

Boolean combinations of modular n-gram languages

Matthew Danish
Advisor: Leonid Kontorovich

May 15, 2007

1 Introduction

Modular n-gram languages are a subclass of regular languages. A factor w of a string u is a contiguous string w present within the string u . A modular n-gram language is a set of strings which all contain a certain factor w some $n \pmod k$ times inside. Boolean combinations of languages are the result of combining languages with the boolean operations of union, intersection, and complement: \cup, \cap, \neg .

The characterization of a class of languages entails a computable decision procedure to determine the membership of a language in said class. The procedure could operate on some description of a language, such as a finite-state automaton, a regular expression, or a syntactic monoid.

I show how to construct deterministic finite-state automata which accept modular n-gram languages. Then I show how the syntactic monoid is obtained from that, and discuss several properties of it. Finally, I explore boolean combinations of these languages and find that this subclass is not closed under them.

1.1 Past Work

Schützenberger characterized the star-free languages by showing a language is star-free if and only if its syntactic monoid is aperiodic. Star-free languages are those which can be represented by a regular expression containing no Kleene star.[3]

A discontinuous subsequence w of a string u is one where the individual letters of $w = w_1 w_2 \dots w_m$ may be distributed in u possibly split up and separated by pieces of u . A piecewise testable language L contains

strings with the property that, for some n , whenever $u \in L$ and v have the same discontinuous subsequences of length no larger than n , then $v \in L$. Simon showed that this class of languages is characterized by syntactic monoids that are finite and \mathcal{J} -trivial.[2]

Investigation of the modular n-gram languages is a reasonable step to take in order to learn more about different subclasses of regular languages.

1.2 Notation

Modular n-gram languages with contiguous factors are notated: $K(w, n, k)$ where w is the factor of length l , n is an integer $0 \leq n < k$, and k is the modulus. More formally,

$$K(w, n, k) = \{u \in \Sigma^* \mid w \in \Sigma^* \wedge |u|_w \equiv n \pmod k\}$$

where $|u|_w$ is the number of times the factor w appears in u , overlapping permitted. If no alphabet is specified, it is presumed to be named Σ .

2 Modular n-gram languages

2.1 Construction

The deterministic finite-state automaton that accepts a modular n-gram language $K(w, n, k)$ can be constructed from $|w| \times k$ states. The easiest way to go about it, which appeals to some intuitive notions about the structure, is to place the states into a $|w| \times k$ matrix. The elements of the matrix are denoted $q_{i,j}$ for $1 \leq i \leq |w|$ and $1 \leq j \leq k$.

The first step is to construct the basic backbone: for each row in the matrix j , connect the columns

from $1 \leq i \leq |w|$ with a transition corresponding to the i^{th} letter of w . Notably, the last letter in the word will not be placed in this manner.

At this point it should be clear that the rows encode a certain piece of information about the input, namely: the number of occurrences of w seen so far. It is necessary to handle input which consists of other factors than w . So-called “failure transitions” will backtrack to near the beginning of the current row when the input does not match w . For each row in the matrix j , add the following transitions:

- A transition labeled with $\Sigma \setminus \{w_1\}$ from $q_{1,j}$ to itself.
- For each column i , a transition labeled with $\Sigma \setminus \{w_i, w_1\}$ from $q_{i,j}$ to $q_{1,j}$.
- For each column i , unless $w_i = w_1$, a transition labeled with w_1 from $q_{i,j}$ to $q_{x,j}$ where $x = \min \{i \mid w_i \neq w_1\}$.

The final, and perhaps most important, part is to insert the transitions between rows. This will involve placing a transition labeled with w_l from row j to $j+1$, unless $j+1 > k$. This is where the modulus comes into play. The very last row will connect back up to the very first row. The transition will go from the last state in the current row to a specific state in the next row. The precise state is selected by considering properties of w , namely, how well it overlaps with itself.

- Compute the “overlap index” of the string w which is a non-negative integer, o , representing the minimal index into w at which a new overlapping w could begin after. If no overlap is possible, then the value is simply zero. Example: the index for aba is 2 because an overlap such as $ababa$ begins after $w_2 = b$.
- For each row j , add a transition labeled with w_l from $q_{l,j}$ to $q_{(-o \bmod l)+1, (j \bmod k)+1}$. The overlap index is basically used to specify how far “backwards” to jump when making the cross-row connection.

The start state is $q_{1,1}$, the finish states are all the states in the row $n+1$.

Theorem 1. *This construction gives a complete deterministic finite-state automaton.*

Proof. All states have defined transitions for all inputs. The backbone w_i is present, of course, and the remaining transitions $\Sigma \setminus w_i$ are categorized as “failure transitions” which jump backwards according to the rules above. □

Theorem 2. *This construction correctly specifies a modular n -gram language $K(w, n, k)$.*

Proof. First, if a string contains w some i times then it will cause the machine to transition to the i^{th} row. There are k rows and if $i > k$ then it wraps around to the start. Then the machine will halt once it has reached the $n \equiv i \pmod k$ row.

Second, if a string does not contain w some $n \pmod k$ times, then it will not reach the n^{th} row. The string occurs some j times where $j \not\equiv n \pmod k$. Since that is the case, the transitions cannot wrap around and halt on row n . □

Theorem 3. *This construction gives a minimal deterministic finite-state automaton.*

Proof. The construction indicates that $k|w|$ is a sufficient number of states to create a machine for a modular n -gram language. It is also a necessary number of states. The machine needs to store at least two pieces of information: the current residue, and the current position within the substring w . The number of different possible residues is k , and the length of the word is $|w|$. The only way for a deterministic finite-state automaton to store information is by having a state to represent it. Therefore there must be $k|w|$ states. □

2.2 Examples

See figures 1, 2, and 3 for examples of deterministic finite-state automaton recognizing modular n -gram languages.

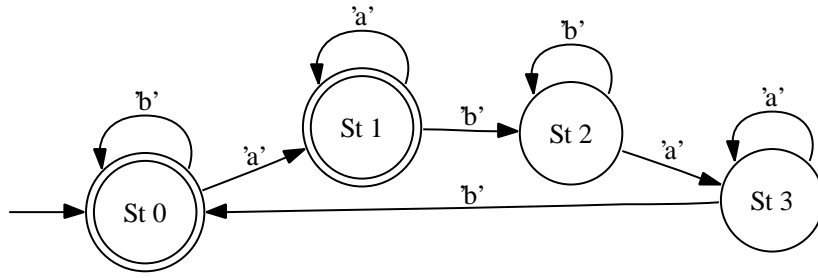


Figure 1: $K(ab, 0, 2)$

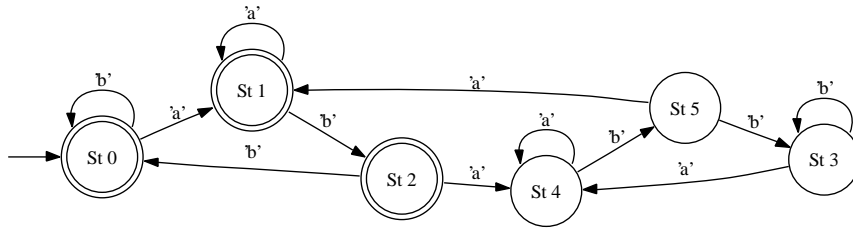


Figure 2: $K(aba, 0, 2)$

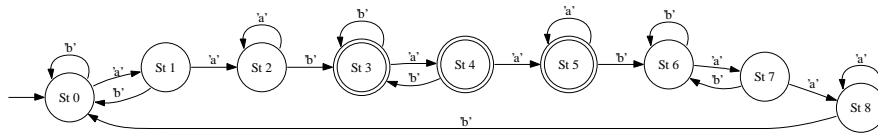


Figure 3: $K(aab, 1, 3)$

2.3 Transition semigroup

Much of the following few sections owes a great deal to the overview provided by Pin in *Syntactic Semigroups*[1]. I have attempted to elaborate from an operational perspective the construction and relevance of the syntactic semigroup, or monoid.

A transition semigroup or monoid for a deterministic finite-state automaton can be formed by a set of matrices closed under matrix multiplication. The matrices in question all have the following form:

$$\begin{bmatrix} m_{1,1} & m_{2,1} & \cdots \\ m_{1,2} & m_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

where $m_{i,j} \in \{0,1\}$ for $1 \leq i \leq |Q|$ and $1 \leq j \leq |Q|$; the number of states in the automaton being $|Q|$.

Each matrix is labeled with a string of symbols, u , from Σ^* . The rows and columns represent states in the automaton.

$$m_{i,j} = \begin{cases} 1 & \text{if } \hat{\delta}(i, u) = j \\ 0 & \text{otherwise} \end{cases}$$

where $\hat{\delta}$ is the “extended transition” function for the automaton: the value of which being the successive application of the δ transition function on consecutive letters in the string u .

The transition semigroup or monoid is necessarily finite, no more than $2^{|Q|^2}$ in size—the number of possible matrices. Typical semigroups will be much smaller.

Construction of the transition semigroup proceeds by this algorithm:

INPUT A deterministic finite-state automaton $\langle Q, \Sigma, q_0, F, \delta \rangle$.

OUTPUT A transition semigroup.

1. Create a matrix for each $\sigma \in \Sigma$, labeled with σ , and encoding the single-step transition from each state in the matrix. Place these in a set called T .
2. Iteratively;

- (a) Let $T' = T \cup \{a \times b \mid a, b \in T\}$. Whenever you multiply matrices, the resulting label is the concatenation of two input labels.
- (b) If $T' \setminus T = \emptyset$ then output T and halt.
- (c) Otherwise set $T = T'$ and loop.

The given algorithm is bounded in time by the finite number of possible matrices with $|Q| \times |Q|$ size. On average, the transition semigroup will be relatively small. With the addition of the identity matrix labeled ε , the semigroup is a monoid.

2.4 Syntactic semigroup

The syntactic semigroup, or monoid, is an alternative view of the transition semigroup—in which the elements are strings and the binary operation is defined by a table specific to the automaton.

Construction of the syntactic semigroup from a transition semigroup is fairly simple. The elements are $\{a_u \mid a \in T\}$ and the table for the binary operation is defined by

$$\{(a_u, b_u) \rightarrow c_u \mid a, b \in T, c = a \times b \in T\}$$

where T is a transition semigroup and a_u is the label of the matrix $a \in T$. A transition monoid with an identity matrix will result in a syntactic monoid with an empty element ε . Table 1 has an example of a syntactic monoid constructed for $K(ab, 0, 2)$.

2.5 Characterization

Properties of the regular language can be determined by examining the syntactic monoid for patterns.

2.5.1 Idempotency

An element u is called idempotent if and only if $u^2 = u$. In the case of figure 1, the element a is idempotent because $a^2 = a$.

2.5.2 Aperiodicity

An element u is called aperiodic if and only if there exists some integer n such that $u^n = u^{n+1}$. Clearly, idempotent elements are also aperiodic. There are

no examples of other aperiodic elements in figure 1 but there are several examples of periodic elements. Let $u = ab$ then $u^2 = abab$ and $u^3 = ab$. This series of powers alternates endlessly between those two strings.

Some modular n -gram languages do, in fact, have aperiodic generators.

Conjecture 1. *Consider modular n -gram language $K(w, n, k)$ where $w = w_1 \dots w_l$. If $l > 2$ then $|w_2 w_{l-1}|$ has non-zero length. The substrings of $w_2 w_{l-1}$, and itself, will appear as aperiodic generators in the syntactic monoid of $K(w, n, k)$.*

3 Boolean combinations

Languages L_1 and L_2 are sets and can be operated upon using set operations such as union ($L_1 \cup L_2$), intersection ($L_1 \cap L_2$), and complement ($\neg L_1$). These are called the Boolean combinations of languages. As it turns out, in general, the class of modular n -gram languages is not closed under these operations.

Theorem 4. *The class of modular n -gram languages is not closed under intersection.*

Proof. Consider $K(ab, 1, 2)$ and $K(cd, 1, 2)$. Taking $\Sigma = \{a, b, c, d\}$, the strings $abcd$ and $cdab$ are both contained in the intersection. The only contiguous substrings shared in common are ab and cd , and they cannot be combined into a single substring because there is no constraint on the ordering. Therefore there is no modular n -gram language containing these strings and excluding improper strings. \square

Even though theorem 4 shows that intersection is not closed in general, there is an interesting subset for which it is. When w is held constant over a subclass of modular n -gram languages for which the moduli are relatively prime, the problem regarding substrings disappears and then it is possible to construct a new modular n -gram language from the intersection.

Theorem 5. *The subclass of modular n -gram languages given by $\{K(w, n, k) \mid k \in \mathbb{N}, 0 \leq n < k\}$, for some w where k is relatively prime to the modulus of*

the other languages in the subclass, can form intersections which are modular n -gram languages.

Proof. $K(w, n_1, k_1) \cap K(w, n_2, k_2)$ can be described by the set of congruences

$$\begin{aligned} c &\equiv n_1 \pmod{k_1} \\ c &\equiv n_2 \pmod{k_2} \end{aligned}$$

which can be solved by the Chinese Remainder Theorem for a unique $0 \leq c < k_1 k_2$. Then, the resulting language is $K(w, c, k_1 k_2)$. \square

Theorem 6. $\neg K(w, n, k) = \bigcup_{m \neq n} K(w, m, k)$

Proof. If there is a string $u \in K(w, n, k)$ then it is known that $|u|_w \equiv n \pmod{k}$. There is no other residue m for which this is true: $|u|_w \not\equiv m \pmod{k}$ for all $m \neq n$. Therefore, $u \notin \bigcup_{m \neq n} K(w, m, k)$.

Similarly, if there is a string $u \in \bigcup_{m \neq n} K(w, m, k)$ then $|u|_w \not\equiv n \pmod{k}$ and $u \notin K(w, n, k)$. \square

Corollary 1. $\bigcup_{0 \leq m < k} K(w, m, k) = \Sigma^*$

Proof. This follows from above. For all strings $u \in \Sigma^*$ the $|u|_w$ must be congruent to some residue modulo k . Therefore, the union of all the languages $K(w, m, k)$ contains all strings.

This can also be seen by noting that $m \pmod{k}$ is an equivalence class of natural numbers under congruence, and all of the natural numbers are contained in one of k such equivalence classes $0, \dots, k-1 \pmod{k}$. \square

Theorem 7. *The class of modular n -gram languages is not closed under the operations of union or complement.*

Proof. Take $K(ab, 0, 3)$ and $K(ab, 1, 3)$. Call L the union of these two. For all strings $u \in L$, $|u|_{ab}$ is congruent to either 0 or 1 modulo 3. There is no single modular n -gram language equivalent to L because a modular n -gram language can only accept one equivalence class modulo k .

L can also be described as $\neg K(ab, 2, 3)$ which shows that the complement of $K(ab, 2, 3)$ is not a modular n -gram language. \square

4 Conclusion

I spent the first half the semester getting very familiar with the formalisms underlying the regular languages. I also spent a fair bit of time actually implementing the algorithms for regular expression, non-deterministic and deterministic automata conversions, minimization of the latter, and the construction of the syntactic monoid, boolean combinations, as well as helpful visualizations. The programming experience, while somewhat orthogonal to the aim of the project, was incredibly helpful to developing a good intuitive feel for the theory. For example, once I had the software tools, it became trivial to test the counter-examples I discovered for the above proofs. I also found the operational understanding of transition matrices to be a great boon. And attempting to formally prove the correctness of my algorithms uncovered quite a few bugs and misconceptions.

So while the modular n-gram languages themselves turn out to be a bit of a disappointment, I feel that the process of exploring them had significant benefit itself.

References

- [1] Pin, J.-E. *Syntactic Semigroups*. Handbook of Formal Language Theory I (G. Rozenberg and A. Salomaa, eds.), Springer, 1997.
- [2] Simon, Imre. *Piecewise Testable Events*. Proceedings of the 2nd GI Conference on Automata Theory, pp 214-222. Springer-Verlag, 1975.
- [3] Schützenberger, M.P. *On Finite Monoids Having Only Trivial Subgroups*. Information and Control 8, pp 190-194. 1965.

| | ε | a | b | ab | ba | aba | bab | abab | baba |
|---------------|---------------|------|------|------|------|------|------|------|------|
| ε | ε | a | b | ab | ba | aba | bab | abab | baba |
| a | a | a | ab | ab | aba | aba | abab | abab | a |
| b | b | ba | b | bab | ba | baba | bab | b | baba |
| ab | ab | aba | ab | abab | aba | a | abab | ab | a |
| ba | ba | ba | bab | bab | baba | baba | b | b | ba |
| aba | aba | aba | abab | abab | a | a | ab | ab | aba |
| bab | bab | baba | bab | b | baba | ba | b | bab | ba |
| abab | abab | a | abab | ab | a | aba | ab | abab | aba |
| baba | baba | baba | b | b | ba | ba | bab | bab | baba |

Table 1: Syntactic monoid of $K(ab, 0, 2)$