15-354: CDM **K. Sutner**

Assignment 4 Due: **Sep. 29, 2023, 24:00**.

1. Primitive Words (30)

Background

Let w be a non-empty word. A word z is the root of w if $w \in z^\star$ but there is no shorter word with this property. A word is primitive if it is its own root.

For example, ab is the root of $abababab$. The primitive words of length 3 over $\{a, b\}$ are aab, aba, abb, baa, bab, bba.

Task

A. Show that any two non-empty words u, v that commute, $uv = vu$, have the same root.

B. Show that any two non-empty words u and v commute iff they have equal powers: $u^k = v^\ell$ for positive k and l.

C. Show that a non-empty word w is primitive iff $w^2 = xwy$ implies that $x = \varepsilon$ or $y = \varepsilon$.

D. Count the number of primitive words of length n over a k letter alphabet.

E. Show that the language of primitive words over an alphabet of size at least 2 is not regular.

F. As part of some algorithm it is necessary to store a few thousand binary words of length 20. Prof. Dr. Wurzelbrunft suggests to speed things up by storing the roots of words, rather than the words directly. What professional advice can you give Wurzelbrunft?

Comment

Möbius inversion might be helpful for part (D).

It is conjectured that primitive words are not even context-free, but that is currently an open problem. Extra credit.

Solution: Primitive Words

Part A: Commutativity

By induction on $|uv|$.

For $|uv| = 2$ the claim is obvious, so assume $|uv| > 2$. Also assume without loss of generality (check that this is really true!) $|v| < |u|$. But then $uv = vu$ implies $u = vu_0$ where $|u_0| < |u|$. Note that $u_0 v = vu_0$, so by induction we may assume that for some w we have $u_0, v \in w^\star$. Our claim follows.

Part B: Powers

Commuting words have the same root, so by part (A) our claim follows.

On the other hand, suppose $u^k = v^\ell$. WLOG $u = vx$. Then $(vx)^k = v^\ell$, hence also $(xv)^k = v^\ell$ and thus $(vx)^k = (xv)^k$. But then x and v commute, done by part (A).

Part C: Primitivity

If w fails to be primitive it is easy to violate the condition on the right hand side: let $w = u^r$ where $r \geq 2$. Then $ww = u\, w\, u^{r-1}$, done.

So suppose w is primitive but $ww = xwy$ where $x, y \neq \varepsilon$. Then $w = w_- w_+ = w_+ w_-$ where $0 < |w_-| = |x| < |w|$. By part (A) w is not primitive.

Part D: Root Count

Clearly any word w of length $n > 0$ is either primitive or can be written as $w = u^r$ for some $r \geq 2$. Of course, r is uniquely determined. But then

$$k^n = \sum_{d|n} \pi(d)$$

By Möbius inversion we get

$$\pi(n) = \sum_{d|n} \mu(d)\, k^{n/d}.$$

where μ is the Möbius function. Good enough to compute $\pi(n)$ for any reasonable value of n.

Part E: Irregular

Consider all strings of the form $ab^i ab^j$. The strings form a regular language L and a string in L is primitive iff $i = j$. Suppose there is DFA recognizing the language P of primitive words. Then $P \cap L = \{\, ab^i ab^i \mid i \geq 0 \,\}$ is also regular. Letting i be the number of states in a DFA for this language easily produces a contradiction.

Part F: Wurzelbrunft

As is usually (though not always!) the case, Wurzelbrunft is clueless: the fraction of imprimitive binary words of length 20 is about 0.999012, so only a handful of the "thousands" of words would be any shorter.

2. Word Binomials (40)

Background

By a subsequence of a word $v = v_1 v_2 \dots v_m$ we mean any word $u = v_{i_1} v_{i_2} \dots v_{i_r}$ where $1 \leq i_1 < i_2 < \dots i_r \leq m$ is a strictly increasing sequence of indices. Thus *bbc* and *cab* are subsequences of *ababacaba* but *cbb* is not.

Note that a specific word can occur multiple times as a subsequence of another. For example, *aab* appears 7 times in *ababacaba*. We write

$$\binom{v}{u} = C(v,u) = \text{ number of occurrences of } u \text{ as a subsequence of } v.$$

The notation is justified since "word binomials" generalize ordinary binomial coefficients: $\binom{n}{k} = \binom{a^n}{a^k}$. Note that instances of u as a subsequence of v in general overlap, e.g., $C(a^3, a^2) = 3$.

Task

Let $\delta_{a,b} = 1$ iff $a = b$, 0 otherwise, be the Kronecker delta; $a, b \in \Sigma$ and $u, v, u_i, v_i \in \Sigma^\star$.

A. Show that

$$\binom{va}{ub} = \binom{v}{ub} + \delta_{a,b}\binom{v}{u}$$

B. Show that

$$\binom{v_1 v_2}{u} = \sum_{u=u_1 u_2} \binom{v_1}{u_1}\binom{v_2}{u_2}$$

C. Give an efficient algorithm to compute word binomials.

D. Give a simple description (in terms of union, concatenation and Kleene star) of the language

$$L = \{\, v \in \{a,b\}^\star \mid C(v, ab) = 3 \,\}$$

E. Construct the minimal DFA for L by diagram chasing (aka doodling).

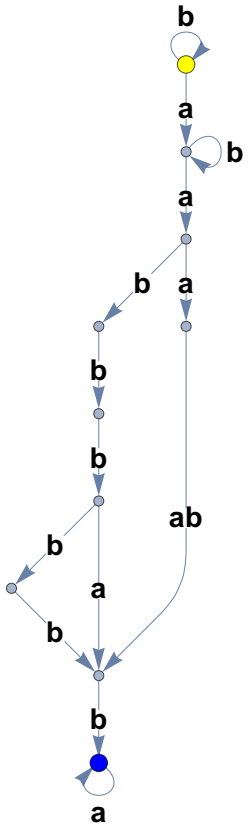
F. Generalize: given a word u and an integer r construct a DFA that accepts

$$L(u,r) = \{\, v \in \Sigma^\star \mid C(v,u) = r \,\}$$

Is your machine always minimal?

Comment

For what it's worth, here is a picture of the smallest possible DFA checking for 6 subwords *aab*. Make sure you understand how this machine works. Your construction will probably produce a much larger machine–but one that is also much easier to describe than this minimal one.



Solution: Word Binomials

Part A: Recursion

For ub to be a subsequence of va it must either be a subsequence of v alone or u must be a subsequence of v provided that $a = b$.

Part B: Summation

For u be subsequence of $v_1 v_2$ there has to be a factorization $u = u_1 u_2$ such that u_i is a subsequence of v_i.

Part C: Algorithm

We can use the recursion in part (A) to compute word binomials by dynamic programming. To this end, let $n = |x|$ and $m = |y|$ (we may assume $m \leq n$).

We construct an $(n+1)(m+1)$ table of integers T such that

$$T(i,j) = \binom{x_1 \dots x_i}{y_1 \dots y_j}$$

for $0 \leq i \leq n$ and $0 \leq j \leq m$. Thus $T(n,m) = \binom{x}{y}$ is desired result. Initialize $T(i,0) = 1$, $T(0,j) = 0$ for $j > 0$ and use

$$T(i,j) = T(i-1,j) + \delta_{x_i,y_j} T(i-1,j-1).$$

to fill in the table. The running time is clearly $O(nm)$.

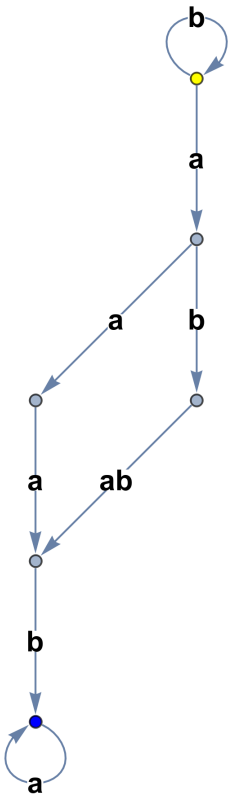
Part D: Sample Language

It is not hard to see that $\binom{v}{ab} = 3$ implies that $\binom{x}{ab} = 3$ for all $x \in b^\star v a^\star$, so we only need to look for words of the form $v = aub$. A little thought shows that the only choices are $v = aaab, abab, abbb$.

Thus $L = b^\star\{aaab, abab, abbb\}a^\star$.

Part E: Minimal DFA for Sample

The obvious DFA checks for those three words and ignores initial b's and final a's:



The DFA is minimal since the distance between the initial and final state cannot be less than 4 and we must have $\delta(q_0, aa) \neq \delta(q_0, ab)$.

Part F: General DFA

The idea is to use part (A). Let $n = |u|$ and write $u(i) = u_1 \dots u_i$ for the prefix of u of length i. We use as state set

$$Q = [0, r+1]^n,$$

the initial state is $q_0 = (0, \dots, 0)$ and the final states are of the form $(\dots, r)$. Define the transition function by

$$\delta(\boldsymbol{p}, a)_i = \max(p_i + \Delta_{a,v_i} p_{i-1}, r+1)$$

where $p_0 = 0$ and Δ denotes the Kronecker delta.

An easy induction on x then shows that

$$\delta(q_0, x) = \left(\binom{x}{u(1)}, \binom{x}{u(2)}, \dots, \binom{x}{u(n)}\right)$$

Hence the automaton works properly.

In a serious implementation one would only construct the accessible part of this machine, starting at q_0 and applying δ to obtain the appropriate closure.

Even the accessible part is in general not minimal. For example, for the sample language from above we get a DFA with 21 states (rather than the potential maximum of 25). But the minimal DFA has only 7 states in this case.

3. Semilinear Counting (30)

Background

It is often stated that "finite state machines cannot count." To a point, this is correct, but there are special cases when a finite state machine is perfectly capable of counting. Here are some fairly involved examples of counting in zero space.

Recall that a set $C \subseteq \mathbb{N}$ is semilinear if it is a finite union of sets of the form $t + p\,\mathbb{N}$, where $t, p \in \mathbb{N}$ (for $p = 0$ this is just $\{t\}$) (transient and period). Let $L_C = \{\, 0^\ell \mid \ell \in C \,\} \subseteq 0^\star$.

Let $U \subseteq \Sigma^+$ be a regular language. A U-factorization of $x \in \Sigma^+$ is a sequence $u_1, \ldots, u_\ell$ of words in U such that $x = u_1 \ldots u_\ell$, $\ell \geq 1$. Write $\mathsf{fac}(x, U)$ for the set of all U-factorizations of x and define

$$L(U, C) = \{\, x \mid |\mathsf{fac}(x, U)| \in C \,\}$$

Thus, $L(U, C)$ collects all words that have exactly ℓ many U-factorizations where $\ell \in C$.

Task

A. Construct the minimal automaton for L_C.

B. Conclude that the semilinear sets form a Boolean algebra.

C. Show that $L(U, C)$ is regular.

Comment For (A), make sure your automaton is really minimal. For the last part, you probably want to use a pebbling argument and closure properties. Try $C = \{3\}$ first, then $C = \mathsf{evens}$.

Solution: Semilinear Counting

Part A: Minimal

Lasso.

Part B: Boolean

Follows directly from (A) and the fact that regular languages form a Boolean algebra.

Part C: Factorizations

Since C is semilinear and regular languages are closed under union, it suffices to handle the special case $C = t + p\,\mathbb{N}$. We can also exclude $C = \emptyset$ and $C = \mathbb{N}$ (why?).

Suppose $\mathcal{A}$ is a DFA for U. First assume $p = 0$, so we need to check for t factorizations. Place a pebble on the initial state of $\mathcal{A}$ and move as usual until a pebble arrives at a final state; in this case, place another pebble on the initial state. The machine accepts whenever the total number of pebbles on the final states is t. Here is the trick: if the number of pebbles on a single state grows beyond t, truncate it to $t + 1$, indicating failure. This guarantees finitely many pebble configurations and thus produces a finite state machine.

If $p > 0$, we count "modulo t and p" in the sense that we keep up to $t + p - 1$ pebbles at a single state; if we reach $t + p$, we reset to t. Essentially, we are counting on the lasso with transient t and period p. Again we get a finite state machine.