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

Assignment 4 Due: **Sep. 27, 2024, 24:00**.

## 1. Word Binomials (40)

**Background**

By a subsequence or subword 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. In other words, we can erase a few letters in $v$ to get $u$. 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**

Recall the Kronecker delta defined by $\delta_{a,b} = 1$ iff $a = b$, 0 otherwise. Let $a, b \in \Sigma$ 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

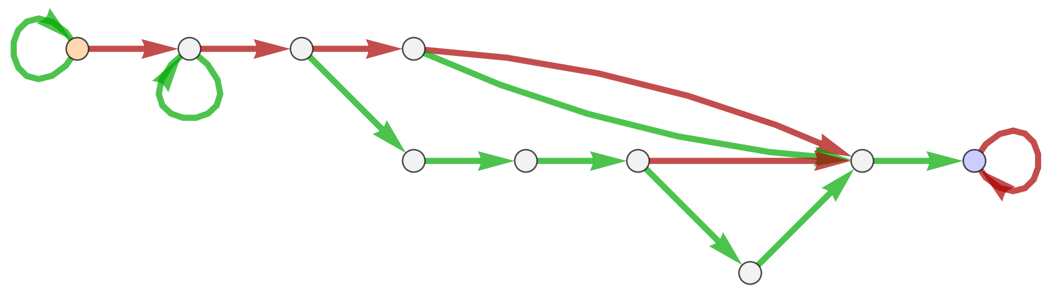
$$L(u, r) = \{ v \in \Sigma^\star \mid C(v, u) = r \}$$

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.

subword aab 6-count



## 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

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

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

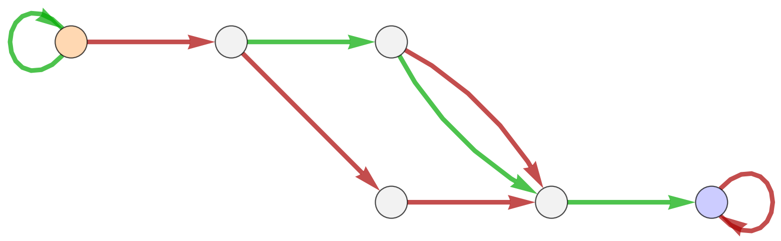
It is not hard to see that $\binom{v}{ab} = 3$ implies that $\binom{x}{ab} = 3$ for all $x \in b^\star v a^\star$, so we only need to look for words of the form $v = aub$. A little thought shows that the only choices are $v = aaab, abab, abbb$.

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:

subword ab 3-count



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

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

where $p_0 = 0$ and $\mathit{\Delta}$ 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 $\delta$ to obtain the appropriate closure. Not unexpectedly, even the accessible part is in general not minimal. For example, for the sample language, we get an accessible DFA with 21 states rather than the potential maximum of 25; the trim part has 11 states, but the minimal PDFA has only 6 states.

## 2. Semilinear Counting (30)

**Background**

It is often stated that "finite state machines cannot count." To a point, that 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

$$C_{t,p} = t + p\,\mathbb{N} = \{\, t + i\,p \mid i \geq 0 \,\}$$

where $t, p \in \mathbb{N}$; for $p = 0$ this is just the singleton $\{t\}$ (think of transient and period). Let $L_C = \{\, 0^\ell \mid \ell \in C \,\} \subseteq 0^\star$, the numbers in $C$ written in unary.

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

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

Thus, $L(U, C)$ collects all words that have exactly $\ell$ 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

First off, the minimal DFA for any tally language has the shape of a lasso with some transient $t$ and some period $p$. In particular the minimal DFA for $C_{t,p}$ has transient $t$ and period $p$; the only final state is the anchor where the transient and cycle meet.

For the unions, we use the standard product construction that produce yet another lasso automaton. For such a lasso DFA to be minimal, we need two conditions. First, think of the final states on the cycle as being given by a binary word $u \in \mathbf{2}^p$. Then $u$ must be primitive (not a repetition of a shorter word, the so-called root): otherwise we could replace the loop by a shorter one based on the root of $u$. Second, consider the state $p$ where the transient meets the loop (if there is none, we are already done) and let $p_1$ and $p_2$ be the two predecessors. Then we must have $p_1 \in F \oplus p_2 \in F$: otherwise, they could be merged.

**Part B:** Boolean

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

**Part C:** Factorizations

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

Suppose $\mathcal{A}$ is a DFA for $U$. Here is the argument for $p \geq 1$, the case $p = 0$ is easier and left as an exercise. We need a convenient way to express the position of a speed-1 particle on a lasso with transient $t$ and period $p$.

$$s \bmod t{:}p = \begin{cases} s & \text{if } s < t, \\ t + (s - t \bmod p) & \text{otherwise.} \end{cases}$$

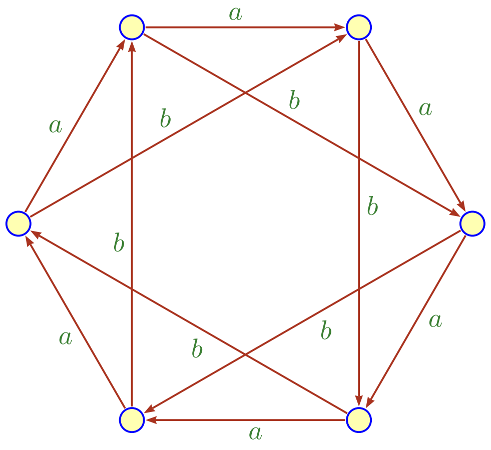
For $t = 0$ this is just the ordinary mod. We place a pebble on the initial state of $\mathcal{A}$ and move as usual except that two pebbles on a single state do not simply merge, they are counted modulo $\bmod t{:}p$. When a pebble arrives at a final state, another pebble is placed on the initial state. The machine accepts whenever the total number $N$ of pebbles on the final states satisfies $t = N \bmod t{:}p$.

Since the total number of pebbles placed in $\mathcal{A}$ is bounded by $(t + p - 1)|\mathcal{A}|$, we still have a finite state machine and in fact a DFA.

## 3. Blowup (30)

**Background**

Write $\mathcal{A}_n$ for the (boring) automaton on $n$ states whose diagram is the circulant with $n$ nodes and strides 1 and 2. The edges with stride 1 are labeled $a$ and the edges with stride 2 are labeled $b$. For example, the following picture shows $\mathcal{A}_6$. We assume $I = F = Q$.



Let $\mathcal{B}_n$ be the (interesting) automaton obtained from $\mathcal{A}_n$ by switching one of the $b$ labels to an $a$ label; write $K_n$ for the acceptance language of $\mathcal{B}_n$.

**Task**

A. Show that determinization of $\mathcal{B}_n$ produces an accessible automaton $\mathcal{B}'_n$ of $2^n$ states.

B. Argue that $\mathcal{B}'_n$ is already reduced and conclude that $K_n$ has state complexity $2^n$.

**Comment**

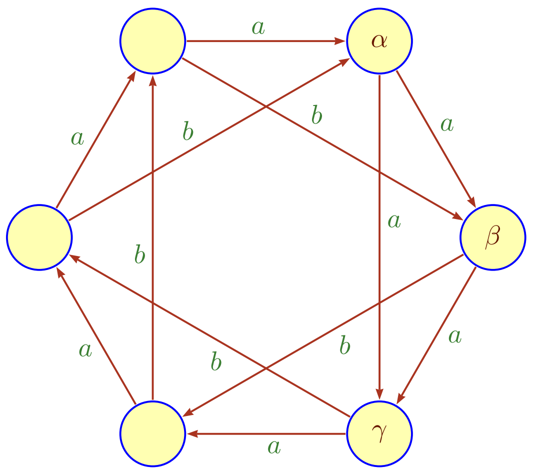
The language $K_n$ has no particular significance (as far as I know). Thinking about pebble automata might help with the argument.

**Extra credit:** If you switch an $a$ to a $b$, there is still full blow-up for odd $n$, but for even $n$ the power automaton has only size $2^n - 1$.

## Solution: Blow-Up

**Part A:** Powerautomaton

For simplicity we omit subscripts and write $\mathcal{B}$ for the NFA. We refer to the three critical states as $\alpha$, $\beta$ and $\gamma$ as in the following picture:



As to intuition, action $a$ produces a "funky rotation": all the pebbles are rotated, but then an additional pebble is placed on $\gamma$ if there was a pebble on $\alpha$. Action $b$ removes a pebble from $\alpha$, and permutes the other pebbles.

**Suspicion:** One should be able to remove pebbles arbitrarily while preserving other pebbles.

Here is a more formal version of this idea. Let us call a set $P \subseteq Q$ persistent if $\bigcap_{p \in P} [\![p]\!]_{\mathcal{B}} - [\![Q - P]\!]_{\mathcal{B}} \neq \emptyset$. Thus, there is a word in the behavior of all $p \in P$, but not in the behavior of any $q \notin P$.

**Claim 1:** All non-empty $P \subseteq Q$ are persistent.

*Proof.* Place a black pebble on each state in $P$, and a red pebble on all the other states. If a black pebble sits on $\alpha$, we fire an $a$, but we underestimate the result: assume the pebble does not split and just moves to $\beta$. The advantage of this underestimate is that it avoids collisions and tedious bookkeeping. So, we pretend that action $a$ induces a plain cyclic rotation on black pebbles. Otherwise, everything moves according to the standard pebbling rules.

As long as there is a red pebble somewhere, let $i$ minimal such that $\delta(P, a^i)$ has a red pebble on $\alpha$. Then fire a letter $b$. This reduces the number of red pebbles, but does not affect the number of black ones. □

Let $\mathcal{B}^{\mathsf{op}}$ be the reversal of $\mathcal{B}$.

**Claim 2:** For all $P \subseteq Q$: $P$ is persistent iff $P$ is reachable in $\mathcal{B}^{\mathsf{op}}$ from $Q$.

*Proof.* Assume $P$ is persistent and let $x$ be a corresponding word as in claim 1. Then $\delta^{\mathsf{op}}(Q, x^{\mathsf{op}}) = P$.

On the other hand, if $\delta^{\mathsf{op}}(Q, w) = P$, then $w^{\mathsf{op}}$ shows that $P$ is persistent.

Here comes the nasty trick: $\mathcal{B}$ and $\mathcal{B}^{\mathsf{op}}$ are isomorphic. Done by claims 1 and 2.

**Part B:** Minimality

Let $P_1 \neq P_2 \subseteq Q$, say, $p \in P_1 - P_2$. From Claim 1 above it follows that $\{p\}$ is persistent. Since $P_2 \subseteq Q - \{p\}$ we are done.