counter example is the following:*

$$L_0 = \{\varepsilon\} \qquad L_1 = \{ab\} \qquad L_2 = \{aabb\} \qquad \ldots \qquad L_k = \left\{a^k b^k\right\} \qquad \ldots$$

But taking the countable union yields:

$$\bigcup_{k=0}^{\infty} L_k = \left\{a^k b^k \ : k \in \mathbb{Z}_{\geq 0}\right\}$$

Which is not regular.

2.1 Measure 1: From Non-negative Integers

We first consider the non-negative integers $\mathbb{Z}_{\geq 0}$. Let η be a σ-finite measure on $\mathbb{Z}_{\geq 0}$. The σ-finite conditions ensures that no strange singularities occur for any integers under consideration. We may later restrict η to be finite if necessary, if we want nicer conditions.

Observe that, by abuse of notation:

$$\Sigma^\star = \bigcup_{k=0}^{\infty} \Sigma^k$$

In English: $\Sigma^\star$ is the union of the set (language) of finite strings of length k, denoted Σ^k.

Because we assumed $|\Sigma| < \infty$, this also means that: $\left|\Sigma^k\right| = |\Sigma|^k$.

Consider now some language $L \in \sigma(\mathcal{L})$. Also decompose L into disjoint sub-languages by length as follows, with convenient subscripting:

$$L = \bigcup_{k=0}^{\infty} L_k$$

Of course, $L_k \subseteq \Sigma^k$.

Because we are able to precisely calculate $\left|\Sigma^k\right|$, one "natural" way of defining a measure λ_η on the measurable space $(\mathcal{L}, \sigma(\mathcal{L}))$ is as follows:

$$\lambda_\eta(L) = \sum_{k=0}^{\infty} \lambda_\eta(L_k) = \sum_{k=0}^{\infty} \frac{|L_k|}{|\Sigma^k|}\eta(k)$$

We claim that $(\mathcal{L}, \sigma(\mathcal{L}), \lambda_\eta)$ forms a measure space.

Theorem 2. λ_η *is a measure.*

Proof. We check (1) measure under empty set is zero and (2) countable additivity, which will satisfy the requirements of a measure.

(1) Observe that $\lambda_\eta(\emptyset) = 0$ because the sum will be trivial.

(2) Let $L_0, L_1, L_2, \ldots$ be a countable collection of pairwise disjoint languages. We decompose each of these languages into a countably indexed set, where $L_{j,k}$ is the jth language's sub-language that only contains strings of length k. In other words:

$$L_j = \bigcup_{k=0}^{\infty} L_{j,k}$$

Observe that by (de)-construction, for any fixed j and for all $k_1 \neq k_2$, we have L_{j,k_1} and L_{j,k_2} are pairwise disjoint.

However, we have a stronger condition because each L_j is assumed to be pairwise disjoint. Thus, for all $j_1 \neq j_2$ and $k_1 \neq k_2$, L_{j_1,k_1} and L_{j_2,k_2} are disjoint. Then:

$$\begin{aligned}
\lambda_\eta\left(\bigcup_{j=0}^{\infty} L_j\right) &= \lambda_\eta\left(\bigcup_{j=0}^{\infty}\bigcup_{k=0}^{\infty} L_{j,k}\right) \\
&= \sum_{j=0}^{\infty} \lambda_\eta\left(\bigcup_{k=0}^{\infty} L_{j,k}\right) \\
&= \sum_{j=0}^{\infty}\sum_{k=0}^{\infty} \frac{|L_{j,k}|}{|\Sigma^k|}\eta(k) \\
&= \sum_{j=0}^{\infty} \lambda_\eta(L_j)
\end{aligned}$$

This shows that λ_η is indeed a measure. □

2.2 Measure 2: Extending the Above

We may generalize λ_η as defined before slightly. Recall the definition, where $L_0, L_1, L_2, \ldots$ again defines a partition of L by size:

$$\lambda_\eta\left(L\right) = \sum_{k=0}^{\infty} \frac{|L_k|}{|\Sigma^k|} \eta\left(k\right)$$

Instead of dividing out by $\left|\Sigma^k\right|$ at each iteration of the sum, we may take a countable series of measures $\nu = \{\nu_0, \nu_1, \nu_2, \ldots\}$, where each ν_k has support on precisely Σ^k. Then, define $\lambda_{\eta,\nu}$ as follows, taking again $L_0, L_1, L_2, \ldots$ the size partition of L:

$$\lambda_{\eta,\nu} = \sum_{k=0}^{\infty} \nu_k\left(L_k\right) \eta\left(k\right)$$

Often it's probably convenient to just assume each $\nu_k \in N$ to be the uniform distribution probability measure, which gets us λ_η as defined above.

Theorem 3. *$\lambda_{\eta,\nu}$ is a measure.*

Proof. We take a similar approach as before, and show (1) measure under empty set is zero and (2) countable additivity, which will show that $\lambda_{\eta,\nu}$ is indeed a measure.

(1) Again, observe that $\lambda_{\eta,\nu}\left(\emptyset\right) = 0$ since the sum will be trivial.

(2) Take $L_0, L_1, L_2, \ldots$ to be a countable collection of pairwise disjoint languages. Implicitly define a countably indexed set, where we take each $L_{j,k}$ as the jth language's sub-language with only strings of length k.

As with before, for $j_1 \neq j_2$ and $k_1 \neq k_2$, every L_{j_1,k_1} and L_{j_2,k_2} are pairwise disjoint. Then, doing the calculation:

$$\begin{aligned}
\lambda_{\eta,\nu}\left(\bigcup_{j=0}^{\infty} L_j\right) &= \lambda_{\eta,\nu}\left(\bigcup_{j=0}^{\infty}\bigcup_{k=0}^{\infty} L_{j,k}\right) \\
&= \sum_{j=0}^{\infty} \lambda_{\eta,\nu}\left(\bigcup_{k=0}^{\infty} L_{j,k}\right) \\
&= \sum_{j=0}^{\infty}\sum_{k=0}^{\infty} \nu_k\left(L_k\right) \eta\left(k\right) \\
&= \sum_{j=0}^{\infty} \lambda_{\eta,\nu}\left(L_j\right)
\end{aligned}$$

$\lambda_{\eta,\nu}$ is therefore a measure.

□

3 Approximating Languages

Given a measure space $(\mathcal{L}, \sigma(L), \lambda)$, we may consider how similar two languages are. For two languages $L_1, L_2 \in \sigma(\mathcal{L})$, a "natural" difference is to consider their symmetric set difference:

$$d(L_1, L_2) = \lambda(L_1 \triangle L_2) = \lambda((L_1 \setminus L_2) \cup (L_2 \setminus L_1))$$

Recall that by the definition of a σ-algebra, the symmetric set difference $L_1 \triangle L_2$ is in $\sigma(\mathcal{L})$, and therefore measurable.

We hope restrict our attention to regular languages for now, or in other words, the class of languages precisely recognized by DFAs, and ask the following:

Question 1. *Given a regular language L recognized by a minimal DFA A and some $\varepsilon > 0$, does there exist a regular language L' recognized by A' such that A' has less states than A, and $d(L, L') < \varepsilon$?*

Example 1. *Consider $\Sigma = \{a\}$, where $L_1 = aa^\star$ and $L_2 = aaa^\star$, and assume a geometric probability measure for η with success of probability $0 < p \leq 1$, through which λ is defined.*

Recall that for the geometric distribution where $n \in \mathbb{Z}_+$:

$$\eta(n) = (1-p)^{n-1} p$$

Observe that for L_1 and L_2, we have:

$$L_1 = \underbrace{\{\}}_{\textit{length } 0} \cup \underbrace{\{a\}}_{\textit{length } 1} \cup \underbrace{\{aa\}}_{\textit{length } 2} \cup \underbrace{\{aaa\}}_{\textit{length } 3} \cup \ldots$$
$$L_2 = \underbrace{\{\}}_{\textit{length } 0} \cup \underbrace{\{\}}_{\textit{length } 1} \cup \underbrace{\{aa\}}_{\textit{length } 2} \cup \underbrace{\{aaa\}}_{\textit{length } 3} \cup \ldots$$

In other words, the only set on which the two languages differ is strings of length 1, which L_1 has, but L_2 does not. For the difference, this then means that:

$$d(L_1, L_2) = \lambda(L_1 \triangle L_2) = \lambda(\{a\}) = \frac{|\{a\}|}{|\Sigma^k|}\eta(1) = \frac{1}{1}p = p$$

In other words, with probability p, we can distinguish random strings generated from L_1 and L_2, where the probability distribution is geometric, and over the length of the strings. Selection of the strings once length is fixed is irrelevant because each set corresponding to a length contains only one string.

Assume a (regular) language L given as an automata or regular expression, whichever may be convenient, and measure λ_n. A few challenges lie ahead:

Question 2. *How can we quickly measure $\lambda_\eta(L)$?*

Question 3. *How can we quickly generate a word of some length from L?*

The word "quickly" here is key: many methods that can be thrown together to answer these questions are intrinsically exponential, so if we can introduce sub-exponential time techniques that would be pretty cool.

3.1 Counting

How many unique strings of length k does an automata have? The question is relatively straightforward for a DFA, and slightly more complicated for a NFA.

Theorem 4. *There exists a polynomial-time algorithm that counts the number of strings accepted by a DFA.*

Proof. Consider the matrix M representation of a DFA $A = (\Sigma, Q, \delta, q_1, F)$, which we claim can be conjured in polynomial time. Roughly, the sketch is that we can enumerate each element of the transition δ and iteratively populate our matrix M.

We define M' where $M'_{i,j} = 1_{M_{i,j} \neq \emptyset}$. In other words, M' is the adjacency matrix corresponding to the directed graph described by A and M.

Let u be the vector where each element is such that $u_i = 1_{q_i \in F}$, then:

$$(1, 0, 0, \ldots)\left(M'\right)^k u$$

Will count the number of strings of length k. The algorithm runs in time polynomial to k the length and $n = |Q|$ the number of states, because matrix multiplication is polynomial in complexity. $\square$

However, counting the number of strings of length k is known to be #**P** [1], and exponential-time algorithms are known: just reduce the NFA to a DFA and run the solution above. Indeed, this is not very satisfying, but at least it's something.

3.2 Approximate Counting

If counting is hard, then perhaps approximate counting is easier? Indeed, this may be the step towards fast comparison of regular languages. To do this, we attempt to adapt work by Kannan [1], which we have cited quite a few times now.

Kannan's construction presents a few "proofs left to the reader", which we complete here. However, we first present a few definitions that are used throughout the paper.

Definition 4 ($g(n)$-randomized approximation scheme)**.** *A $g(n)$-ras for a non-negative real-valued function f is a probabilistic algorithm which, on input x and $\varepsilon > 0$ and $\delta < 1$, computes $\widetilde{f}(x)$ where:*

$$1 - \delta \leq \Pr\left[f(x)(1+\varepsilon)^{-1} \leq \widetilde{f}(x) \leq f(x)(1+\varepsilon)\right]$$

Further, the algorithm runs in expected time $O(g(n))$ where:

$$n = \max\left\{|x|, \varepsilon^{-1}, \log\left(\delta^- 1\right)\right\}$$

In other words, this randomized approximation scheme is pretty tight.

Definition 5 ($g(n)$-almost uniform generator)**.** *Let R be a polynomial-time computable binary relation. For any x, let*

$$\phi(x) = \{y \,:\, (x,y) \in R\}$$

A $g(n)$-almost uniform generator for the relation R is a probabilistic algorithm A, which, on input x, $\varepsilon > 0$, outputs some $y \in \phi(x)$ such that:

$$|\phi(x)|^{-1}(1+\varepsilon)^{-1} \leq \Pr[A(x,\varepsilon) = y] \leq |\phi(x)|^{-1}(1+\varepsilon)$$

Also, A runs in time $O(g(n))$ where:

$$n = \max\left\{|x|, \log\left(\varepsilon^{-1}\right)\right\}$$

Definition 6 (m-level NFA)**.** *An m-level NFA is an NFA in which the states can be partitioned into $m+1$ levels with the following properties:*

(1) There is exactly one state at level 0 and it is the start state.

(2) There is exactly one state at level m and it is the accept state.

(3) All transitions are from a node in level i to a node in level $i+1$ for $i \in \{0,\ldots,m-1\}$.

(4) For any state, p, the accept state is reachable from p, and p is reachable from the start state (fully connected automata).

Kannan further poses the following lemma as an exercise to the reader, which we also prove:

Lemma 1. *For any NFA, M, and integer m, there exists an NFA M' such that M' is an m-level NFA with no ϵ-transitions, a binary alphabet, and such that $|L(M')| = |L(M) \cap \Sigma^m|$. Further, the size of M' is polynomial in size of M and m.*

Proof. Given an NFA M_1, it is well-known that there exist a polynomial-time and polynomial-space reduction to an equivalent NFA M_2 without ϵ-transitions.

As M_2 is presumed to be defined over some alphabet Σ, we now need to reduce this to an equivalent NFA over a binary alphabet Σ_2. To do this, recall from standard results in complexity that we need $\lceil \log(|\Sigma|) \rceil$ bits to represent every element of Σ. Hence, in order to make an ϵ-transition-free and binary alphabet NFA, we may augment every transition edge of M_2 by a sequence of transitions of length $\lceil \log(|\Sigma|) \rceil$ that now uses the binary alphabet Σ_2 instead of Σ. Take $m' = m\lceil \log(|\Sigma|) \rceil$. This produced M_3, which lacks ε-transitions and is over a binary alphabet Σ_2.

To convert the NFA M_3 into a m'-level NFA, we do the following. First, let Q be the set of states of M_3. We create m' copies of Q, and write them as $Q_0, Q_1, \ldots, Q_{m'-1}$. These serve to eventually form the first m'-layers out of the $m'+1$ for an m'-level NFA. For each set of states Q_i, with $i \in \{0,\ldots,m'-2\}$ as follows: if there exists a one-step transition from some q_a to q_b on character $a \in \Sigma_2$ within M_3, then we connect $q_a \in Q_i$ and $q_b \in Q_{i+1}$ using letter a. We add a final layer consisting of just one state q_F, and connect all the final states of $Q_{m'-1}$ to q_F using the technique just described. This creates M_4, which has $m'+1$ layers, the last of which has a single final state.

In order to make M_4 into an m'-level NFA (which has $m'+1$ layers), we need another step of transformation. Take $q_0 \in Q$ to be the start state of M_3. Then, in the first layer, mark q_0 as the start state. Now take the sub-graph of M_4 that is reachable from q_0 in the first layer, and this is the desired m'-level automaton M_5.

Because the start state q_0 is unique, it is clear that the first layer only has a single state. Furthermore, the last layer also has a single state q_F, which we added by construction earlier. In addition, all transition edges by construction flow from layer i to the next layer $i+1$.

Because we have taken the parts reachable from the initial state q_0, we guarantee reachability. It may be the case that the final state q_F is not reachable from q_0, in which case this is fine, and the m'-level NFA cannot be constructed anyways, because it implies that no strings of length m exists in the language. Further note that m' is polynomial in complexity to m. Hence, M_5 is the desired m'-level NFA.

□

3.3 An Algorithm For Measuring

We are now ready to present a quasi-polynomial algorithm for calculating the measure of a language's strings up to some length N, which we describe as follows:

(1) Take as input a NFA M that represents language L, some positive integer N, and error bounds $\varepsilon > 0$ and confidence $\delta < 1$.

(2) Let A be the algorithm specified in Kannan's work [1] that takes as input a m_A-level NFA, error tolerance ε_A, and confidence δ_A.

(3) At each stage, let M_m be the m-level NFA of M, and calculate the following:

$$v = \sum_{k=0}^{N} \frac{A\left(M_k, \varepsilon/N, \delta/N\right)}{|\Sigma^k|} \eta\left(k\right)$$

(4) Return v as an estimate for the measure up to strings of length N.

Theorem 5. *The algorithm runs in quasi-polynomial time with respect to N, $|M|$, ε, and δ.*

Proof. It suffices to show that each iteration of the sum is quasi-polynomial in complexity. Because only N iterations occur, if each iteration is quasi-polynomial, then the entire sum is also quasi-polynomial.

We have shown before that a polynomial-time reduction from M to M_k is possible for each k. Furthermore, this is directly fed into the quasi-polynomial time algorithm A. We implicitly assume η to be polynomial time in complexity. Therefore, each iteration is quasi-polynomial time in complexity.

Together, this means that the entire algorithm is quasi-polynomial time in complexity. □

3.4 Metric Spaces

Lemma 2. *$(A \triangle C) \subseteq (A \triangle B) \cup (B \triangle C)$.*

Proof. Observe that we may rewrite the above as follows:

$$(A \setminus C) \cup (C \setminus A) \subseteq [(A \setminus B) \cup (B \setminus C)] \cup [(B \setminus A) \cup (C \setminus B)]$$

It then suffices to show that:

$$A \setminus C \subseteq (A \setminus B) \cup (B \setminus C) \qquad\qquad C \setminus A \subseteq (B \setminus A) \cup (C \setminus B)$$

We take turns examining these

(i) If $x \in A \setminus C$, then this implies that $x \in A$ and $x \notin C$. There are now two cases, where $x \in B$ or $x \notin B$. First assume that $x \in B$, which will imply that $x \in B \setminus C$. Now assume that $x \notin B$, which will imply that $x \in A \setminus B$. Either way, the implication is that $x \in (A \setminus B) \cup (B \setminus C)$, and so it follows that $A \setminus C \subseteq (A \setminus B) \cup (B \setminus C)$.

(ii) If $x \in C \setminus A$, then this implies that $x \in C$ and $x \notin A$. The argument is similar to the above, in which either $x \in B$ or $x \notin B$. If $x \in B$, then $x \in B \setminus A$, and otherwise if $x \notin B$ implies that $x \in C \setminus B$. Collectively, the two imply that $C \setminus A \subseteq (B \setminus A) \cup (C \setminus B)$.

Collectively, this shows that $(A \triangle C) \subseteq (A \triangle B) \cup (B \triangle C)$. □

Theorem 6. *If (X, Σ, λ) is a measure space, then for $d \colon \Sigma \times \Sigma \to \mathbb{R}_{\geq 0}$ defined as:*

$$d(A, B) = \lambda(A \triangle B)$$

Is a metric function.

Proof. We prove the conditions necessary for a metric: identity, symmetry, and triangle inequality.

(a) As λ is a measure, then for any $A \in \Sigma$:

$$d(A, A) = \lambda(A \triangle A) = \lambda(\emptyset) = 0$$

(b) By the symmetry of symmetric set difference, for any $A, B \in \Sigma$:

$$d(A, B) = \lambda(A \triangle B) = \lambda(B \triangle A) = d(B, A)$$

(c) For any $A, B, C \in \Sigma$, we have by convexity as shown in the lemma above:

$$A \triangle C \subseteq (A \triangle B) \cup (B \triangle C)$$

Then by sub-additivity of measures:

$$d(A, C) = \lambda(A \triangle C) \leq \lambda((A \triangle B) \cup (B \triangle C)) \leq \lambda(A \triangle B) + \lambda(A \triangle C) = d(A, B) + d(B, C)$$

□

References

[1] Sampath Kannan, Z Sweedyk, and Steve Mahaney. “Counting and random generation of strings in regular languages”. In: *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 1995, pp. 551–557.

[2] Dexter Kozen. “A completeness theorem for Kleene algebras and the algebra of regular events”. In: *Infor. and Comput.* 110.2 (1994), pp. 366–390.

[3] John E Savage. *Models of computation*. Vol. 136. Addison-Wesley Reading, MA, 1998.